Libro de animación vectorial

Aprende construyendo escenas; consulta la API al final

PYTHON · VELLO · BEVY · GPU

GUÍA DE LECTURA

Índice general

1. Instalación 5
2. El playground 9
3. Tu primera animación 21
4. Cómo piensa Gaanim 25
5. Antes de empezar 33
6. Primera escena 35
7. Objetos y estilo 39
8. Animar el tiempo 43
9. Componer y explicar 47
10. Dar vida a la escena 51
11. Del círculo a la onda 55
12. Terminar el proyecto 59
13. Layout 65
14. Movimiento 79
15. Personajes con SVG por partes 89
16. Tipografía cinética 93
17. Transiciones 99
18. Efectos 105
19. Reactividad 111
20. Cámara y 3D 119
21. Presentaciones 127
22. Ensayar y grabar una charla 137
23. Proyectos y exportación 145
24. Compartir sin Python 149
25. Capturas y comparación visual 161
26. Público en vivo 165
27. Tu primera encuesta 167
28. Anatomía de un juego 171
29. Preguntas 175
30. Zonas vivas 177
31. Recetario de estilos 181
32. Presentar con público 191
33. Examples 197
34. Referencia de la API 209
35. Escena 211
36. Drawable 271
37. Geometría 321
38. Texto 369
39. Layout 399
40. Visualización 415
41. Matrices 465
42. Medios 473
43. Audio 483
44. Recursos 503
45. Diapositivas 509

46. Mecánica 519
47. Público 535
48. Animaciones 545
49. Temas y colores 655
50. Línea de comandos 691
51. Manifiesto gaanim.toml 707
52. Novedades 711
53. Glosario 761
54. Solución de problemas 765

PARTE I

Empezar

Instala Gaanim, crea tu primera animación y aprende cómo piensa una escena

CAPÍTULO

1. Instalación

Instala Gaanim en Windows o Ubuntu, crea un proyecto y resuelve los problemas habituales

1.1. En resumen

Si ya tienes Python 3.14 y uv:

1. Descarga el paquete de tu sistema desde la última release.
2. Extráelo y pon la carpeta de los ejecutables en el `PATH` (Windows o Ubuntu).
3. Abre una terminal nueva y crea un proyecto:

```
gaanim init video mi-video   # crea el proyecto y su entorno con uv
gaanim mi-video              # abre la vista previa
```

Si algo falla, ve a Solución de problemas. El resto de la página explica cada paso con detalle.

1.2. Requisitos

	Windows	**Ubuntu**
Sistema	Windows 10/11 x64	Ubuntu 24.04 x64
Python	3.14 o posterior	3.14 exactamente
uv	necesario	necesario
GPU	controladores actuales (Vulkan o DirectX 12)	Vulkan
FFmpeg	opcional	opcional

- **Python** ejecuta tus escenas. Instálalo desde python.org, con `winget install Python.Python.3.14` (Windows) o, en cualquier sistema, con `uv python install 3.14`. En Ubuntu tiene que ser 3.14 porque el motor enlaza `libpython3.14.so`.
- **uv** crea el entorno de cada proyecto.
- **FFmpeg** solo hace falta para exportar video (MP4, WebM, MOV, WebP animado y GIF) y para `scene.media.video()`. Sin él tienes la vista previa completa y puedes exportar secuencias PNG.

macOS todavía no tiene paquete. Para compilar Gaanim desde el código, sigue el README del repositorio.

1.3. Descarga

Cada versión se publica en las releases de GitHub con un paquete por sistema y su suma SHA-256. No necesitas Rust ni compilar nada.

Sistema	Archivo	Contenido
Windows 10/11 x64	`gaanim-v<versión>-windows-x64.zip`	`gaanim.exe` y sus bibliotecas, wheel de autoría
Ubuntu 24.04 x64	`gaanim-v<versión>-linux-x64.tar.gz`	`gaanim` y sus bibliotecas, wheel de autoría

- `gaanim` es el único ejecutable: inicio, vista previa, presentaciones, reproductor de paquetes `.gaanim` y línea de comandos.
- Lo acompañan tres bibliotecas: el motor (`gaanim_engine`), el soporte de Python (`gaanim_python_plugin`), que `gaanim` carga solo al abrir una escena o un proyecto, y la biblioteca estándar de Rust (`std-…`). Deja siempre todos los archivos del paquete en la misma carpeta.
- El wheel `gaanim-<versión>-py3-none-any.whl` es el paquete de autoría: tipos, stubs y ayudas para que tu editor autocomplete la API. No contiene el renderer, así que `python main.py` falla a propósito: las escenas siempre se ejecutan con `gaanim`.

1.3.1. Verifica la descarga

Descarga también el archivo `.sha256` de tu paquete y compara las sumas:

```
# Windows (PowerShell): los dos valores deben coincidir
Get-FileHash .\gaanim-v*-windows-x64.zip -Algorithm SHA256
Get-Content .\gaanim-v*-windows-x64.zip.sha256
```

```
# Ubuntu: imprime "OK" si coincide
sha256sum -c gaanim-v*-linux-x64.tar.gz.sha256
```

1.4. Instala en Windows

1. Extrae el zip en una carpeta, por ejemplo `C:\Tools\gaanim`. El wheel debe quedar junto a `gaanim.exe`: ahí lo busca.
2. Añade esa carpeta al `PATH` de tu usuario, desde *Configuración* ⟶ *Variables de entorno* ⟶ `Path`, o con PowerShell:

```
Expand-Archive .\gaanim-v*-windows-x64.zip -DestinationPath C:\Tools\gaanim
$userPath = [Environment]::GetEnvironmentVariable("Path", "User")
[Environment]::SetEnvironmentVariable("Path", "$userPath;C:\Tools\gaanim", "User")
```

3. Abre una terminal nueva para que tome el `PATH` actualizado.
4. Opcional: `gaanim register` abre los archivos `.gaanim` con doble clic y muestra su portada en el Explorador (ver Compartir sin Python).

`gaanim.exe` arranca aunque Python no esté en el `PATH`: solo lo busca cuando abres una escena o un proyecto.

1.5. Instala en Ubuntu

Extrae el paquete en `~/.local/lib/gaanim`, con el wheel junto al ejecutable, donde `gaanim` lo busca, y enlaza el ejecutable desde `~/.local/bin`:

```
mkdir -p ~/.local/lib/gaanim ~/.local/bin
tar -xzf gaanim-v*-linux-x64.tar.gz -C ~/.local/lib/gaanim
ln -sf ~/.local/lib/gaanim/gaanim ~/.local/bin/gaanim
~/.local/bin/gaanim register   # opcional: doble clic y portadas de los .gaanim
```

Ubuntu añade `~/.local/bin` al `PATH` si la carpeta existe al iniciar sesión. Si acabas de crearla y `gaanim` no aparece, cierra la sesión y vuelve a entrar.

1.6. Comprueba la instalación

```
gaanim --version   # imprime la versión instalada
gaanim --help      # lista los comandos
```

Ninguno de los dos necesita Python. Si funcionan, los ejecutables están bien instalados; el siguiente paso comprueba Python y uv.

1.7. Crea un proyecto

```
gaanim init video mi-video      # video 16:9
gaanim init slides mi-charla    # presentación con pasos
gaanim init video .             # en la carpeta actual
```

`init` genera la estructura del proyecto y, con uv, un entorno `.venv` con Python 3.14 y el wheel de autoría:

```
mi-video/
  gaanim.toml       # nombre, tipo, archivo de entrada y carpetas
  main.py           # escena de ejemplo: edítala
  assets/           # imágenes, SVG, fuentes, audio
  exports/          # destino sugerido para las exportaciones
  pyproject.toml    # proyecto uv con Python 3.14
  .python-version
  README.md
  AGENTS.md         # instrucciones para asistentes de código
  .gitignore
  .venv/            # entorno creado con uv
```

Si la carpeta ya contiene archivos del proyecto, `init` se detiene para no pisarlos. Con `--force` reescribe esos archivos (incluido `main.py`) sin tocar tus assets.

También puedes crear y abrir proyectos sin terminal: `gaanim` sin argumentos abre el Inicio, con los proyectos recientes y un diagnóstico de Python y uv.

1.7.1. Usa el proyecto

Estos son los comandos del día a día. La primera animación los usa paso a paso.

```
gaanim mi-video                                  # vista previa; guardar recarga
gaanim check mi-video                            # ejecuta la escena sin ventana
gaanim check mi-video --strict                   # falla también con avisos
gaanim export mi-video --output mi-video/exports/demo.mp4
gaanim --present --monitor 1 mi-charla           # presenta en el monitor 1
```

Dentro de la carpeta del proyecto, `gaanim .` abre su archivo de entrada. También puedes abrir un script suelto con `gaanim escena.py`.

Tu editor autocompleta la API si usa el intérprete de `mi-video/.venv`.

1.8. Cómo encuentra Python

No hace falta activar el entorno. Al abrir una escena, `gaanim` usa el primer intérprete que encuentra:

1. El entorno activo (`VIRTUAL_ENV`).
2. Una carpeta `.venv`, `venv` o `env` junto al proyecto o al directorio actual, subiendo hasta cuatro niveles.
3. El Python del sistema: `py -3.14`, `python` o `python3` en Windows; `python3.14`, `python3` o `python` en Ubuntu.
4. El Python 3.14 que haya instalado uv.

Los pasos 1 y 2 no siguen buscando si encuentran otra versión: un entorno con Python 3.12 produce un error aunque tengas 3.14 instalado.

1.9. Actualiza

Cuando sale una versión nueva, la pantalla de inicio lo avisa: **What's new** abre su página y **Update** la instala sobre la carpeta de Gaanim; al terminar, **Restart** abre la versión nueva. Desde la terminal hace lo mismo:

```
gaanim update --check   # solo dice si hay una versión nueva
gaanim update           # la descarga y la instala
```

Gaanim descarga el paquete de tu sistema, comprueba su suma SHA-256, lo descomprime junto a la instalación, verifica que arranca y solo entonces reemplaza los archivos; si algo falla, deja los anteriores. Ver `gaanim update`.

A mano también se puede: descarga la release nueva y reemplaza todos los archivos de la carpeta, porque el ejecutable y sus bibliotecas deben venir de la misma versión. En ningún caso hace falta recrear los proyectos: al abrir uno, Gaanim reinstala su paquete de autoría si no coincide con la versión del ejecutable.

1.10. Solución de problemas

- **`Python >=3.14 was not found`** o **`Python 3.14 was not found`**: instala Python 3.14 (`uv python install 3.14`) y vuelve a intentarlo.
- **`Gaanim requires Python … but found Python 3.x`**: el entorno activo o el `.venv` del proyecto usa otra versión. Desactívalo, o recrea el entorno con `uv venv --python 3.14`.
- **`gaanim_engine.dll`** o **`std-….dll` no encontrado, o salida `-1073741515` (Windows)**: faltan archivos del paquete junto a `gaanim.exe`. Extrae de nuevo el zip completo en la misma carpeta.
- **`… belongs to another build of Gaanim`** o **`could not load …gaanim_python_plugin…`**: los archivos de la carpeta son de versiones distintas. Reemplázalos todos con los de una misma release.
- **`authoring environment not ready`**: falta uv, o el wheel no está junto a `gaanim` (en Ubuntu, en `~/.local/lib/gaanim`). La vista previa funciona igual, pero el editor no autocompleta.
- **`gaanim check: could not load project`**: revisa que `entry` en `gaanim.toml` sea una ruta relativa a un archivo que exista.
- **La exportación dice que `ffmpeg` no está en el `PATH`** (`was not found on PATH`): instala FFmpeg y añádelo al `PATH`, o exporta una secuencia PNG (`--output frames/frame.png`).

CAPÍTULO

2. El playground

Escribe y ejecuta escenas de Gaanim en el navegador, sin instalar nada: proyectos, recursos, paquetes, vídeo y publicación

El playground es Gaanim en el navegador. Escribes Python en un editor con autocompletado, lo ejecutas y ves la escena al lado, sin instalar Python ni Gaanim. Es la forma más rápida de probar una idea, de seguir el tutorial o de enseñar una animación a alguien que no tiene la aplicación.

Todo ocurre en tu navegador: Python corre en Pyodide (Python 3.14 compilado a WebAssembly) y la escena se dibuja con WebGPU. Tus proyectos se guardan en el navegador y no se envían a ningún servidor.

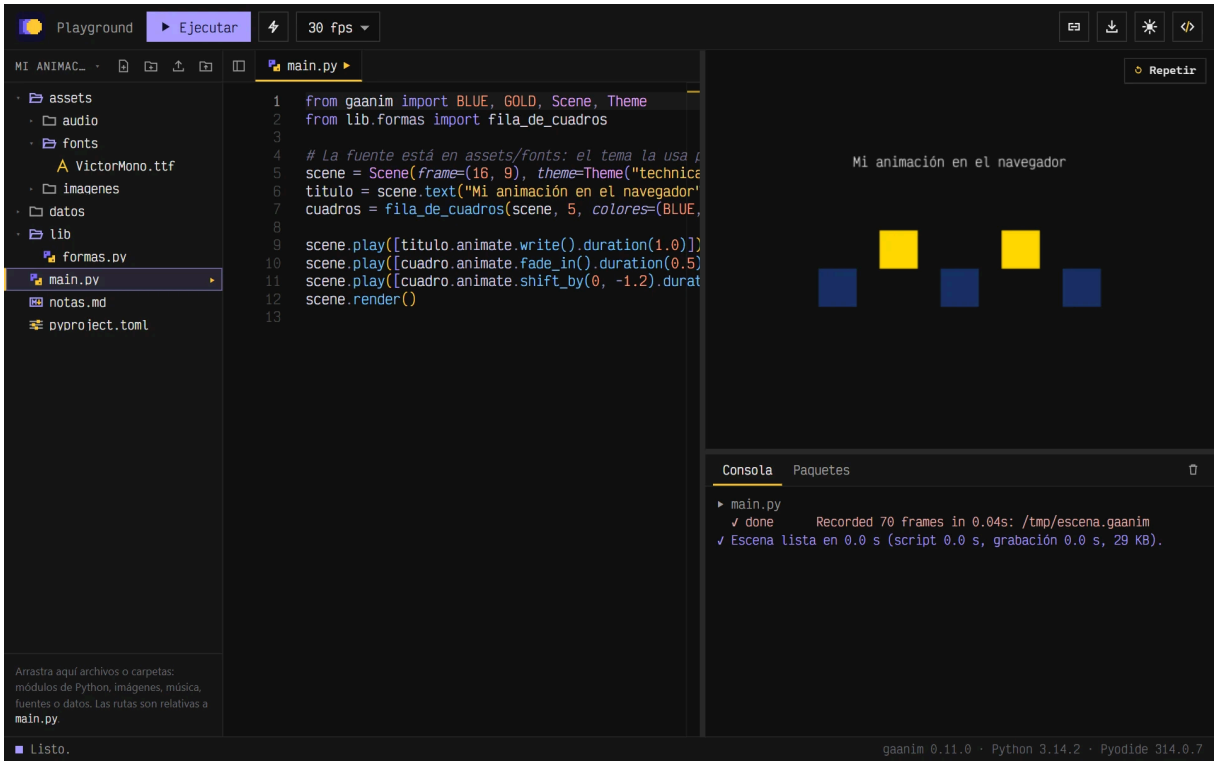


Figura 1: El explorador del proyecto, el editor, la escena grabada y la consola.

2.1. La interfaz

La barra de arriba tiene, de izquierda a derecha:

Control	Qué hace
Ejecutar	Ejecuta `main.py` y graba la escena. `Ctrl+S` guarda y ejecuta; `Ctrl+Enter` ejecuta.
Rayo (**En vivo**)	Ejecuta la escena cada vez que dejas de escribir, si el código no tiene errores de sintaxis.

fps	Fotogramas por segundo de la grabación: 24, 30 o 60.
Enlace (**Compartir**)	Copia un enlace con los archivos de texto del proyecto.
Flecha (**Descargar**)	Descarga un vídeo o GIF, la grabación `.gaanim` o el proyecto en un `.zip`.
Sol o luna	Cambia entre el tema claro y el oscuro.
`</>` (**Insertar**)	Prepara la escena para publicarla en un blog o un sitio web.

Debajo están el **explorador** del proyecto, el **editor** con una pestaña por archivo abierto, el **reproductor**, que muestra la última grabación, y la **consola**, con lo que imprime `print` y los errores. Los separadores entre paneles se arrastran, y el botón de la esquina del editor oculta el explorador. En una pantalla estrecha, el explorador se abre por encima del editor.

2.2. Proyectos

En el playground cada trabajo es un *proyecto*: una carpeta con `main.py`, que es el archivo que se ejecuta, y los módulos, imágenes, audio, fuentes o datos que quieras. El explorador funciona como el de un editor de código:

- Los botones de su cabecera crean un archivo o una carpeta y suben archivos o carpetas enteras. También puedes arrastrarlos desde tu escritorio al explorador, o soltarlos en cualquier parte de la página.
- Arrastra un archivo o una carpeta sobre otra carpeta para moverlo.
- El menú contextual (clic derecho) abre, renombra (`F2`), descarga o elimina (`Supr`). Eliminar se puede deshacer durante unos segundos.

El nombre del proyecto, arriba del explorador, abre el menú de proyectos:

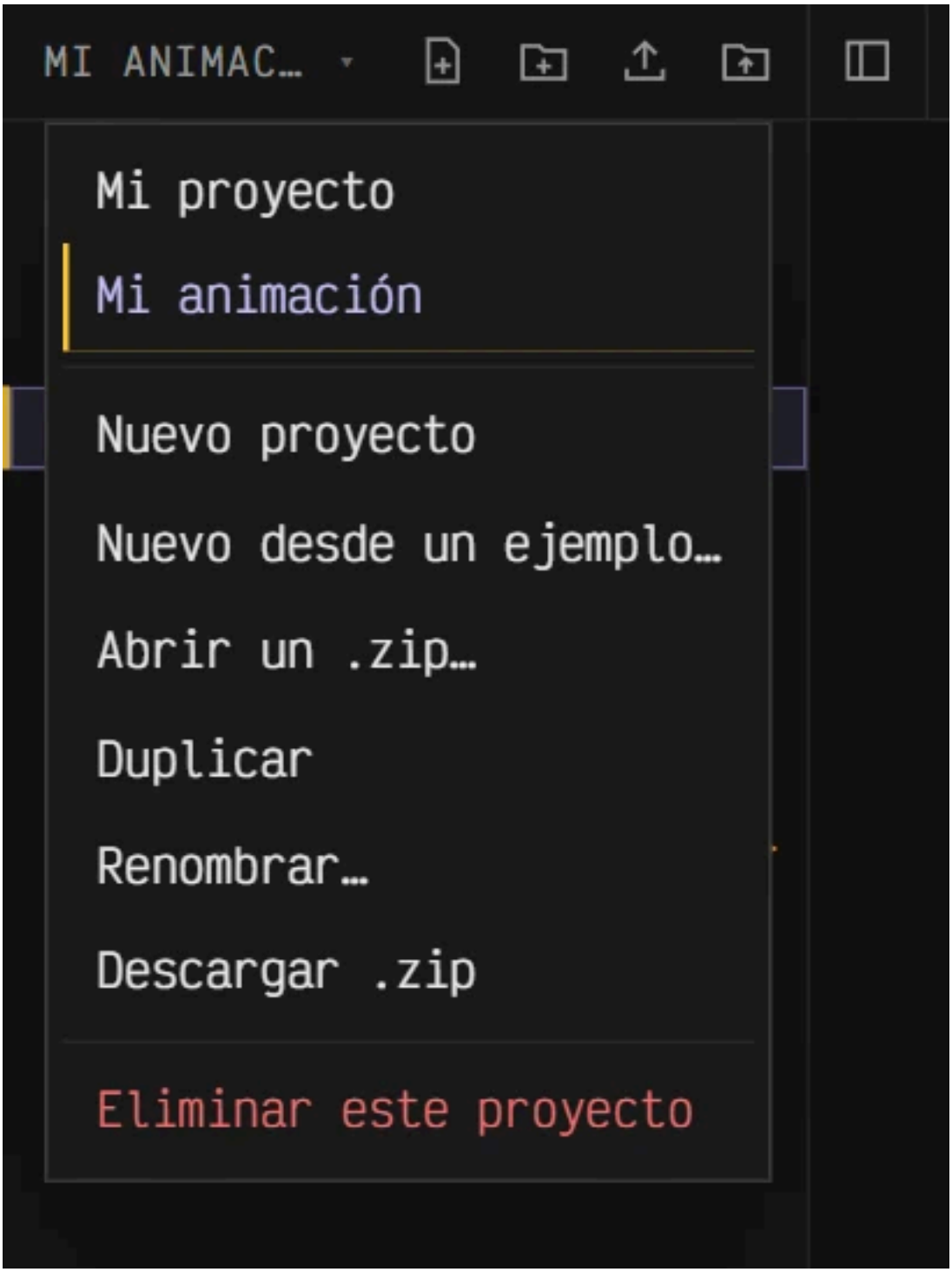


Figura 2: Cambiar de proyecto, crear uno, abrir un `.zip`, duplicar, renombrar, descargar o eliminar.

- **Nuevo proyecto** empieza con un `main.py` de ejemplo.
- **Nuevo desde un ejemplo…** abre una escena de la galería como proyecto propio, sin tocar los que ya tienes.
- **Abrir un .zip…** convierte un archivo ZIP en un proyecto: uno descargado del playground o la carpeta comprimida de un proyecto de la aplicación. Si todo está dentro de una carpeta, su contenido es el proyecto.
- **Duplicar**, **Renombrar…** y **Descargar .zip** hacen lo que dicen. **Eliminar este proyecto** también se puede deshacer.

Los proyectos se guardan en el almacenamiento de tu navegador: siguen ahí cuando vuelves, pero solo en ese navegador y ese equipo. Si borras los datos del sitio, se pierden; descarga el `.zip` de los que quieras conservar.

2.3. Módulos, imágenes, audio y fuentes

`main.py` importa los módulos del proyecto como en tu equipo: con `lib/formas.py`, escribe `from lib.formas import fila_de_cuadros`. Cada ejecución los vuelve a leer, así que un cambio en un módulo se ve en la siguiente.

Las rutas de los recursos son relativas a `main.py`, igual que en la aplicación: `scene.media.image("assets/imagenes/logo.png")` o `scene.media.audio("assets/audio/tema.wav")`. Al abrir una imagen, un audio o una fuente en el explorador, el editor la muestra y propone el código que la usa.

El navegador solo trae las fuentes que vienen con Gaanim: New Computer Modern, Libertinus Serif, DejaVu Sans Mono y JetBrains Mono, la del código. Para usar otra, sube su archivo `.ttf` u `.otf` a una carpeta del proyecto y nómbrala en el tema. La vista previa de la fuente dice el nombre de familia que hay que escribir:

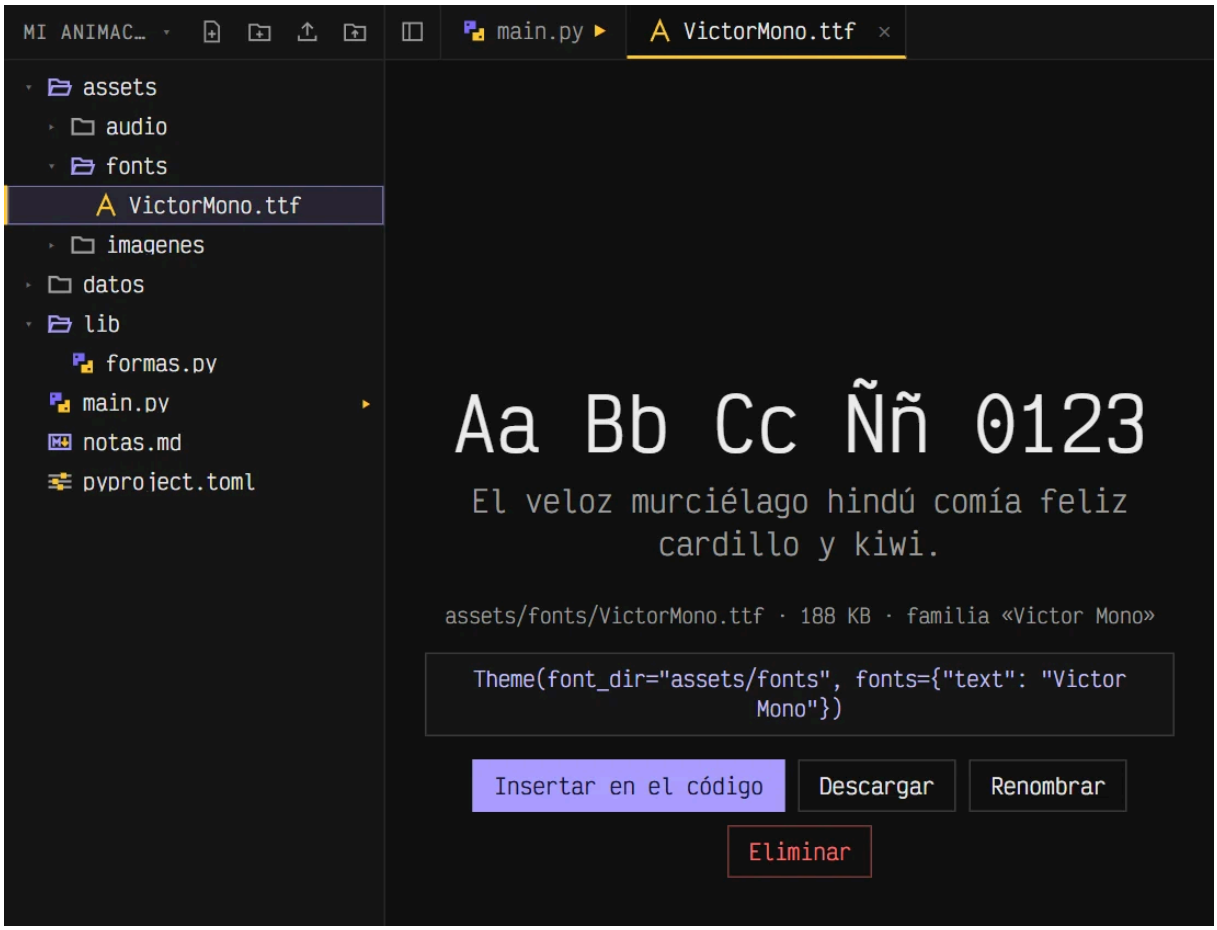


Figura 3: Una fuente en `assets/fonts`, con su familia y el código que la usa.

```
from gaanim import Scene, Theme

scene = Scene(theme=Theme("technical", font_dir="assets/fonts", fonts={"text": "Victor Mono"}))
scene.text("Con mi fuente")
scene.render()
```

Si una escena nombra una fuente que el navegador no tiene, la consola lo advierte y explica cómo añadirla. Los vídeos como recurso (`scene.media.video`) necesitan FFmpeg y solo funcionan en la aplicación.

2.4. Paquetes de Python

La pestaña **Paquetes**, junto a la consola, añade paquetes al proyecto. Escribe un nombre y elige entre las sugerencias: los paquetes que Pyodide compila para el navegador (NumPy, SymPy, SciPy, pandas, NetworkX…) y el paquete con ese nombre en PyPI. **Añadir** (o `Enter`) lo escribe en el proyecto y lo instala en el momento, sin ejecutar la escena; la lista muestra la versión instalada de cada uno, y la × lo quita del proyecto. Una versión escrita con el nombre, como `rich>=13`, se conserva.

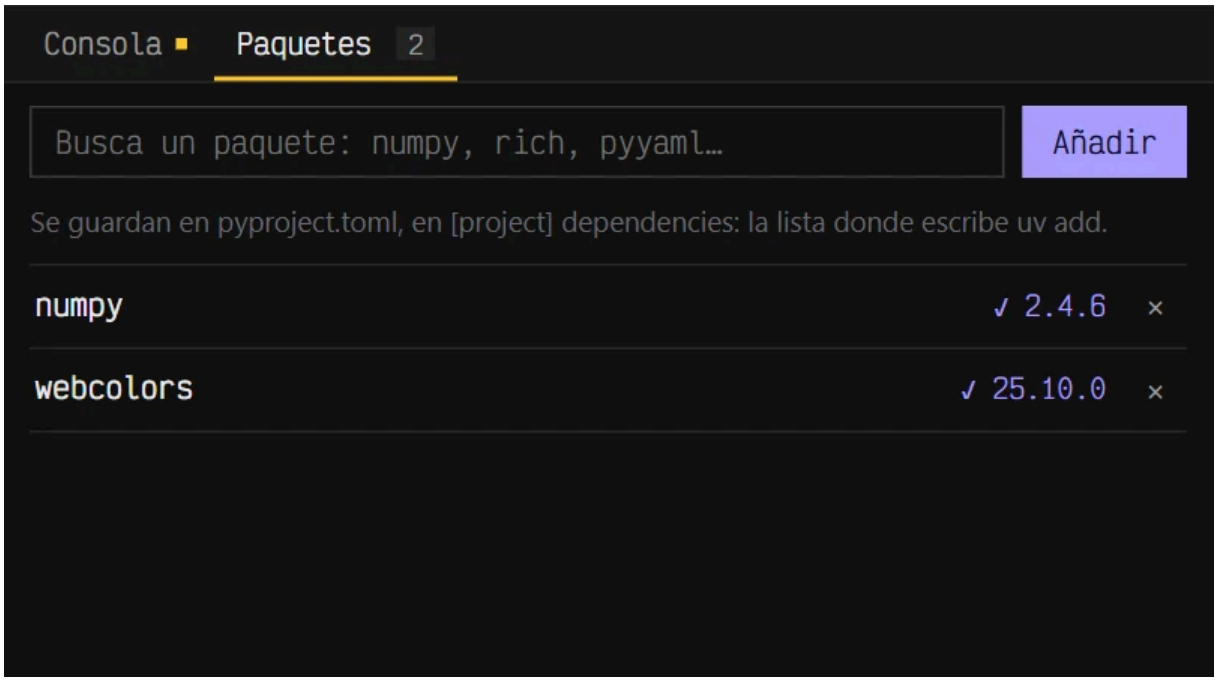


Figura 4: `numpy`, de Pyodide, y `webcolors`, de PyPI, en el `pyproject.toml` del proyecto.

Los paquetes se guardan como en cualquier proyecto de Python. Si el proyecto tiene un `pyproject.toml` con la lista `dependencies` de la tabla `[project]`, como los que crea `gaanim init`, van allí: es la lista donde escribe `uv add`. Si no, van a `requirements.txt`, un paquete por línea. Puedes editar esos archivos a mano; el panel los sigue. `gaanim` no hace falta declararlo, porque el playground ya lo tiene.

Al ejecutar, el playground instala con micropip lo que falte. Los paquetes de Pyodide se cargan también solos cuando el código los importa, aunque no estén en la lista. El editor completa los paquetes instalados igual que la API de Gaanim. Si un `import` falla con `ModuleNotFoundError`, la consola ofrece un enlace que añade el paquete. A veces se llama distinto de su `import`: `import yaml` viene de `pyyaml`.

En el navegador solo se instalan los paquetes de Python puro (con una rueda `py3-none-any`) y los que compila Pyodide, y lo mismo vale para sus dependencias. El panel avisa de los que no se pueden instalar y dice por qué: un paquete que no está en PyPI, o que necesita código nativo, falla sin detener la escena, y **Reintentar** lo intenta otra vez. Esos paquetes funcionan en la aplicación. Instalar necesita conexión; lo instalado sigue disponible mientras no detengas Python.

2.5. Del playground a la aplicación, y de vuelta

Un proyecto del playground y uno de la aplicación tienen la misma forma: `main.py` con rutas relativas a su carpeta. Para pasar de uno a otro:

- **Del playground a tu equipo.** Descarga el `.zip` del proyecto y descomprímelo. Si el proyecto tiene `pyproject.toml`, `uv sync` instala sus dependencias. Si usa `requirements.txt`, crea el proyecto con `gaanim init video mi-video`, copia los archivos y añade los paquetes con `uv add -r requirements.txt`. Luego `gaanim mi-video` abre la vista previa.
- **De tu equipo al playground.** Comprime la carpeta del proyecto y ábrela con **Abrir un .zip…**. Las dependencias de su `pyproject.toml` se instalan al ejecutar; `.venv`, `uv.lock` y los archivos ocultos no hacen falta.

```
uv add -r requirements.txt   # pasa los paquetes del playground al pyproject.toml
uv sync                      # instala las dependencias de un proyecto descargado
```

2.6. Ejecutar, detener y corregir

El reproductor muestra la última grabación. Se detiene en cada `scene.stop()` hasta un clic o `Espacio`, y si la pausa tiene un bucle (`loop=`), lo repite mientras espera, como al presentar; **Continuar** sigue desde el final del bucle. El vídeo exportado no se detiene y reproduce cada bucle una vez.

Mientras la escena corre, el reproductor dice si se está ejecutando el script o grabando la escena, cuánto lleva grabado y cuánto tiempo ha pasado:

Figura 5: Una grabación en curso. El reproductor sigue mostrando la anterior.

Detener (o `Esc`) para la ejecución. Como Python no se puede interrumpir a mitad de un cálculo, el playground lo reinicia. Si la ejecución duraba un rato, ya tenía otro Python cargándose en segundo plano, y vuelves a tenerlo listo casi de inmediato.

Un error escribe su traceback en la consola, abre el archivo donde ocurrió (también dentro de un módulo) y marca la línea. Cada `File "…", line N` de la consola es un enlace a ese punto:

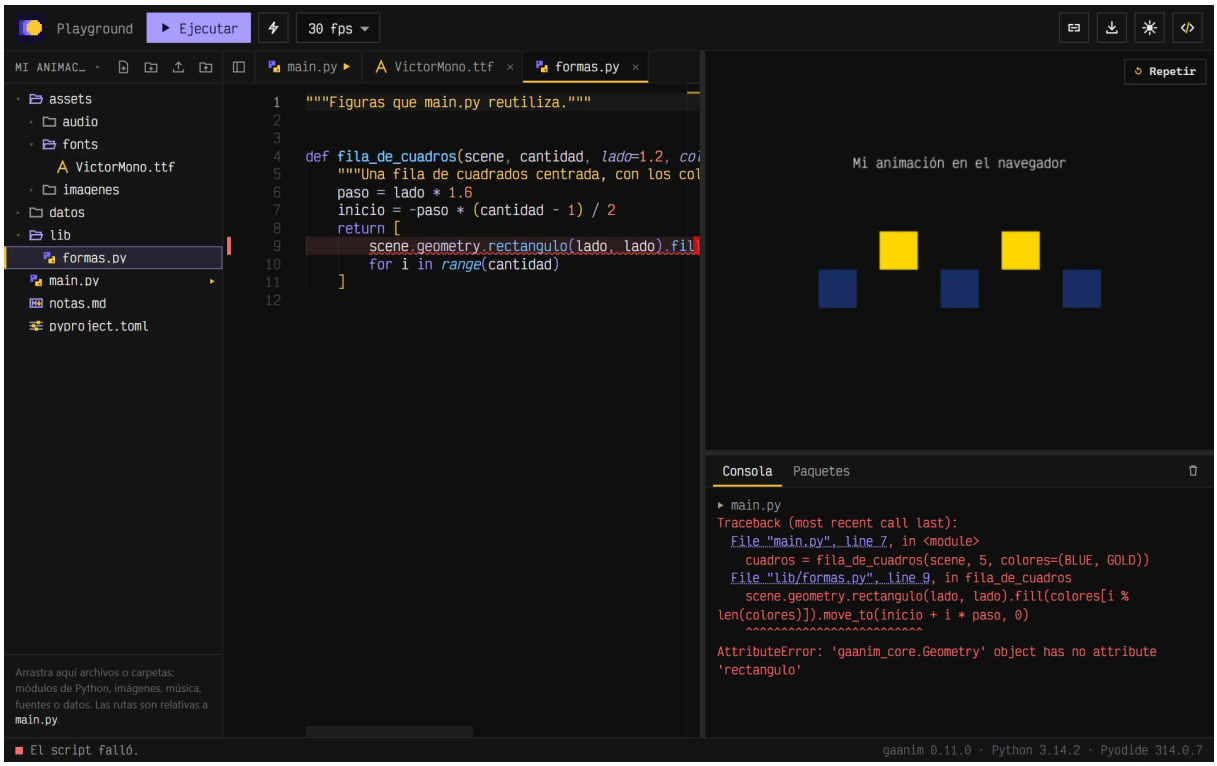


Figura 6: Un error en `lib/formas.py`: el archivo se abre en la línea que falló.

El editor también marca los errores de sintaxis mientras escribes. Con **En vivo** activado, la escena se ejecuta sola cuando dejas de escribir y el código no tiene errores de sintaxis; si una ejecución ya está en marcha, la siguiente espera a que termine.

2.7. Compartir y descargar

Compartir copia un enlace que lleva los archivos de texto del proyecto comprimidos en la propia dirección: quien lo abre recibe una copia como proyecto nuevo. Las imágenes, el audio y las fuentes no caben en un enlace; para compartirlos, descarga el `.zip`.

Descargar ofrece cuatro cosas:

- **Vídeo o GIF…** convierte la última grabación en MP4 (H.264), WebM (VP9) o GIF, a 480p, 720p, 1080p o 4K. La conversión usa los codificadores del navegador y el archivo no sale de tu equipo. El MP4 y el WebM llevan el audio de la escena si el navegador sabe codificarlo; si no, el diálogo avisa de que va sin sonido. El GIF va sin sonido, con hasta 25 fotogramas por segundo y 256 colores.
- El `.gaanim` de la grabación, que se reproduce, presenta y exporta con la aplicación sin Python (ver Compartir sin Python).
- Una página `.html` con la grabación y el reproductor dentro, que se abre con doble clic, sin servidor (ver Sin sitio donde publicarla).
- El proyecto en un `.zip`.

Figura 7: Exportar vídeo, con el tamaño y los fotogramas del resultado.

El vídeo del navegador no dibuja el posprocesado ni los fondos, transiciones y efectos de shader; el diálogo avisa cuando la escena los usa. Para un vídeo con todo, o con la calidad de producción, expórtalo con la aplicación.

2.8. Insertar en un blog o sitio web

Insertar prepara la escena para publicarla: descarga el `.gaanim` con el nombre que elijas y compone el `<iframe>` del reproductor ligero, con la dirección donde lo publicarás y la proporción de la escena. Al lado muestra cómo se verá. Si no tienes dónde publicarlo, debajo de la vista previa descarga en su lugar una página HTML con la animación y el reproductor dentro. Los detalles y las opciones están en Insertar en un blog o sitio web.

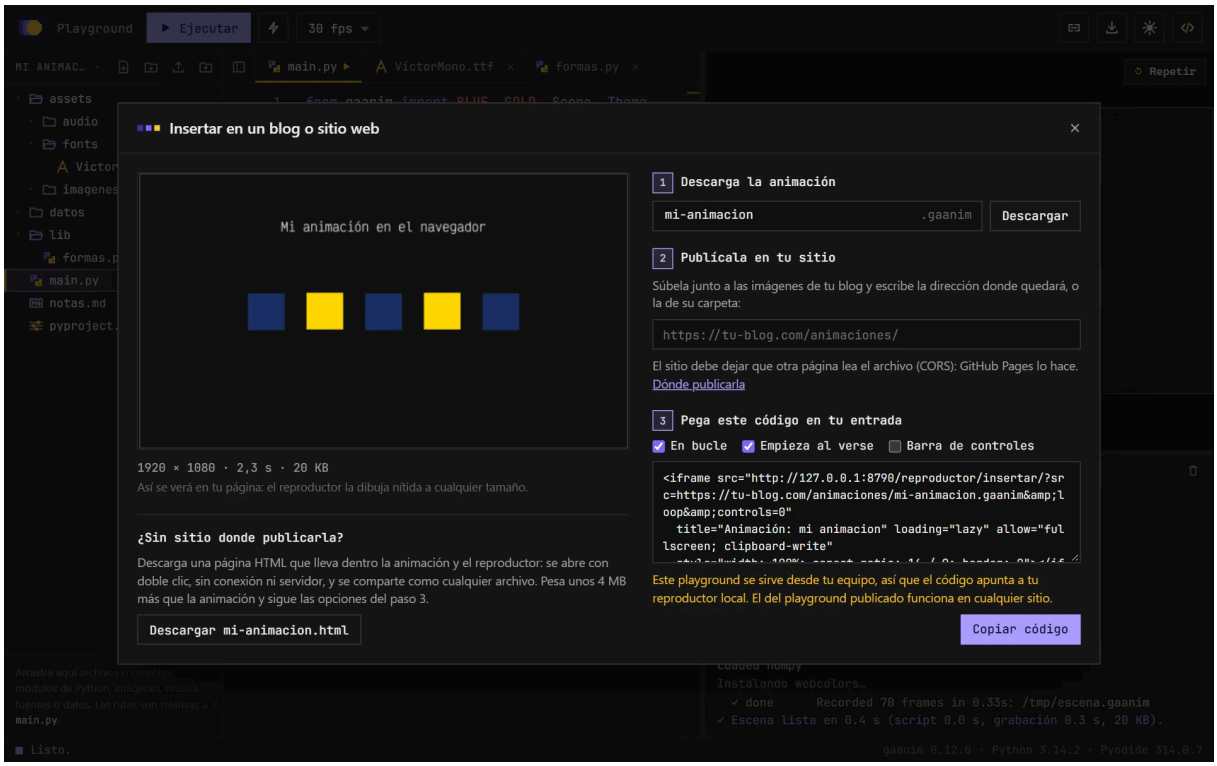


Figura 8: Insertar: el nombre del archivo, su dirección, las opciones y el código; bajo la vista previa, la página HTML.

2.9. Tema claro y oscuro

El playground sigue el tema de tu sistema. El botón del sol o la luna lo cambia y la elección se recuerda; es la misma que la del botón de esta documentación, así que cambiar uno cambia el otro.

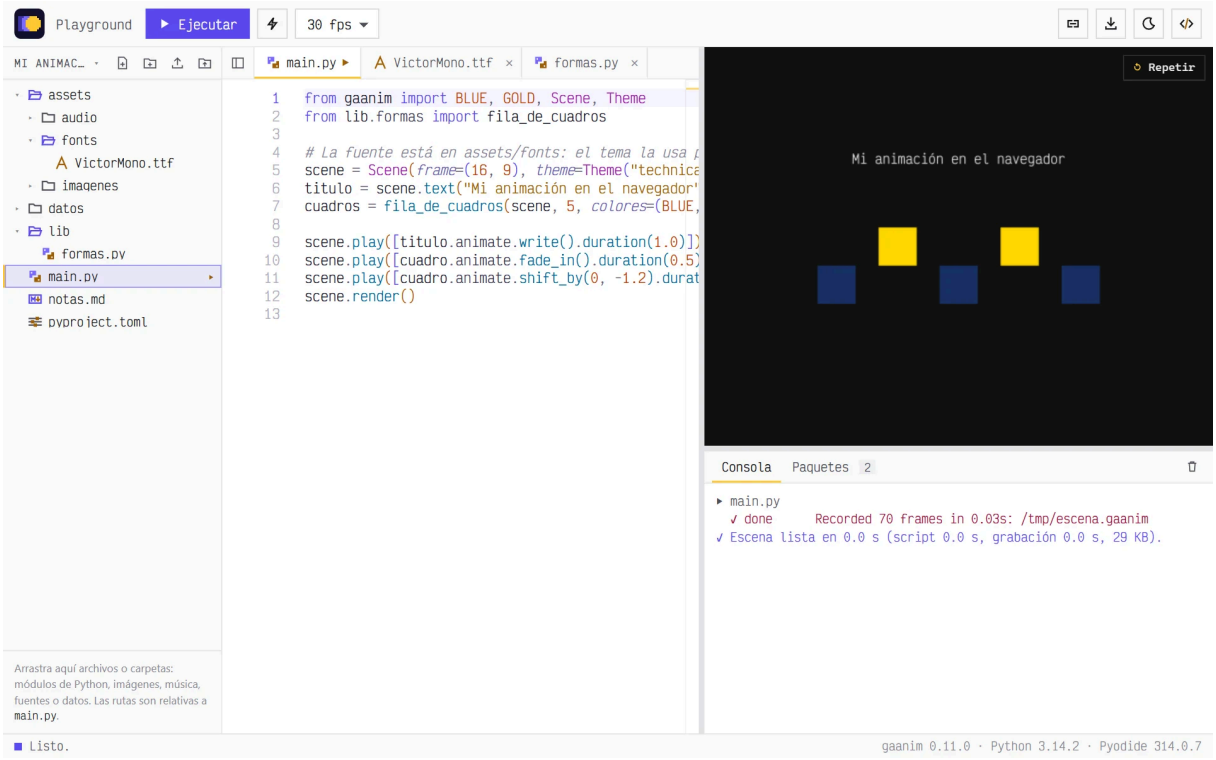


Figura 9: El tema claro. La escena conserva sus propios colores.

2.10. Límites

- Hace falta un navegador con WebGPU: Chrome o Edge recientes en el escritorio, y Safari o Firefox en las versiones que ya lo traen activado. Sin WebGPU, el reproductor no puede dibujar la escena.
- Python corre en un solo hilo del navegador: una escena larga o con muchos objetos tarda más en grabarse que en la aplicación.
- Lo que necesita FFmpeg (vídeos como recurso) y los paquetes con código nativo que no están en Pyodide solo funcionan en la aplicación, igual que `subprocess`, los hilos y las conexiones de red desde el script.
- Los proyectos viven en el navegador. Descarga el `.zip` de los que importen: es también la forma de llevarlos a otro equipo.

CAPÍTULO

3. Tu primera animación

Construye un punto que recorre un círculo, previsualízalo y expórtalo a video

3.1. Qué vas a construir

Un punto amarillo que da vueltas sobre un círculo, con un título que se escribe mientras gira. Son unas diez líneas de Python y aprenderás el ciclo completo: escribir la escena, verla y exportarla a video.

Necesitas Gaanim instalado (ver Instalación). Crea un proyecto y abre su `main.py` en tu editor:

```
gaanim init video mi-orbita
```

Borra el contenido de `main.py`; lo escribiremos por partes.

3.2. Paso 1: la escena y los objetos

```
from gaanim import BLUE, WHITE, YELLOW, Easing, Scene

scene = Scene(frame=(16, 9), background="#0f172a")

orbit = scene.geometry.circle(2).stroke(BLUE, 0.05).no_fill()
point = scene.geometry.dot(0.15).fill(YELLOW).move_to(2, 0)

scene.render()
```

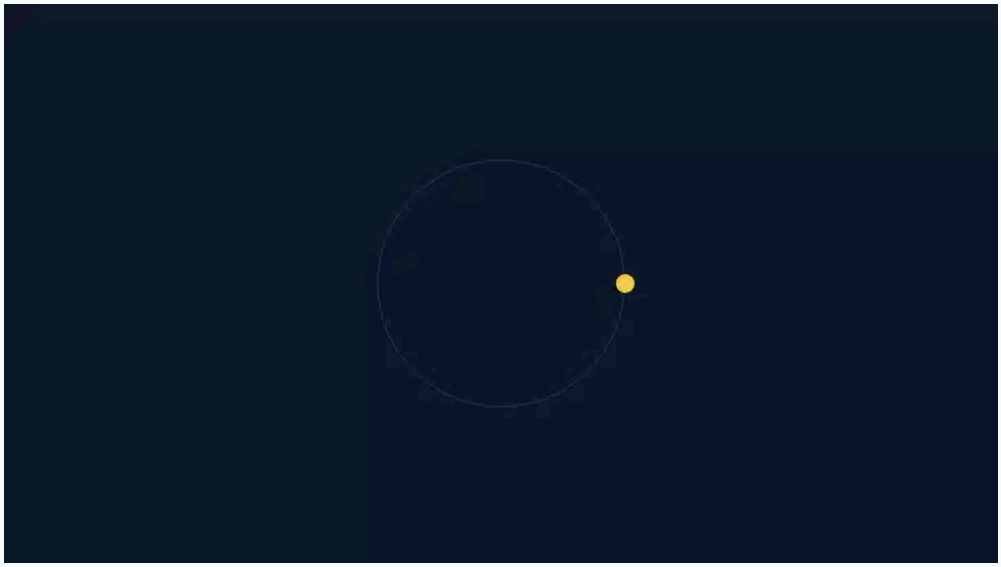
- `Scene(frame=(16, 9), ...)` crea un lienzo de 16 × 9 unidades con el origen en el centro: X va de `-8` a `8` e Y de `-4.5` a `4.5`, hacia arriba. Son unidades lógicas, no píxeles; la resolución se elige al exportar.
- `scene.geometry.circle(2)` crea un círculo de radio 2 centrado en el origen, y `scene.geometry.dot(0.15)` un punto pequeño. Cada fábrica devuelve un *handle*, una referencia al objeto con la que seguimos describiéndolo.
- `stroke`, `no_fill`, `fill` y `move_to` se encadenan: cada uno cambia el objeto y devuelve el handle. El punto empieza en `(2, 0)`, justo sobre el borde derecho del círculo.
- `scene.render()` va siempre al final. Entrega la escena a Gaanim, que decide si la muestra o la exporta.

Todavía no hay movimiento: la escena dura cero segundos.

3.3. Paso 2: haz que el punto gire

Añade una línea antes de `scene.render()`:

```
scene.play(point.animate.move_along(orbit).duration(2).easing(Easing.LINEAR))
scene.render()
```

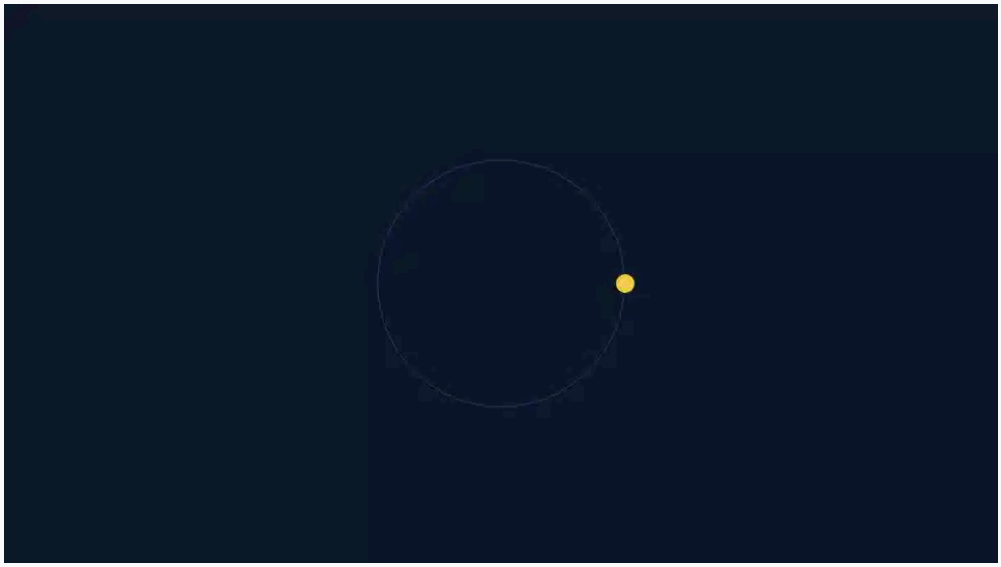


- `point.animate` abre la descripción de una animación del punto; `move_along(orbit)` le pide recorrer el contorno del círculo.
- `duration(2)` fija la vuelta en 2 segundos, y `easing(Easing.LINEAR)` mantiene la velocidad constante. Sin él, el punto arrancaría y frenaría suavemente.
- Nada se mueve hasta que `scene.play(...)` coloca esa animación en la línea de tiempo. La escena ahora dura 2 segundos.

El círculo y el punto no tienen animación de entrada, así que están visibles desde el primer fotograma.

3.4. Paso 3: un título y otra vuelta

```
title = scene.text("Movimiento circular", role="title").fill(WHITE).move_to(0, 3.25)

scene.play([
    title.animate.write().duration(1),
    point.animate.move_along(orbit).duration(2).easing(Easing.LINEAR),
])
scene.wait(0.5)
scene.render()
```



- `scene.text(..., role="title")` crea un texto con el estilo de título. Como su entrada es `write()`, permanece oculto hasta que empieza a escribirse.
- Las animaciones de una misma lista empiezan a la vez: el título se escribe mientras el punto da la segunda vuelta. Cada `play` empieza cuando termina el anterior y dura lo que su animación más larga, aquí 2 segundos.
- `scene.wait(0.5)` deja medio segundo quieto el último fotograma.

3.5. Previsualiza

Guarda el archivo y abre la vista previa desde la carpeta del proyecto:

```
cd mi-orbita
gaanim .
```

Se abre la ventana de Gaanim con la línea de tiempo. Deja la ventana abierta mientras editas: cada vez que guardas `main.py`, la escena se recarga y conserva el instante en que estabas. Prueba a cambiar `duration(2)` por `duration(1)` y guarda.

Para comprobar el script sin abrir ventana, por ejemplo antes de exportar, usa `gaanim check .`: ejecuta la escena e informa de errores y avisos.

3.6. Exporta

Para exportar no cambias el script; se lo pides al ejecutable:

```
gaanim export . --output exports/orbita.mp4
```

La extensión de `--output` elige el formato:

Extensión	Formato	Necesita FFmpeg
`.mp4`	video H.264, con audio si la escena lo tiene	sí
`.webm`	video VP9; admite fondo transparente	sí
`.mov`	Apple ProRes para editores de vídeo; admite fondo transparente	sí

Extensión	Formato	Necesita FFmpeg
`.webp`	WebP animado, útil para webs y documentación	sí
`.gif`	GIF animado	sí
`.png`	secuencia de imágenes, una por fotograma	no

Opciones útiles:

- `--quality draft` exporta rápido a 30 fps para revisar; `standard` (la predeterminada) usa 60 fps y `production`, la mejor codificación.
- `--fps` elige otra frecuencia, como 24 o 25.
- `--width` y `--height` cambian la resolución (1920 × 1080 por defecto). La escena no cambia: sus unidades lógicas se escalan a los píxeles que pidas.
- `--from` y `--to` exportan solo un tramo, en segundos.

`gaanim export --help` lista todas las opciones.

3.7. El archivo completo

```python
from gaanim import BLUE, WHITE, YELLOW, Easing, Scene

scene = Scene(frame=(16, 9), background="#0f172a")

orbit = scene.geometry.circle(2).stroke(BLUE, 0.05).no_fill()
point = scene.geometry.dot(0.15).fill(YELLOW).move_to(2, 0)
scene.play(point.animate.move_along(orbit).duration(2).easing(Easing.LINEAR))

title = scene.text("Movimiento circular", role="title").fill(WHITE).move_to(0, 3.25)
scene.play([
    title.animate.write().duration(1),
    point.animate.move_along(orbit).duration(2).easing(Easing.LINEAR),
])
scene.wait(0.5)

scene.render()
```

3.8. Siguientes pasos

- Cómo piensa Gaanim explica las ideas que acabas de usar: la escena, los handles, la línea de tiempo, los estilos y la salida.
- El tutorial parte de un círculo y un punto como estos y los convierte, lección a lección, en una explicación completa del círculo al seno.

CAPÍTULO

4. Cómo piensa Gaanim

El modelo mental de una escena: coordenadas, handles, línea de tiempo, estilos, layout, recursos y salida

4.1. Un script describe, no dibuja

Un script de Gaanim no pinta fotogramas. Describe una escena: qué objetos hay, cómo se ven y cuándo cambian. Al final, `scene.render()` entrega esa descripción al ejecutable `gaanim`, que la muestra en la vista previa, la comprueba o la exporta.

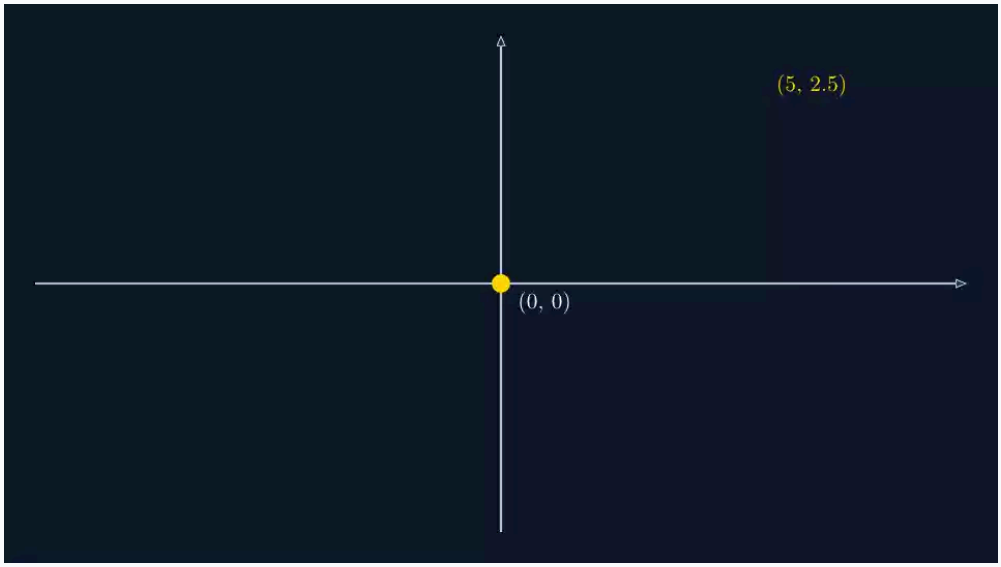
Como la escena es una descripción y no un video grabado, Gaanim puede saltar a cualquier instante, recargarla al guardar el archivo y exportar exactamente lo mismo que ves en la vista previa.

Esta página recorre las piezas de ese modelo. Cada ejemplo es un archivo completo que puedes ejecutar con `gaanim`. Si todavía no has creado ninguna escena, empieza por Tu primera animación.

4.2. La escena y sus coordenadas

`Scene` es el punto de partida: fija el tamaño del lienzo, el fondo, el tema y la línea de tiempo.

```
from gaanim import BLUE, GOLD, WHITE, Scene

scene = Scene(frame=(16, 9), background="#0f172a")

x_axis = scene.geometry.arrow(-7.5, 0, 7.5, 0).fill(BLUE)
y_axis = scene.geometry.arrow(0, -4, 0, 4).fill(BLUE)
origin = scene.text("(0, 0)", role="label").fill(WHITE).move_to(0.7, -0.4)
target = scene.text("(5, 2.5)", role="label").fill(GOLD).move_to(5, 3.1)

point = scene.geometry.dot(0.15).fill(GOLD)
scene.play(point.animate.move_to(5, 2.5).duration(1))
scene.render()
```



- El lienzo mide 16 × 9 **unidades lógicas**. El origen `(0, 0)` está en el centro, X crece hacia la derecha (de `-8` a `8`) e Y hacia arriba (de `-4.5` a `4.5`).
- Posiciones, radios, grosores de trazo y tamaños de texto usan esa misma unidad. Los píxeles solo aparecen al exportar, así que la misma escena sale igual a 1280 × 720 que a 3840 × 2160.
- Un objeto nuevo aparece en el origen hasta que lo mueves.
- Los tiempos se expresan en segundos y los ángulos, en radianes.

`frame` admite otras proporciones, por ejemplo `(9, 16)` para video vertical. `margin` define un área segura que usan el layout y las plantillas.

4.3. Fábricas y handles

Los objetos se crean con *fábricas* agrupadas por tema dentro de la escena:

- `scene.geometry`: círculos, rectángulos, líneas, flechas, arcos, grupos…
- `scene.text`: texto, ecuaciones con Typst y código.
- `scene.layout`: cajas como en CSS (filas, columnas, grids, capas) y zonas.
- `scene.media`: imágenes, SVG, video, audio y Lottie.
- `scene.viz`: ejes, gráficas de funciones y datos.

Cada fábrica registra el objeto en la escena y devuelve un *handle*: una referencia con la que sigues describiéndolo. Los métodos de estilo y posición (`fill`, `stroke`, `move_to`, `rotate_by`…) son *setters fluidos*: cambian el objeto y devuelven un handle, así que puedes encadenarlos o escribirlos en líneas separadas.

```
from gaanim import BLUE, GOLD, WHITE, Scene

scene = Scene(frame=(16, 9), background="#0f172a")

orbit = scene.geometry.circle(2).stroke(BLUE, 0.05).no_fill()
marker = scene.geometry.square(0.3).fill(GOLD).move_to(2, 0).rotate_by(0.785)

label = scene.text("r = 2", role="label")
label.fill(WHITE)
label.move_to(1, 0.3)
```

```
system = scene.geometry.group([orbit, marker, label]).move_to(-3, 0)
scene.render()
```

Un grupo transforma varias piezas como una unidad, y cada handle sigue sirviendo para cambiar su pieza por separado. Pon a los handles nombres que digan su papel (`orbit`, `label`), no su forma (`c1`, `t2`).

4.4. Construcción y línea de tiempo

Una escena tiene dos fases que conviene distinguir:

1. **Construcción**: todo lo que escribes antes del primer `play` o `wait` define el estado inicial.
2. **Línea de tiempo**: cada `play` y cada `wait` avanza un cursor. Lo que programas después ocurre en ese instante.

`objeto.animate` no cambia nada: devuelve la descripción de una animación (un `Anim`) que puedes configurar con `duration`, `easing` o `delay`. `scene.play(...)` la coloca en la línea de tiempo, en la posición del cursor.

```
from gaanim import BLUE, GOLD, RED, Scene

scene = Scene(frame=(16, 9), background="#0f172a")
circle = scene.geometry.circle(1).fill(BLUE).move_to(-2, 0)
square = scene.geometry.square(1.5).fill(GOLD).move_to(2, 0)

scene.play([
    circle.animate.fade_in().duration(1),
    square.animate.fade_in().duration(2),
])
print(scene.cursor)  # el play dura lo que su animación más larga

scene.play(circle.animate.shift_by(0, 1).duration(0.5))
scene.wait(0.5)
print(scene.cursor)

square.fill(RED)  # después del tiempo 3.0: es un corte instantáneo
scene.wait(1)
scene.render()

2.0
3.0
```

- Las animaciones de una misma lista empiezan juntas. Llamadas a `play` consecutivas van una detrás de otra.
- `scene.wait(segundos)` avanza el cursor sin animar nada.
- Un objeto con animación de entrada (`create`, `write`, `fade_in`…) permanece oculto hasta que esta empieza. Uno sin entrada está visible desde el principio.
- Un setter fluido llamado cuando el cursor ya avanzó, como `square.fill(RED)`, no reescribe el estado inicial: produce un cambio instantáneo en ese punto de la línea de tiempo. Para que el cambio sea gradual, usa `.animate`: `square.animate.fill(RED)`.

4.4.1. Grupos de animaciones y easing

Para coordinar varias animaciones, `scene.play` acepta composiciones: `stagger(...)` escalona los inicios, `sequence(...)` encadena y `parallel(...)` agrupa. El *easing* decide cómo avanza cada animación en su duración.

```python
from gaanim import GOLD, WHITE, Easing, Scene, stagger

scene = Scene(frame=(16, 9), background="#0f172a")
easings = {"LINEAR": Easing.LINEAR, "SMOOTH": Easing.SMOOTH, "BOUNCY": Easing.BOUNCY}

dots = []
for i, name in enumerate(easings):
    y = 1.5 - 1.5 * i
    scene.text(name, role="label").fill(WHITE).move_to(-6, y)
    dots.append(scene.geometry.dot(0.25).fill(GOLD).move_to(-4, y))

scene.play(stagger(*[dot.animate.fade_in().duration(0.4) for dot in dots], each=0.1))
scene.play([
    dot.animate.shift_by(8, 0).duration(1.5).easing(easing)
    for dot, easing in zip(dots, easings.values())
])
scene.render()
```

- `stagger(a, b, c, each=0.1)` arranca cada animación 0.1 s después de la anterior.
- `Easing.LINEAR` mantiene la velocidad; `Easing.SMOOTH` acelera y frena, y es el valor por defecto de la mayoría de las animaciones; `Easing.BOUNCY` es un muelle que se pasa del destino y rebota.
- Elige el easing por lo que comunica: velocidad constante para un proceso físico uniforme, suavizado para un cambio de estado, muelle para algo que llega con energía.

La referencia de animaciones lista todas las animaciones, los easings y las composiciones.

4.5. Estilos y temas

Hay dos maneras de dar estilo:

- **Estilo local**: los setters fluidos (`fill`, `stroke`, `opacity`) sobre un objeto concreto. Es lo más directo para una excepción.
- **Tema**: un `Theme` reúne colores con nombre, tipografía y reglas por tipo de objeto o por clase. Cambiar el tema cambia toda la escena.

```python
from gaanim import Scene, Style, Theme

theme = Theme(
    "technical",
```

```
    colors={"brand": "#38bdf8"},
    styles={".destacado": Style(fill="brand")},
)
scene = Scene(theme=theme)

normal = scene.geometry.circle(1).move_to(-3, 0)
featured = scene.geometry.circle(1).style_class("destacado")
explicit = scene.geometry.circle(1).style_class("destacado").fill("#f97316").move_to(3, 0)
scene.render()
```

El primero toma el estilo del tema `technical`; el segundo, el de la clase `.destacado`; el tercero, el color explícito, porque un setter fluido siempre gana. Los roles de texto (`title`, `subtitle`, `body`, `label`…) también vienen del tema, así que `scene.text(..., role="title")` tiene el aspecto de título sin fijar tamaño ni fuente.

Toda escena empieza con el tema `technical` (fondo gris casi negro, texto claro y formas con el color de acento), así que un `Scene()` sin argumentos ya se ve bien. `Scene(theme="paper")` usa otro tema incluido tal cual; `Theme.schemes()` lista los disponibles y `Scene(theme=None)` crea un lienzo sin tema, blanco, en el que das color a cada objeto. Los detalles están en Temas y colores.

4.6. Coordenadas o layout

Para colocar objetos tienes dos herramientas, cada una para un tipo de pieza:

- **Coordenadas** (`move_to`, `shift_by`) cuando la posición tiene significado: el punto de una gráfica, el centro de una órbita.
- **Layout** cuando lo que importa es la composición: un título arriba, una fila de pasos, columnas, tarjetas. El layout mide el contenido y se reajusta si cambia un texto.

```
from gaanim import Direction, Scene

scene = Scene(frame=(16, 9), theme="technical")

title = scene.text("Tres pasos", role="title").to_edge(Direction.UP, buff=0.6)
steps = scene.layout.row(
    *[scene.text(word, role="body") for word in ("Crear", "Animar", "Exportar")],
    gap=1.2,
).move_to(0, 1)

orbit = scene.geometry.circle(1).move_to(0, -2)
radius = scene.text("r = 1", role="caption").next_to(orbit, Direction.RIGHT, spacing=0.3)
scene.render()
```

`to_edge` y `next_to` colocan un objeto respecto al borde del lienzo o a otro objeto. `scene.layout.row` es un contenedor: decide dónde va cada hijo, y colocar un hijo por su cuenta produce un `LayoutOwnershipError`. Mueve el contenedor entero, como hace `.move_to(0, 1)`. Animar a un hijo con `.animate` sí está permitido: parte del sitio que le da el contenedor.

La guía de Layout cubre cajas con padding, gap, bordes y sombras, grids, estilos reutilizables, zonas y cómo animar la estructura.

4.7. Recursos

Imágenes, SVG, audio, video y fuentes viven en la carpeta `assets/` del proyecto. `scene.assets.load_project()` lee el `gaanim.toml` que está junto al script y resuelve las rutas relativas dentro de esa carpeta:

```
from gaanim import WHITE, Scene

scene = Scene(frame=(16, 9), background="#0f172a")
scene.assets.load_project()

cover = scene.media.image("cover.png", width=3).move_to(0, 0.5)
caption = scene.text("assets/cover.png", role="caption").fill(WHITE).move_to(0, -1.5)
scene.render()
```

Así el proyecto funciona igual lo abras desde donde lo abras. `width` usa unidades de la escena, como todo lo demás. Consulta Recursos para la precarga y los formatos, y Audio para sonido y narración.

4.8. La salida

El script no decide qué se hace con la escena; lo decide el comando con el que lo abres:

Comando	Qué hace
`gaanim .`	Abre la vista previa y recarga al guardar.
`gaanim check .`	Ejecuta la escena sin ventana e informa de errores y avisos.
`gaanim export . --output video.mp4`	Renderiza la línea de tiempo a un archivo.
`gaanim --present .`	Presenta la escena y se detiene en cada `scene.stop()`.

Por eso `scene.render()` va siempre al final del script: es el momento en que la descripción está completa. La exportación y sus formatos se explican en Tu primera animación la organización de proyectos, en Proyectos, y las presentaciones, en Presentaciones.

4.9. Dónde seguir

- El tutorial aplica estas ideas a un proyecto completo, lección a lección.
- La referencia documenta cada fábrica, setter y animación: Scene, geometría y texto.

PARTE II

Tutorial: del círculo al seno

Un proyecto completo, capítulo a capítulo, que siempre se puede ejecutar

CAPÍTULO

5. Antes de empezar

Qué vamos a construir, cómo está organizado el tutorial y cómo preparar el proyecto

5.1. Qué vamos a construir

En este tutorial construirás, paso a paso, una animación de diez segundos que explica una idea matemática: cómo un punto que gira sobre un círculo dibuja una onda seno.

Al terminar, la escena mostrará un título, un círculo con su radio y un punto que da una vuelta completa. A la derecha, una recta numerada de 0 a 2π y una onda que se dibuja al mismo ritmo, unida al punto por una línea de proyección. Todo el movimiento depende de un único valor: el ángulo `theta`.

Cada capítulo añade una idea de Gaanim a la misma escena:

1. **Primera escena**: el lienzo, las coordenadas y los primeros objetos.
2. **Objetos y estilo**: una paleta con propósito, trazos, texto y orden de dibujo.
3. **Animar el tiempo**: `scene.play`, duraciones, `stagger` y easing.
4. **Componer y explicar**: una fórmula y un panel organizado con Layout.
5. **Dar vida a la escena**: un parámetro animable que mueve el punto y el radio.
6. **Del círculo a la onda**: una recta numerada, la curva seno y la proyección.
7. **Terminar el proyecto**: un cierre, la revisión y la exportación.

5.2. Cómo leer cada capítulo

Todos los capítulos siguen el mismo orden:

1. **Objetivo**: qué verás al terminar el capítulo.
2. **Cambios**: fragmentos de código con la explicación de cada uno. Un comentario como `# Círculo` indica en qué zona de `main.py` va cada fragmento.
3. **Archivo completo**: el `main.py` entero en ese punto, con una vista previa de lo que debe mostrar. Si algo no coincide, compara tu archivo con este.

Cada archivo completo se ejecuta al construir esta documentación, así que siempre funciona tal como aparece.

IDEA CLAVE

No copies todos los fragmentos de golpe. Aplica un cambio, guarda y mira el resultado en el editor. Ver cómo reacciona la escena a cada línea enseña más que leer el código terminado.

5.3. Lo que necesitas saber

El tutorial supone Python básico: imports, variables, tuplas, funciones y listas. No necesitas saber Rust, Bevy ni programación de GPU.

Gaanim usa una API fluida: cada llamada devuelve el mismo objeto, así que puedes encadenarlas. Esta expresión crea un círculo, le da un trazo y lo coloca en la escena, de izquierda a derecha:

```
circle = scene.geometry.circle(1.5).stroke(WHITE, 0.05).move_to(-4, 0)
```

Todas las medidas usan unidades lógicas de un fotograma de 16 × 9, nunca píxeles.

5.4. Preparar el proyecto

Si todavía no tienes Gaanim, sigue primero Instalación. Después crea el proyecto que usaremos durante todo el tutorial y ábrelo:

```
gaanim init video del-circulo-al-seno
cd del-circulo-al-seno
gaanim .
```

`gaanim .` abre el editor con la escena de ejemplo del proyecto. Mientras editas `main.py`, el editor vuelve a ejecutar la escena cada vez que guardas.

Estas son las piezas del proyecto que importan en el tutorial:

```
del-circulo-al-seno/
  gaanim.toml   # nombre del proyecto y archivo de entrada
  main.py       # la escena: el único archivo que editaremos
  assets/       # imágenes, SVG y fuentes
  exports/      # videos exportados
```

El scaffold crea además un `README.md`, un `pyproject.toml` para el entorno de Python y un `AGENTS.md` para asistentes de código.

PUNTO DE CONTROL

El editor muestra la escena de ejemplo del proyecto sin errores. Si no se abre, ejecuta `gaanim check .` desde la carpeta del proyecto para ver qué falla y revisa Instalación.

5.5. El archivo final

El resultado de este tutorial también se instala con Gaanim: está en la carpeta del paquete `gaanim`, en `_docs/tutorial/circulo_al_seno.py`. Úsalo para comparar si te pierdes, pero intenta llegar a él por tu cuenta.

En el siguiente capítulo, Primera escena, sustituirás el ejemplo por el primer fotograma de la explicación.

CAPÍTULO

6. Primera escena

El lienzo, las coordenadas y el primer fotograma del proyecto

6.1. Objetivo

Al terminar este capítulo, `main.py` mostrará un fotograma quieto: el título, un círculo a la izquierda, un punto amarillo en su borde derecho y el radio que los une. Todavía no habrá movimiento.

Abre `main.py` y borra su contenido: vamos a escribir la escena desde cero.

6.2. Cambios

6.2.1. El lienzo

```
from gaanim import CYAN, WHITE, YELLOW, Scene

scene = Scene(frame=(16, 9), background="#0f172a", margin=0.6)

CENTER = (-4.5, -1.0)
R = 1.5
```

`Scene(frame=(16, 9), ...)` crea un lienzo horizontal de 16 unidades de ancho y 9 de alto. El origen `(0, 0)` está en el centro: X crece hacia la derecha e Y hacia arriba, así que X va de `-8` a `8` e Y de `-4.5` a `4.5`. `margin=0.6` reserva un borde de seguridad de 0.6 unidades para que nada quede pegado al límite del fotograma.

Estas unidades son lógicas y no dependen de la resolución: la misma escena se exporta a 1280 × 720 o a 3840 × 2160 sin cambiar el código. Radios, grosores de trazo y tamaños de texto usan la misma unidad.

`CENTER` y `R` guardan el centro y el radio del círculo. Toda la geometría del tutorial se calculará a partir de ellos: si cambias uno, todo lo demás lo acompaña.

6.2.2. El título

```
title = scene.text("Del círculo al seno", role="title")
title.fill(WHITE).move_to(0, 3.4)
```

`scene.text` crea texto vectorial. El rol `"title"` elige el tamaño de un título; `fill` le da color y `move_to` coloca su centro cerca del borde superior.

La variable `title` no contiene una imagen ni unas coordenadas: es un *handle*, una referencia al objeto que ya vive en la escena. Con él seguiremos modificándolo y, más adelante, animándolo.

6.2.3. El círculo y el punto

```
orbit = scene.geometry.circle(R)
orbit.stroke(CYAN, 0.05).no_fill().move_to(*CENTER)
```

```
point = scene.geometry.dot(0.125)
point.fill(YELLOW).move_to(CENTER[0] + R, CENTER[1])
```

`scene.geometry` agrupa las fábricas de formas. El círculo solo tiene trazo (`stroke` con color y grosor, y `no_fill`) porque representa una trayectoria. `move_to(*CENTER)` desempaqueta la tupla en sus dos coordenadas.

El punto se coloca a `R` unidades a la derecha del centro, es decir, justo sobre el borde del círculo.

> **IDEA CLAVE**
>
> Nombra los objetos por su papel, no por su aspecto. `orbit` sigue siendo un buen nombre aunque en el próximo capítulo cambie de color; `cyan_circle` dejaría de serlo.

6.2.4. El radio

```
radius = scene.geometry.line(CENTER, point).stroke(WHITE, 0.025)
```

`scene.geometry.line` acepta coordenadas u objetos como extremos. Aquí el origen es el punto fijo `CENTER`, pero el final es el handle `point`: la línea termina siempre donde esté el punto. Hoy eso no se nota, porque el punto está quieto; en el capítulo 6, cuando el punto gire, el radio lo seguirá sin tocar esta línea.

6.2.5. Mantener el fotograma

Termina el archivo con:

```
scene.wait(1)
scene.render()
```

`scene.render()` entrega la escena al editor y debe ser siempre la última línea. `scene.wait(1)` mantiene el fotograma durante un segundo: una escena sin duración no tiene nada que exportar. En el capítulo 4 sustituiremos esa espera por animaciones.

6.3. Archivo completo

```
from gaanim import CYAN, WHITE, YELLOW, Scene

scene = Scene(frame=(16, 9), background="#0f172a", margin=0.6)

CENTER = (-4.5, -1.0)
R = 1.5

title = scene.text("Del círculo al seno", role="title")
title.fill(WHITE).move_to(0, 3.4)

orbit = scene.geometry.circle(R)
orbit.stroke(CYAN, 0.05).no_fill().move_to(*CENTER)

point = scene.geometry.dot(0.125)
point.fill(YELLOW).move_to(CENTER[0] + R, CENTER[1])

radius = scene.geometry.line(CENTER, point).stroke(WHITE, 0.025)

scene.wait(1)
scene.render()
```

PUNTO DE CONTROL

Al guardar, el editor muestra el título arriba, un círculo cian a la izquierda y un radio blanco horizontal que termina en el punto amarillo, sobre el borde derecho del círculo. Si el punto no toca el círculo, revisa que su posición use `CENTER[0] + R`.

En el siguiente capítulo, Objetos y estilo, convertiremos este boceto en una composición con jerarquía visual.

CAPÍTULO

7. Objetos y estilo

Una paleta con propósito, rellenos, trazos, texto secundario y orden de dibujo

7.1. Objetivo

El primer fotograma funciona, pero todos sus elementos compiten por la atención. Al terminar este capítulo tendrá una jerarquía clara: la órbita define el espacio, el punto amarillo es el foco, el radio queda en segundo plano y un subtítulo gris explica la escena. Seguirá sin haber movimiento.

7.2. Cambios

7.2.1. Una paleta con propósito

Sustituye el import y la línea de `Scene` por:

```
from gaanim import WHITE, YELLOW, Color, Scene

# Paleta
BACKGROUND = Color(15, 23, 42)
PRIMARY = Color(96, 165, 250)
ACCENT = YELLOW
MUTED = Color(148, 163, 184)

scene = Scene(frame=(16, 9), background=BACKGROUND, margin=0.6)

CENTER = (-4.5, -1.0)
R = 1.5
```

`Color(r, g, b)` crea un color a partir de sus componentes de 0 a 255; el fondo es el mismo azul noche de antes. Los nombres dicen para qué sirve cada color, no cómo es: `PRIMARY` para la geometría principal, `ACCENT` para lo que debe mirar el espectador y `MUTED` para lo secundario. Si mañana cambias la identidad visual, solo tocas la paleta.

`CYAN` ya no se usa, así que desaparece del import.

7.2.2. Texto principal y secundario

En la zona `# Texto`, añade un subtítulo debajo del título:

```
# Texto
title = scene.text("Del círculo al seno", role="title")
title.fill(WHITE).move_to(0, 3.4)
caption = scene.text("Un punto que gira a radio constante", role="subtitle")
caption.fill(MUTED).move_to(0, 2.8)
```

Los roles conectan el texto con la tipografía del tema: `"subtitle"` es más pequeño que `"title"`. El color `MUTED` termina de marcarlo como información secundaria.

7.2.3. Relleno, trazo y color

Reescribe la geometría con la paleta. Ahora cada objeto cabe en una sola cadena de llamadas:

```
# Círculo
orbit = scene.geometry.circle(R).stroke(PRIMARY, 0.05).no_fill().move_to(*CENTER)
point = scene.geometry.dot(0.125).fill(ACCENT).move_to(CENTER[0] + R, CENTER[1])
radius = scene.geometry.line(CENTER, point).stroke(MUTED, 0.025)
```

Una forma vectorial puede tener relleno (`fill`), trazo (`stroke`) o ambos. La órbita solo tiene trazo porque es una trayectoria; el punto solo tiene relleno porque es una masa. El radio usa la mitad de grosor que la órbita y un color apagado: está para explicar, no para protagonizar.

7.2.4. El orden de dibujo

Si te fijas en el capítulo anterior, el extremo del radio se dibujaba encima del punto. Por defecto, lo que se crea después queda encima, y `radius` tiene que crearse después de `point` porque lo usa como extremo. Súbelo de capa:

```
point.z_index(1)
```

`z_index` fija la capa de dibujo: los valores más altos se dibujan encima. Con `1`, el punto queda por delante del radio y de la órbita, que siguen en la capa `0`.

IDEA CLAVE

El estilo también es contenido. Un color de acento y un trazo más fino ya le dicen al espectador dónde mirar antes de que ocurra nada.

7.3. Archivo completo

```
from gaanim import WHITE, YELLOW, Color, Scene

# Paleta
BACKGROUND = Color(15, 23, 42)
PRIMARY = Color(96, 165, 250)
ACCENT = YELLOW
MUTED = Color(148, 163, 184)

scene = Scene(frame=(16, 9), background=BACKGROUND, margin=0.6)

CENTER = (-4.5, -1.0)
R = 1.5

# Texto
title = scene.text("Del círculo al seno", role="title")
title.fill(WHITE).move_to(0, 3.4)
caption = scene.text("Un punto que gira a radio constante", role="subtitle")
caption.fill(MUTED).move_to(0, 2.8)

# Círculo
orbit = scene.geometry.circle(R).stroke(PRIMARY, 0.05).no_fill().move_to(*CENTER)
point = scene.geometry.dot(0.125).fill(ACCENT).move_to(CENTER[0] + R, CENTER[1])
radius = scene.geometry.line(CENTER, point).stroke(MUTED, 0.025)
point.z_index(1)

scene.wait(1)
scene.render()
```

PUNTO DE CONTROL

La órbita es azul claro, el radio gris y más fino, y el punto amarillo se ve entero, por encima del extremo del radio. Bajo el título aparece el subtítulo «Un punto que gira a radio constante» en gris.

En el siguiente capítulo, Animar el tiempo, haremos que cada elemento entre en escena en su momento.

CAPÍTULO

8. Animar el tiempo

La línea de tiempo: scene.play, duraciones, stagger, easing y pausas

8.1. Objetivo

Hasta ahora todo aparecía de golpe. Al terminar este capítulo la escena contará su contenido en orden: primero se escribe el título, después se dibuja la órbita, crece el radio y el punto entra con un pequeño rebote. La animación durará tres segundos.

8.2. Cambios

8.2.1. Nuevos imports

Amplía el import:

```
from gaanim import WHITE, YELLOW, Color, Easing, Scene, stagger
```

`stagger` escalona animaciones y `Easing` elige cómo avanzan en el tiempo.

8.2.2. La entrada del texto

Añade una zona nueva, `# Línea de tiempo`, justo antes de `scene.wait(1)`:

```
# Línea de tiempo
scene.play(stagger(
    title.animate.write().duration(0.8),
    caption.animate.fade_in().duration(0.5),
    each=0.5,
))
```

Cada objeto tiene una propiedad `.animate` con el mismo vocabulario que sus métodos, más las animaciones de entrada y salida. `title.animate.write()` no cambia nada todavía: devuelve una animación, un objeto `Anim` que describe qué ocurrirá. `.duration(0.8)` fija cuántos segundos dura.

`scene.play(...)` coloca esas animaciones en la línea de tiempo, a partir del instante actual. `stagger(..., each=0.5)` las lanza una tras otra con medio segundo entre inicios: el título empieza en 0 s y el subtítulo en 0.5 s. El bloque termina cuando acaba su última animación, en el segundo 1.

8.2.3. La entrada del círculo

Justo debajo, añade la segunda entrada:

```
scene.play(stagger(
    orbit.animate.create().duration(0.8),
    radius.animate.create().duration(0.4),
    point.animate.grow_from_center().duration(0.5).easing(Easing.SNAPPY),
    each=0.25,
))
```

Cada llamada a `scene.play` empieza cuando termina la anterior, así que esta entrada ocupa del segundo 1 al 2. `create()` dibuja el trazo desde su inicio: la órbita se cierra en 0.8 s y el radio crece desde el centro hacia el punto.

`grow_from_center()` hace crecer el punto desde tamaño cero. Por defecto, las animaciones aceleran al empezar y frenan al terminar (`Easing.SMOOTH`). `Easing.SNAPPY` es un muelle: el punto se pasa un poco de su tamaño y vuelve, lo justo para atraer la mirada.

8.2.4. Una pausa final

La línea `scene.wait(1)` que ya tenías cambia de sentido: ahora deja un segundo para mirar el resultado cuando termina la segunda entrada.

IDEA CLAVE

Un objeto con animación de entrada permanece oculto hasta que esta empieza. Por eso los objetos se crean todos arriba, en su estado final, y la línea de tiempo solo decide cuándo aparecen. Gaanim conoce así la escena completa y puede mostrar cualquier instante exacto al mover la barra de reproducción.

8.3. Archivo completo

```python
from gaanim import WHITE, YELLOW, Color, Easing, Scene, stagger

# Paleta
BACKGROUND = Color(15, 23, 42)
PRIMARY = Color(96, 165, 250)
ACCENT = YELLOW
MUTED = Color(148, 163, 184)

scene = Scene(frame=(16, 9), background=BACKGROUND, margin=0.6)

CENTER = (-4.5, -1.0)
R = 1.5

# Texto
title = scene.text("Del círculo al seno", role="title")
title.fill(WHITE).move_to(0, 3.4)
caption = scene.text("Un punto que gira a radio constante", role="subtitle")
caption.fill(MUTED).move_to(0, 2.8)

# Círculo
orbit = scene.geometry.circle(R).stroke(PRIMARY, 0.05).no_fill().move_to(*CENTER)
point = scene.geometry.dot(0.125).fill(ACCENT).move_to(CENTER[0] + R, CENTER[1])
radius = scene.geometry.line(CENTER, point).stroke(MUTED, 0.025)
point.z_index(1)

# Línea de tiempo
scene.play(stagger(
    title.animate.write().duration(0.8),
    caption.animate.fade_in().duration(0.5),
    each=0.5,
))
scene.play(stagger(
    orbit.animate.create().duration(0.8),
    radius.animate.create().duration(0.4),
    point.animate.grow_from_center().duration(0.5).easing(Easing.SNAPPY),
    each=0.25,
))
scene.wait(1)
scene.render()
```

PUNTO DE CONTROL

La vista previa dura 3 segundos. En el primero se escribe el título y aparece el subtítulo; en el segundo se dibuja la órbita, crece el radio y el punto entra con un rebote; en el tercero la escena queda quieta. Arrastra la barra de reproducción del editor hasta 0.5 s: la órbita todavía no debe verse.

En el siguiente capítulo, Componer y explicar, añadiremos una fórmula y aprenderemos cuándo usar Layout en lugar de coordenadas.

CAPÍTULO

9. Componer y explicar

Texto matemático, un panel con Layout y un reparto claro del espacio

9.1. Objetivo

La izquierda ya contiene el círculo. Al terminar este capítulo, arriba a la derecha aparecerá un panel con la fórmula $y = r\,\sin(\theta)$ y una frase que anuncia lo que ocurrirá: la altura del punto dibujará una onda. El panel se escribirá justo después de la entrada del círculo.

El espacio queda repartido con intención: a la izquierda ocurre el fenómeno, arriba a la derecha lo nombramos y debajo del panel queda sitio libre para la onda del capítulo 7.

9.2. Cambios

9.2.1. Texto matemático

Añade al final de la zona `# Texto`:

```
formula = scene.text("$y = r sin(theta)$", role="subtitle").fill(WHITE)
explanation = scene.text("La altura del punto dibuja la onda.", role="body")
explanation.fill(MUTED)
```

`scene.text` compone con Typst lo que va entre `$...$`: `sin` se escribe en redonda, `theta` se convierte en θ y las variables van en cursiva. El resto de la cadena es texto normal.

Estos dos textos no llevan `move_to`: su posición la decidirá el panel.

9.2.2. Un panel con Layout

Justo debajo, agrupa los dos textos en una columna:

```
panel = scene.layout.column([formula, explanation], gap=0.2, width=6.5)
panel.move_to(3.5, 1.6)
```

`scene.layout.column` apila sus hijos de arriba abajo, separados por `gap` unidades y alineados a la izquierda. `width=6.5` fija el ancho del panel: si la frase no cupiera, se partiría en varias líneas dentro de ese ancho. Después movemos el panel entero; sus hijos lo acompañan.

A partir de aquí, el panel decide dónde descansan `formula` y `explanation`. No los coloques con un `move_to` inmediato: Gaanim lo rechaza con un `LayoutOwnershipError`. Para recolocarlos, mueve `panel`; para animarlos, usa `.animate` sobre ellos, que parte del sitio que les da el panel.

IDEA CLAVE

Usa coordenadas para la geometría y Layout para el contenido. El círculo depende de un centro y un radio exactos, así que usa coordenadas. El panel depende de lo que miden sus textos, así que usa Layout: si cambias la frase, la columna se recalcula sola.

9.2.3. La entrada del panel

En la zona `# Línea de tiempo`, añade una tercera entrada antes de `scene.wait(1)`:

```
scene.play(stagger(
    formula.animate.write().duration(0.8),
    explanation.animate.write().duration(0.8),
    each=0.2,
))
```

El panel ocupa del segundo 2 al 3, cuando el espectador ya conoce el círculo: la explicación llega después del fenómeno y no compite con él.

9.3. Archivo completo

```
from gaanim import WHITE, YELLOW, Color, Easing, Scene, stagger

# Paleta
BACKGROUND = Color(15, 23, 42)
PRIMARY = Color(96, 165, 250)
ACCENT = YELLOW
MUTED = Color(148, 163, 184)

scene = Scene(frame=(16, 9), background=BACKGROUND, margin=0.6)

CENTER = (-4.5, -1.0)
R = 1.5

# Texto
title = scene.text("Del círculo al seno", role="title")
title.fill(WHITE).move_to(0, 3.4)
caption = scene.text("Un punto que gira a radio constante", role="subtitle")
caption.fill(MUTED).move_to(0, 2.8)

formula = scene.text("$y = r sin(theta)$", role="subtitle").fill(WHITE)
explanation = scene.text("La altura del punto dibuja la onda.", role="body")
explanation.fill(MUTED)
panel = scene.layout.column([formula, explanation], gap=0.2, width=6.5)
panel.move_to(3.5, 1.6)

# Círculo
orbit = scene.geometry.circle(R).stroke(PRIMARY, 0.05).no_fill().move_to(*CENTER)
point = scene.geometry.dot(0.125).fill(ACCENT).move_to(CENTER[0] + R, CENTER[1])
radius = scene.geometry.line(CENTER, point).stroke(MUTED, 0.025)
point.z_index(1)

# Línea de tiempo
scene.play(stagger(
    title.animate.write().duration(0.8),
    caption.animate.fade_in().duration(0.5),
    each=0.5,
))
scene.play(stagger(
    orbit.animate.create().duration(0.8),
    radius.animate.create().duration(0.4),
```

```
    point.animate.grow_from_center().duration(0.5).easing(Easing.SNAPPY),
    each=0.25,
))
scene.play(stagger(
    formula.animate.write().duration(0.8),
    explanation.animate.write().duration(0.8),
    each=0.2,
))
scene.wait(1)
scene.render()
```

PUNTO DE CONTROL

La vista previa dura 4 segundos. Entre el segundo 2 y el 3 se escriben la fórmula y la frase, alineadas a la izquierda y en una sola línea cada una, arriba a la derecha y por debajo del subtítulo. Bajo el panel queda un espacio vacío: lo ocupará la onda.

En el siguiente capítulo, Dar vida a la escena, el punto empezará a girar y el radio lo seguirá.

CAPÍTULO

10. Dar vida a la escena

Un parámetro animable que mueve el punto y arrastra al radio

10.1. Objetivo

Al terminar este capítulo, después de que se escriba el panel, el punto dará una vuelta completa al círculo en sentido antihorario, a velocidad constante, y el radio lo seguirá en todo momento.

No vamos a animar la posición del punto de un sitio a otro. Vamos a animar un solo número, el ángulo `theta`, y a declarar que la posición del punto depende de él. En el capítulo siguiente, la onda dependerá del mismo número.

10.2. Cambios

10.2.1. Un import de Python

Añade `import math` al principio del archivo: necesitaremos `math.pi`.

```
import math

from gaanim import WHITE, YELLOW, Color, Easing, Scene, stagger
```

10.2.2. Un ángulo animable

Sustituye la zona `# Círculo` completa por esta versión:

```
# Círculo
theta = scene.viz.parameter(0.0)
orbit = scene.geometry.circle(R).stroke(PRIMARY, 0.05).no_fill().move_to(*CENTER)
circle_ref = scene.geometry.polar_point(CENTER, R, theta)
point = scene.geometry.dot(0.125).fill(ACCENT).follow(circle_ref)
radius = scene.geometry.line(CENTER, point).stroke(MUTED, 0.025)
point.z_index(1)
```

Solo cambian tres cosas:

- `scene.viz.parameter(0.0)` crea `theta`, un `Parameter`: un número de la escena que empieza en `0` y que se puede animar como cualquier objeto.
- `scene.geometry.polar_point(CENTER, R, theta)` describe un punto a distancia `R` de `CENTER` con ángulo `theta`, en radianes, medido desde la derecha y en sentido antihorario. No dibuja nada: es una referencia que Gaanim recalcula cuando cambia `theta`.
- `point` ya no usa `move_to`, sino `follow(circle_ref)`: su centro queda pegado a esa referencia. Con `theta = 0`, el punto está donde estaba antes, en el borde derecho.

`radius` no cambia. En el capítulo 2 le dimos `point` como extremo, así que ya sigue al punto allá donde vaya.

10.2.3. Animar el ángulo

En la zona `# Línea de tiempo`, añade la vuelta completa antes de `scene.wait(1)`:

```
scene.play([theta.animate.set(2 * math.pi).duration(4).easing(Easing.LINEAR)])
```

`theta.animate.set(2 * math.pi)` lleva el ángulo de `0` a 2π, una vuelta completa, en 4 segundos. `scene.play` también acepta una lista de animaciones, que empiezan a la vez; aquí solo hay una.

`Easing.LINEAR` hace que el ángulo avance siempre al mismo ritmo. Con el easing por defecto, el punto arrancaría despacio y frenaría al final, y el movimiento circular dejaría de ser uniforme.

IDEA CLAVE

Anima el valor, no la forma. `theta` es la única fuente de verdad: el punto depende de él y el radio depende del punto. Como todo se calcula a partir del tiempo, puedes saltar a cualquier instante y la escena será exactamente la misma que al reproducirla de corrido.

10.3. Archivo completo

```
import math

from gaanim import WHITE, YELLOW, Color, Easing, Scene, stagger

# Paleta
BACKGROUND = Color(15, 23, 42)
PRIMARY = Color(96, 165, 250)
ACCENT = YELLOW
MUTED = Color(148, 163, 184)

scene = Scene(frame=(16, 9), background=BACKGROUND, margin=0.6)

CENTER = (-4.5, -1.0)
R = 1.5

# Texto
title = scene.text("Del círculo al seno", role="title")
title.fill(WHITE).move_to(0, 3.4)
caption = scene.text("Un punto que gira a radio constante", role="subtitle")
caption.fill(MUTED).move_to(0, 2.8)

formula = scene.text("$y = r sin(theta)$", role="subtitle").fill(WHITE)
explanation = scene.text("La altura del punto dibuja la onda.", role="body")
explanation.fill(MUTED)
panel = scene.layout.column([formula, explanation], gap=0.2, width=6.5)
panel.move_to(3.5, 1.6)

# Círculo
theta = scene.viz.parameter(0.0)
orbit = scene.geometry.circle(R).stroke(PRIMARY, 0.05).no_fill().move_to(*CENTER)
circle_ref = scene.geometry.polar_point(CENTER, R, theta)
point = scene.geometry.dot(0.125).fill(ACCENT).follow(circle_ref)
radius = scene.geometry.line(CENTER, point).stroke(MUTED, 0.025)
point.z_index(1)

# Línea de tiempo
scene.play(stagger(
    title.animate.write().duration(0.8),
    caption.animate.fade_in().duration(0.5),
    each=0.5,
))
scene.play(stagger(
    orbit.animate.create().duration(0.8),
    radius.animate.create().duration(0.4),
```

```
    point.animate.grow_from_center().duration(0.5).easing(Easing.SNAPPY),
    each=0.25,
))
scene.play(stagger(
    formula.animate.write().duration(0.8),
    explanation.animate.write().duration(0.8),
    each=0.2,
))
scene.play([theta.animate.set(2 * math.pi).duration(4).easing(Easing.LINEAR)])
scene.wait(1)
scene.render()
```

PUNTO DE CONTROL

La vista previa dura 8 segundos y la vuelta ocupa del segundo 3 al 7. Arrastra la barra de reproducción del editor: en el segundo 4 el punto está arriba ($\theta = \pi/2$), en el 5 a la izquierda, en el 6 abajo y en el 7 de vuelta a la derecha. El radio une siempre el centro con el punto.

La reactividad tiene más herramientas: valores calculados, rastros y updaters para simulaciones. Las encontrarás en la guía Reactividad. En el siguiente capítulo, Del círculo a la onda, usaremos `theta` para dibujar la curva seno.

CAPÍTULO

11. Del círculo a la onda

Una recta numerada, la curva seno y una proyección gobernadas por el mismo ángulo

11.1. Objetivo

Este capítulo completa la explicación. A la derecha del círculo aparecerá una recta numerada de 0 a 2π. Mientras el punto gira, un segundo punto avanza por la recta a la misma altura que el primero, unido a él por una línea de proyección, y deja dibujada la onda seno.

No añadiremos otro movimiento: la recta, la onda y el segundo punto dependen del mismo `theta` que ya mueve el círculo. La animación de `theta` del capítulo anterior no cambia.

11.2. Cambios

11.2.1. Nuevos imports

```
import math

from gaanim import WHITE, YELLOW, Axis, Color, Easing, Scene, computed, stagger
```

`Axis` describe una escala numérica y `computed` crea valores calculados a partir de otros.

11.2.2. Una recta numerada con π

Añade una zona nueva, `# Recta y onda`, después de la zona `# Círculo`:

```
# Recta y onda
axis = (
    Axis.linear(0, 2 * math.pi)
    .ticks(math.pi)
    .numbers("pi", denominator=1)
    .style(color=MUTED, number_color=MUTED)
)
number_line = scene.viz.number_line(axis, length=8)
number_line.drawable().move_to(2, CENTER[1])
```

`Axis.linear(0, 2 * math.pi)` va de 0 a 2π, la vuelta completa. `ticks` pone una marca en cada múltiplo de π y `numbers("pi", denominator=1)` las rotula como `0`, `π` y `2π`, sin escribir los textos a mano. `style` usa el color secundario de la paleta.

`scene.viz.number_line` convierte la escala en una recta de 8 unidades de largo. La movemos a la altura del centro del círculo, de `x = -2` a `x = 6`.

11.2.3. La onda

```
wave = number_line.function(math.sin, normal_scale=R, reveal=theta)
wave.stroke(PRIMARY, 0.04).no_fill()
```

`number_line.function` dibuja una función perpendicular a la recta: para cada valor de la recta, la curva se separa de ella `math.sin(valor)` por `normal_scale`. Con `normal_scale=R`, la onda alcanza la misma altura máxima que el punto sobre el círculo.

`reveal=theta` muestra la curva solo hasta el valor actual de `theta`. Con `theta = 0` no se ve nada; a medida que el ángulo crece, la onda se dibuja. Por eso no necesita animación de entrada.

11.2.4. El punto de la onda y la proyección

```
height = computed(lambda angle: R * math.sin(angle), inputs=[theta])
wave_dot = scene.geometry.dot(0.1).fill(ACCENT)
wave_dot.follow(number_line.point_ref(theta, normal_offset=height))
projection = scene.geometry.line(point, wave_dot).stroke(ACCENT, 0.025)
```

`computed` crea un valor que Gaanim recalcula cuando cambian sus entradas: `height` vale siempre `R * sin(theta)`, la altura del punto respecto al centro del círculo.

`number_line.point_ref(theta, normal_offset=height)` es el lugar de la recta que corresponde a `theta`, desplazado `height` en perpendicular. `wave_dot` lo sigue, igual que `point` sigue a `circle_ref`.

`projection` une los dos puntos. Como ambos están siempre a la misma altura, la línea es siempre horizontal: esa es la prueba visual de que la onda es la altura del punto.

11.2.5. La entrada de la recta

En la zona `# Línea de tiempo`, añade una entrada justo antes de la animación de `theta`:

```
scene.play(stagger(
    number_line.animate.create().duration(0.8),
    projection.animate.create().duration(0.4),
    wave_dot.animate.grow_from_center().duration(0.5).easing(Easing.SNAPPY),
    each=0.25,
))
```

Es la misma estructura que la entrada del círculo: primero la recta, después la línea y al final el punto con rebote. La vuelta de `theta` empieza ahora en el segundo 4.

> **IDEA CLAVE**
>
> Un solo parámetro evita la deriva. Si el punto y la onda tuvieran cada uno su propia velocidad, bastaría un pequeño error para que dejaran de coincidir. Aquí el círculo, la onda y la proyección se calculan del mismo `theta`, así que coinciden en cualquier instante.

11.3. Archivo completo

```
import math

from gaanim import WHITE, YELLOW, Axis, Color, Easing, Scene, computed, stagger

# Paleta
BACKGROUND = Color(15, 23, 42)
PRIMARY = Color(96, 165, 250)
ACCENT = YELLOW
MUTED = Color(148, 163, 184)

scene = Scene(frame=(16, 9), background=BACKGROUND, margin=0.6)

CENTER = (-4.5, -1.0)
```

```
R = 1.5

# Texto
title = scene.text("Del círculo al seno", role="title")
title.fill(WHITE).move_to(0, 3.4)
caption = scene.text("Un punto que gira a radio constante", role="subtitle")
caption.fill(MUTED).move_to(0, 2.8)

formula = scene.text("$y = r sin(theta)$", role="subtitle").fill(WHITE)
explanation = scene.text("La altura del punto dibuja la onda.", role="body")
explanation.fill(MUTED)
panel = scene.layout.column([formula, explanation], gap=0.2, width=6.5)
panel.move_to(3.5, 1.6)

# Círculo
theta = scene.viz.parameter(0.0)
orbit = scene.geometry.circle(R).stroke(PRIMARY, 0.05).no_fill().move_to(*CENTER)
circle_ref = scene.geometry.polar_point(CENTER, R, theta)
point = scene.geometry.dot(0.125).fill(ACCENT).follow(circle_ref)
radius = scene.geometry.line(CENTER, point).stroke(MUTED, 0.025)
point.z_index(1)

# Recta y onda
axis = (
    Axis.linear(0, 2 * math.pi)
    .ticks(math.pi)
    .numbers("pi", denominator=1)
    .style(color=MUTED, number_color=MUTED)
)
number_line = scene.viz.number_line(axis, length=8)
number_line.drawable().move_to(2, CENTER[1])

wave = number_line.function(math.sin, normal_scale=R, reveal=theta)
wave.stroke(PRIMARY, 0.04).no_fill()

height = computed(lambda angle: R * math.sin(angle), inputs=[theta])
wave_dot = scene.geometry.dot(0.1).fill(ACCENT)
wave_dot.follow(number_line.point_ref(theta, normal_offset=height))
projection = scene.geometry.line(point, wave_dot).stroke(ACCENT, 0.025)

# Línea de tiempo
scene.play(stagger(
    title.animate.write().duration(0.8),
    caption.animate.fade_in().duration(0.5),
    each=0.5,
))
scene.play(stagger(
    orbit.animate.create().duration(0.8),
    radius.animate.create().duration(0.4),
    point.animate.grow_from_center().duration(0.5).easing(Easing.SNAPPY),
    each=0.25,
))
scene.play(stagger(
    formula.animate.write().duration(0.8),
    explanation.animate.write().duration(0.8),
    each=0.2,
))
scene.play(stagger(
    number_line.animate.create().duration(0.8),
    projection.animate.create().duration(0.4),
    wave_dot.animate.grow_from_center().duration(0.5).easing(Easing.SNAPPY),
    each=0.25,
))
scene.play([theta.animate.set(2 * math.pi).duration(4).easing(Easing.LINEAR)])
scene.wait(1)
scene.render()
```

PUNTO DE CONTROL

La vista previa dura 9 segundos. Entre el segundo 3 y el 4 aparecen la recta, la línea de proyección y el punto de la onda; la vuelta ocupa del 4 al 8. Usa la barra de reproducción del editor para comprobar estos instantes:

- En el segundo 5 ($\theta = \pi/2$), el punto está arriba del círculo y el de la onda en su cresta, a mitad de camino entre `0` y π.
- En el 6 ($\theta = \pi$), los dos puntos están a la altura de la recta y el de la onda coincide con la marca π.
- En el 7 ($\theta = 3\pi/2$), ambos están en su punto más bajo.

En todos los instantes, la proyección es horizontal.

En el último capítulo, Terminar el proyecto, cerraremos la animación, la revisaremos y la exportaremos.

CAPÍTULO

12. Terminar el proyecto

Un cierre para la animación, el archivo final, la revisión y la exportación

12.1. Objetivo

La explicación ya funciona. En este capítulo le damos un final: cuando el punto completa la vuelta, la proyección se retira y la escena se queda quieta un momento con el círculo y la onda. Después comprobaremos el proyecto y lo exportaremos a video.

12.2. Cambios

12.2.1. Describir el archivo

Añade un docstring al principio de `main.py`. Es lo primero que leerá quien abra el archivo, incluido tú dentro de unos meses:

```
"""Del círculo al seno: el proyecto final del tutorial de Gaanim.

Un punto gira sobre un círculo y su altura dibuja una onda seno. Un único
parámetro, ``theta``, gobierna el punto, el radio, la onda y su proyección.
"""
```

12.2.2. Un cierre que respira

Al final de la zona `# Línea de tiempo`, sustituye `scene.wait(1)` por:

```
scene.play([projection.animate.fade_out().duration(0.5)])
scene.wait(1.5)
```

La línea de proyección era un andamio para explicar la relación entre los dos puntos. Al terminar la vuelta queda superpuesta a la recta, así que la retiramos con `fade_out`. La pausa final de 1.5 segundos da tiempo a mirar el resultado antes de que el video termine: una exportación acaba exactamente en el último instante de la línea de tiempo.

12.3. Archivo completo

Este es el programa terminado. Es el mismo archivo que se instala con Gaanim en `_docs/tutorial/circulo_al_seno.py`, dentro de la carpeta del paquete `gaanim`:

```
"""Del círculo al seno: el proyecto final del tutorial de Gaanim.

Un punto gira sobre un círculo y su altura dibuja una onda seno. Un único
parámetro, ``theta``, gobierna el punto, el radio, la onda y su proyección.
"""

import math
```

```python
from gaanim import WHITE, YELLOW, Axis, Color, Easing, Scene, computed, stagger

# Paleta
BACKGROUND = Color(15, 23, 42)
PRIMARY = Color(96, 165, 250)
ACCENT = YELLOW
MUTED = Color(148, 163, 184)

scene = Scene(frame=(16, 9), background=BACKGROUND, margin=0.6)

CENTER = (-4.5, -1.0)
R = 1.5

# Texto
title = scene.text("Del círculo al seno", role="title")
title.fill(WHITE).move_to(0, 3.4)
caption = scene.text("Un punto que gira a radio constante", role="subtitle")
caption.fill(MUTED).move_to(0, 2.8)

formula = scene.text("$y = r sin(theta)$", role="subtitle").fill(WHITE)
explanation = scene.text("La altura del punto dibuja la onda.", role="body")
explanation.fill(MUTED)
panel = scene.layout.column([formula, explanation], gap=0.2, width=6.5)
panel.move_to(3.5, 1.6)

# Círculo
theta = scene.viz.parameter(0.0)
orbit = scene.geometry.circle(R).stroke(PRIMARY, 0.05).no_fill().move_to(*CENTER)
circle_ref = scene.geometry.polar_point(CENTER, R, theta)
point = scene.geometry.dot(0.125).fill(ACCENT).follow(circle_ref)
radius = scene.geometry.line(CENTER, point).stroke(MUTED, 0.025)
point.z_index(1)

# Recta y onda
axis = (
    Axis.linear(0, 2 * math.pi)
    .ticks(math.pi)
    .numbers("pi", denominator=1)
    .style(color=MUTED, number_color=MUTED)
)
number_line = scene.viz.number_line(axis, length=8)
number_line.drawable().move_to(2, CENTER[1])

wave = number_line.function(math.sin, normal_scale=R, reveal=theta)
wave.stroke(PRIMARY, 0.04).no_fill()

height = computed(lambda angle: R * math.sin(angle), inputs=[theta])
wave_dot = scene.geometry.dot(0.1).fill(ACCENT)
wave_dot.follow(number_line.point_ref(theta, normal_offset=height))
projection = scene.geometry.line(point, wave_dot).stroke(ACCENT, 0.025)

# Línea de tiempo
scene.play(stagger(
    title.animate.write().duration(0.8),
    caption.animate.fade_in().duration(0.5),
    each=0.5,
))
scene.play(stagger(
    orbit.animate.create().duration(0.8),
    radius.animate.create().duration(0.4),
    point.animate.grow_from_center().duration(0.5).easing(Easing.SNAPPY),
    each=0.25,
))
scene.play(stagger(
    formula.animate.write().duration(0.8),
    explanation.animate.write().duration(0.8),
    each=0.2,
))
```

```
scene.play(stagger(
    number_line.animate.create().duration(0.8),
    projection.animate.create().duration(0.4),
    wave_dot.animate.grow_from_center().duration(0.5).easing(Easing.SNAPPY),
    each=0.25,
))
scene.play([theta.animate.set(2 * math.pi).duration(4).easing(Easing.LINEAR)])
scene.play([projection.animate.fade_out().duration(0.5)])
scene.wait(1.5)
scene.render()
```

PUNTO DE CONTROL

La vista previa dura 10 segundos. Hasta el segundo 8 es igual que la del capítulo anterior; entre el 8 y el 8.5 la línea de proyección se desvanece y en los últimos 1.5 segundos quedan el círculo con su radio, la recta y la onda completa, con los dos puntos amarillos a la altura de la recta.

12.4. Comprobar el proyecto

Desde la carpeta del proyecto:

```
gaanim check .
```

`gaanim check` ejecuta la escena sin abrir ventana e informa de su duración, en este caso 10 segundos. Si algo falla, muestra el error de Python con su línea. Con `--strict` también falla ante las advertencias.

Después abre la escena con `gaanim .` y revísala entera, también a saltos con la barra de reproducción:

- Todo el texto queda dentro del margen de seguridad.
- Ningún objeto aparece antes de su animación de entrada.
- El radio, la proyección y la onda siguen al punto en cualquier instante.
- Las pausas son lo bastante largas para leer la fórmula.

12.5. Exportar

Cuando la escena esté lista, expórtala sin tocar el script:

```
gaanim export . --output exports/del-circulo-al-seno.mp4
```

La extensión elige el formato: `.mp4` y `.webm` para video, `.webp` y `.gif` para animaciones ligeras en una web o un documento, y `.png` para una secuencia de imágenes. Todos salvo PNG necesitan FFmpeg. `--quality` elige entre `draft` (rápido), `standard` (el valor por defecto) y `production`, y `--from` y `--to` exportan solo un tramo:

```
gaanim export . --output exports/vuelta.webp --from 4 --to 8
```

La referencia completa de estas opciones está en Línea de comandos.

> **IDEA CLAVE**
>
> Una buena animación no es la que usa más funciones de la API. Es la que mantiene una relación clara entre lo que se ve, el orden en que se revela y la idea que debe entender el espectador.

12.6. Lo que has aprendido

- `Scene` define un lienzo de unidades lógicas, con el origen en el centro y un margen de seguridad.
- `scene.text` y `scene.geometry` crean objetos y devuelven handles; `fill`, `stroke`, `no_fill` y `z_index` les dan estilo y orden.
- `.animate`, `scene.play`, `stagger`, `duration`, `Easing` y `scene.wait` construyen la línea de tiempo.
- `scene.layout.column` organiza contenido que depende de su medida y pasa a ser el dueño de la posición de sus hijos.
- Un `Parameter` animado con `theta.animate.set(...)` gobierna todo lo que depende de él: `polar_point`, `follow`, líneas con objetos como extremos y valores `computed`.
- `Axis`, `scene.viz.number_line`, `number_line.function(..., reveal=theta)` y `number_line.point_ref` convierten el ángulo en una curva sin deriva.
- `gaanim check` y `gaanim export` validan y producen la salida sin cambiar el script.

12.7. Cómo continuar

Modifica el proyecto antes de empezar otro. Guarda una copia y prueba, en este orden:

1. Cambia `R` a `1.2`: el círculo, el punto, la onda y la proyección se adaptan.
2. Da dos vueltas: lleva el eje y la animación de `theta` hasta `4 * math.pi`.
3. Añade una segunda curva con `number_line.function(math.cos, ...)` y otro color.
4. Anima el panel con otras entradas y easings de Movimiento.

Para ir más lejos, las guías resuelven tareas concretas: Layout para composiciones más ricas, Reactividad para rastros, valores calculados y simulaciones, Presentaciones para convertir la escena en diapositivas y Capturas y comparación para comprobar que un cambio no altera lo que se ve. Consulta la Referencia cuando tengas una pregunta concreta sobre una función.

PARTE III

Guías

Cómo resolver cada tarea: composición, reactividad, presentaciones y producción

CAPÍTULO

13. Layout

Diseña como en CSS: cajas, filas, grids, estilos, zonas y animación, sin una sola coordenada

Si sabes armar una página web, ya sabes componer en Gaanim. `scene.layout` te da **cajas** que funcionan como un `div` de CSS: tienen padding, gap, borde, radio y sombra; ponen a sus hijos en fila, en columna, en una rejilla o en capas; miden el texto y reparten el espacio. Tú describes la estructura y Gaanim coloca cada cosa, en cualquier formato y durante toda la animación.

```
from gaanim import BoxStyle, Scene

scene = Scene(frame=(16, 9), theme="paper", background="#eef2ff", margin=0.5)
L = scene.layout
L.classes(
    card=BoxStyle(padding="28px", gap="14px", radius="24px", background="white",
                  shadow={"color": "#1e1b4b26", "y": "-10px", "blur": "30px"}),
    pill=BoxStyle(direction="row", padding=("6px", "16px"), radius="full", font_size="20px"),
)

def stat(label, value, delta, color):
    return L.box(
        L.box(label, font_size="20px", color="#64748b"),
        L.row(L.box(value, font_size="52px", weight=700),
              L.box(delta, class_="pill", background=color + "22", color=color),
              gap="12px", align="center"),
        class_="card", grow=1,
    )

cards = L.row(
    stat("Visitas", "48.2k", "+12%", "#16a34a"),
    stat("Registros", "1 284", "+4%", "#16a34a"),
    stat("Rebote", "31%", "-2%", "#dc2626"),
    gap="24px", width="fill",
)
page = L.column(
    L.box("Panel semanal", font_size="64px", weight=700, color="#1e1b4b"),
    cards,
    gap="36px", within="safe", width="fill", height="fill", justify="center",
)
scene.play([page.animate.fade_in().duration(0.6)])
cards.add(stat("Ventas", "$9.1k", "+30%", "#4f46e5"), duration=0.7)
scene.wait(0.6)
scene.render()
```

Nada de lo anterior usa coordenadas: la tarjeta nueva entra y las demás se hacen sitio solas.

13.1. Tu primera caja

`scene.layout.box(...)` crea una caja. Sus hijos pueden ser objetos de la escena, otras cajas o simplemente cadenas de texto, que se convierten en `Text` con la tipografía de la caja.

```
from gaanim import Scene

scene = Scene(frame=(16, 9), theme="paper")
L = scene.layout

tag = L.box("Nuevo", padding=("8px", "20px"), radius="full",
            background="#4f46e5", color="white", font_size="32px")

scene.play([tag.animate.fade_in().duration(0.5)])
scene.render()
```

Una caja sin `within` se coloca donde la ponga su padre; si es la raíz, se centra en el área segura. Las propiedades usan las mismas ideas que CSS:

- **Longitudes**: un número (unidades de escena), `"24px"` (píxeles de diseño: el fotograma mide 1080 px de alto, sea cual sea la resolución de salida) o un token del tema como `"space_md"`.
- **Tamaños** (`width`, `height`): una longitud, `"50%"` del padre, `"hug"` (lo que mide el contenido) o `"fill"` (todo el espacio libre).
- **Relleno y margen**: como en CSS, uno, dos, tres o cuatro valores: `padding=("8px", "20px")` es 8 px arriba y abajo, 20 px a los lados.

13.2. Filas y columnas

`L.row` y `L.column` son cajas con dirección. `gap` separa a los hijos, `justify` los reparte en el eje principal y `align` los alinea en el otro.

```
from gaanim import BoxStyle, Scene

scene = Scene(frame=(16, 9), theme="paper", margin=0.6)
L = scene.layout
chip = BoxStyle(padding=("10px", "18px"), radius="10px", background="#e0e7ff", font_size="26px")

def lane(justify):
    return L.row(L.box("A", style=chip), L.box("B", style=chip), L.box("C", style=chip),
                 justify=justify, width="fill", padding="8px", radius="14px", background="#f1f5f9")

page = L.column(
    *[L.row(L.box(name, font_size="22px", color="#64748b", width="170px"), lane(name).item(grow=1),
            align="center", width="fill")
      for name in ("start", "center", "end", "between", "evenly")],
    gap="14px", within="safe", width="fill",
)
scene.play([page.animate.fade_in().duration(0.5)])
scene.render()
```


- `justify`: `start`, `center`, `end`, `between`, `around`, `evenly`.
- `align`: `start`, `center`, `end`, `stretch` (estira a todos los hijos), `baseline`.
- `wrap=True` pasa los hijos a la línea siguiente cuando no caben, ideal para etiquetas.

13.3. El modelo de caja

Una caja dibuja su propio fondo: `background`, `border`, `border_width`, `radius` (`"full"` para una píldora), `shadow` y `clip` para recortar lo que se salga de sus bordes redondeados.

```
from gaanim import Scene

scene = Scene(frame=(16, 9), theme="paper", margin=0.6)
L = scene.layout

plain = L.box("Borde", padding="28px", radius="18px", border="#94a3b8", border_width="2px")
lifted = L.box("Sombra", padding="28px", radius="18px", background="white",
               shadow={"color": "#0f172a33", "y": "-8px", "blur": "24px"})
clipped = L.stack(scene.geometry.circle(1.5).fill("#f97316").item(anchor="top"),
                  width="220px", height="110px", radius="18px", clip=True, background="#fff7ed")
page = L.row(plain, lifted, clipped, gap="48px", align="center", within="safe",
             font_size="30px")
scene.play([page.animate.fade_in().duration(0.5)])
scene.render()
```

La tipografía también se hereda: `font_size`, `color`, `weight`, `italic`, `font`, `line_spacing`, `letter_spacing`… definidas en una caja se aplican a todas las cadenas de texto que contiene, como el `font-size` de CSS.

13.3.1. La caja del texto: `text_box`

Por defecto un texto ocupa líneas completas, de la ascendente a la descendente: dos textos del mismo estilo miden lo mismo aunque uno tenga «g» o «Á». Para calcar un diseño en el que las mayúsculas tocan el borde de la caja, usa `text_box="cap"`: el texto va de la altura de las mayúsculas a su línea base. `text_box="ink"` se ajusta a la tinta de los glifos.

```
from gaanim import Scene

scene = Scene(frame=(16, 9), theme="paper", margin=0.6)
L = scene.layout

def sample(mode):
    return L.column(
        L.box(f'text_box="{mode}"', font_size="24px", color="#64748b"),
        L.box("Título", font_size="96px", weight=700, text_box=mode,
              background="#e0e7ff", radius="8px"),
```

```
        gap="12px", align="center",
    )

page = L.row(sample("line"), sample("cap"), sample("ink"), gap="64px",
             align="end", within="safe")
scene.play([page.animate.fade_in().duration(0.5)])
scene.render()
```


13.4. Estilos reutilizables: `BoxStyle` y clases

Un `BoxStyle` agrupa propiedades, como una regla CSS. Pásalo con `style=`, derívalo con `.but(...)`, o regístralo por nombre y úsalo con `class_=`.

```
from gaanim import BoxStyle, Scene

scene = Scene(frame=(16, 9), theme="paper")
L = scene.layout
L.classes(
    pill=BoxStyle(direction="row", padding=("6px", "16px"), radius="full", font_size="24px",
                  background="#4f46e5", color="white"),
    soft=BoxStyle(background="#e0e7ff", color="#3730a3"),
)
warning = BoxStyle(background="#fef3c7", color="#92400e")

row = L.row(
    L.box("Diseño", class_="pill"),
    L.box("Datos", class_="pill soft"),
    L.box("Revisión", class_="pill", style=warning),
    L.box("Urgente", class_="pill", background="#dc2626"),
    gap="12px",
)
scene.play([row.animate.fade_in().duration(0.5)])
scene.render()
```

El orden de prioridad es el de CSS: clases, después `style=` y al final las propiedades escritas en la llamada. Varias clases se separan con espacios y la última gana.

13.5. Grids

`L.grid` coloca a sus hijos en celdas. `columns` y `rows` aceptan un número de pistas o una lista: longitudes, `"N%"`, `"auto"` o fracciones `"1fr"` del espacio libre. Cada hijo puede ocupar varias celdas con `row_span` y `column_span`.

```
from gaanim import BoxStyle, Scene

scene = Scene(frame=(16, 9), theme="paper", margin=0.5)
L = scene.layout
cell = BoxStyle(padding="16px", radius="14px", background="#dbeafe", font_size="26px",
                align="center", justify="center")
hot = cell.but(background="#fde68a")

bento = L.grid(
    L.box("Destacado", style=hot).item(row_span=2),
    L.box("Ancho", style=hot).item(column_span=2),
    L.box("A", style=cell), L.box("B", style=cell),
    L.box("Pie", style=hot).item(column_span=3),
    columns=["260px", "1fr", "2fr"], rows=["1fr", "1fr", "90px"],
    gap="14px", within="safe", width="fill", height="fill",
)
scene.play([bento.animate.fade_in().duration(0.5)])
scene.render()
```

Las celdas se estiran por defecto. Con `align` en la rejilla, o `align_self` en un hijo, lo colocas arriba, en el centro o abajo de su celda.

13.6. Cada hijo decide cómo se coloca: `item()`

Cualquier objeto dentro de una caja acepta `.item(...)`, que es el CSS del elemento dentro de su contenedor:

- `grow` / `shrink` / `basis`: cuánto crece o encoge en el eje principal.
- `align_self`: su alineación propia en el otro eje.
- `margin`: espacio alrededor; `"auto"` se come el espacio libre, así que `margin=(0, 0, 0, "auto")` empuja un botón al extremo derecho.
- `row`, `column`, `row_span`, `column_span`: su celda en un grid.
- `fit`: cómo se ajusta una imagen o figura a su caja (`contain`, `cover`, `stretch`, `scale_down`).
- `anchor`, `offset`, `absolute`: posición dentro de una capa (`L.stack`) o fuera del flujo.
- `width` / `height`: un tamaño de layout que sustituye al propio del objeto.

```
from gaanim import Scene

scene = Scene(frame=(16, 9), theme="paper", margin=0.6)
L = scene.layout

bar = L.row(
    L.box("Gaanim", font_size="30px", weight=700),
    L.box("Docs", font_size="24px", color="#475569"),
    L.box("Blog", font_size="24px", color="#475569"),
    L.box("Empezar", font_size="24px", color="white", background="#4f46e5", radius="full",
          padding=("8px", "22px")).item(margin=(0, 0, 0, "auto")),
    gap="28px", align="center", width="fill", padding=("14px", "24px"),
    radius="18px", background="white", border="#e2e8f0",
)
progress = L.row(L.box("Carga", font_size="24px"),
                 L.box(height="14px", radius="full", background="#4f46e5").item(grow=1),
                 L.box("72%", font_size="24px", weight=700),
                 gap="16px", align="center", width="fill")
page = L.column(bar, progress, gap="60px", within="safe", width="fill", justify="center",
                height="fill")
```

```
scene.play([page.animate.fade_in().duration(0.5)])
scene.render()
```

13.7. Animar la estructura

Una caja es un objeto más de la escena: `page.animate.fade_in()` y el resto de animaciones actúan sobre ella entera. Además, todos los cambios de estructura aceptan `duration=`: los hijos se deslizan a su nuevo sitio, lo que entra aparece con un fundido y lo que sale se desvanece.

- `box.add(hijo, at=1, duration=0.6)`, `box.remove(hijo, duration=0.6)`, `box.replace(viejo, nuevo, duration=0.6)`.
- `box.set(columns=2, gap="40px", duration=0.8)` cambia cualquier propiedad.
- `hijo.item(column_span=2, duration=0.8)` cambia las reglas de un hijo.

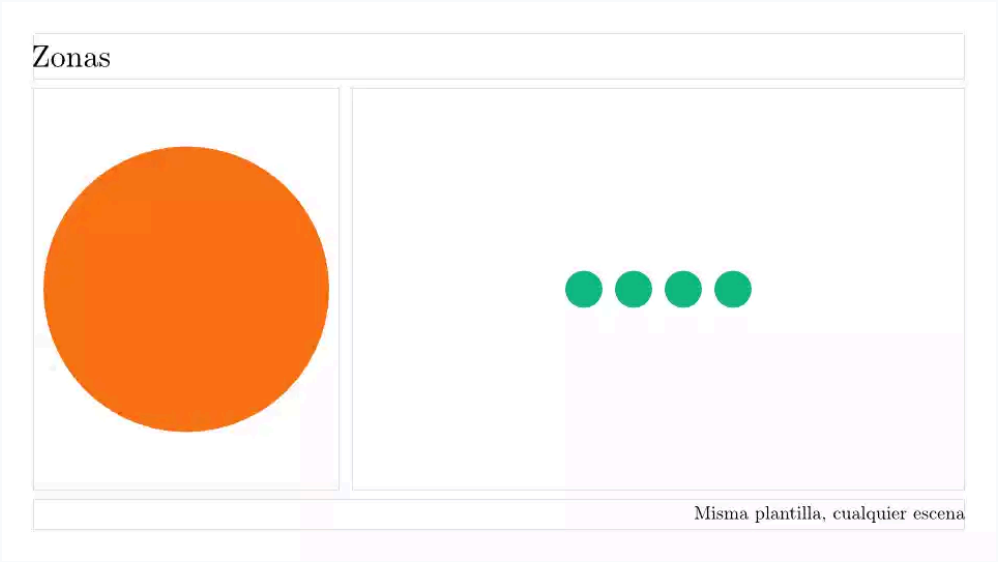
Cada cambio con `duration` avanza el cursor de la escena, como `scene.play`. Con `advance=False` el siguiente cambio empieza a la vez.

```
from gaanim import BoxStyle, Scene

scene = Scene(frame=(16, 9), theme="paper")
L = scene.layout
card = BoxStyle(padding="20px", radius="16px", background="#e0e7ff", font_size="28px",
                width="170px", align="center")

board = L.grid(*[L.box(name, style=card) for name in ("Uno", "Dos", "Tres")],
               columns=3, gap="18px")
scene.play([board.animate.fade_in().duration(0.4)])
board.add(L.box("Nueva", style=card, background="#bbf7d0"), at=1, duration=0.6)
board.set(columns=2, duration=0.8)
board.replace(board[0], L.box("Cambio", style=card, background="#fecaca"), duration=0.6)
board.set(gap="40px", duration=0.8, advance=False)
board[1].item(column_span=2, width="fill", duration=0.8)
scene.wait(0.4)
scene.render()
```

13.8. Zonas: plantillas que no atan a los objetos

A veces no quieres que un objeto viva dentro de una caja: un rótulo que entra deslizándose, una fórmula que cruza la pantalla, una ilustración libre. Las **zonas** dividen una región en rectángulos con nombre y te dejan colocar ahí objetos que siguen siendo independientes.

```
from gaanim import Scene, Zones

scene = Scene(frame=(16, 9), theme="paper", margin=0.5)
L = scene.layout
page = Zones.rows(["90px", "1fr", "60px"], names=["cabecera", "cuerpo", "pie"], gap="16px")
split = Zones.columns(["1fr", "2fr"], names=["lateral", "principal"], gap="24px")

z = L.zones(page)
body = z["cuerpo"].split(split)
for zona in (z["cabecera"], z["pie"], *body):  # contornos solo para verlas
    scene.geometry.rect(zona.width, zona.height).no_fill().stroke("#cbd5e1", 0.02).place(zona)

title = scene.text("Zonas", size=0.5).place(z["cabecera"], anchor="left")
note = scene.text("Misma plantilla, cualquier escena", size=0.3).place(z["pie"], anchor="right")
ball = scene.geometry.circle(1).fill("#f97316").place(body["lateral"], fit="contain", padding="20px")
dots = [scene.geometry.circle(0.3).fill("#10b981") for _ in range(4)]
body["principal"].arrange(*dots, gap="24px")

scene.play([ball.animate.place(z["cabecera"], anchor="right", fit="contain").duration(0.8)])
scene.render()
```



- `L.zones(plantilla)` aplica la plantilla al área segura (respeta el `margin` de la escena); `within="frame"`, una zona o un objeto cambian la región.
- `Zones.rows`, `Zones.columns` y `Zones.grid` usan las mismas pistas que los grids. Una plantilla se reutiliza en otra escena, en una zona (`zona.split(plantilla)`) o dentro de un objeto.
- `objeto.place(zona, anchor=..., fit=..., padding=..., offset=...)` lo coloca una vez; `objeto.animate.place(...)` lo lleva hasta ahí.
- `zona.arrange(*objetos)` los alinea en fila o columna sin adoptarlos, y `zonas.fill(*objetos)` pone uno en cada zona.

13.9. Componentes

Un componente es una función de Python que devuelve una caja. Con `@component` Gaanim comprueba sus argumentos y tu editor los autocompleta. Cada llamada crea objetos nuevos.

```
from gaanim import Scene, component

scene = Scene(frame=(16, 9), theme="paper")
L = scene.layout

@component
def profile(scene, *, name: str, role: str, color: str = "#4f46e5"):
    initials = "".join(part[0] for part in name.split())
    return L.row(
        L.box(initials, width="64px", height="64px", radius="full", background=color,
              color="white", font_size="26px", weight=700, align="center", justify="center"),
        L.column(L.box(name, font_size="28px", weight=700),
                 L.box(role, font_size="20px", color="#64748b")),
        gap="18px", align="center", padding="18px", radius="20px", background="white",
        border="#e2e8f0",
    )

team = L.column(
    profile(scene, name="Ana Ruiz", role="Diseño"),
    profile(scene, name="Luis Paz", role="Datos", color="#0891b2"),
    profile(scene, name="Eva Gil", role="Producción", color="#f97316"),
    gap="16px",
)
scene.play([team.animate.fade_in().duration(0.5)])
scene.render()
```

Las plantillas de presentación (`title_slide`, `lecture`, `comparison`…) son componentes escritos exactamente así.

13.10. Recorrer una caja

Una tarjeta suele animarse pieza por pieza: el fondo, el título, cada fila. En lugar de bajar por el árbol a mano, `box.walk()` devuelve las piezas a cualquier profundidad, en orden de dibujo y con el fondo de cada caja antes de su contenido; con `boxes=True` incluye también las cajas anidadas.

- `box.find(text="Título")`, `box.find(type=Text)` y `box.find_all(...)` buscan una pieza por su clase, su texto o un predicado `where=`.
- `box.cascade(each=0.1).fade_in()` anima todas las piezas en cascada con cualquier animación de `.animate`. Para elegir qué anima cada pieza (o saltarse alguna), usa `box.stagger(lambda pieza: ...)`.
- `box.each(lambda pieza: pieza.fill(...), type=Text)` aplica un ajuste inmediato a cada pieza y devuelve la caja.
- `box.swap(a, b)`, `box.move_child(hijo, 0)` y `box.reverse()` reordenan los hijos; con `duration=` se deslizan a su nuevo sitio.

Antes de su turno, una pieza que entra con `grow_*` o `fade_in` no se ve, aunque la caja se reorganice después.

```
from gaanim import BoxStyle, Scene, Text

scene = Scene(frame=(16, 9), theme="paper")
L = scene.layout
chip = BoxStyle(padding="14px", radius="10px", background="#e0e7ff", font_size="26px",
                width="130px", align="center")
row = L.row(*[L.box(name, style=chip) for name in ("Uno", "Dos", "Tres")], gap="16px")
card = L.column(L.box("Tarjeta", role="title"), row, gap="24px", padding="30px",
                align="center", background="#f8fafc", border="#cbd5e1", radius="18px")

scene.play(card.cascade(each=0.12).grow_from_center())
card.each(lambda piece: piece.fill("#1e3a8a"), type=Text)
row.swap(row[0], row[2], duration=0.6)
row.reverse(duration=0.6)
```

```
scene.wait(0.4)
scene.render()
```


13.11. Coordenadas o cajas

Usa cajas y zonas para la composición: títulos, tarjetas, paneles, listas, rótulos. Usa coordenadas cuando la posición tiene significado: el punto de una gráfica, una órbita, una trayectoria.

13.12. Animar dentro de una caja

La caja decide dónde descansa cada hijo, no cómo se mueve. Todas las animaciones de `.animate` funcionan sobre un hijo: `fade_in_from`, `shift_by`, `move_to`, `scale_by`, `rotate_by`, `grow_from_edge`… Parten del sitio que le dio la caja y no empujan a sus hermanos, como un `transform` de CSS. Si después la caja se reorganiza (un `add`, un `replace`, un texto que cambia), el hijo conserva el desplazamiento que le diste respecto a su sitio.

```
from gaanim import Direction, Scene

scene = Scene(frame=(16, 9), theme="paper", margin=0.5)
L = scene.layout
title = L.box("Resultados", font_size="96px", weight=700, color="#312e81")
points = L.column(
    L.box("Menos pasos", font_size="44px", padding=("14px", "28px"), radius="16px",
          background="#e0e7ff"),
    L.box("Menos errores", font_size="44px", padding=("14px", "28px"), radius="16px",
          background="#e0e7ff"),
    gap="20px",
)
slide = L.column(title, points, gap="48px", align="center", justify="center",
                 width="fill", height="fill")

scene.play([title.animate.fade_in_from(Direction.LEFT).duration(0.5)])
scene.play([points[0].animate.fade_in_from(Direction.UP).duration(0.4),
            points[1].animate.fade_in_from(Direction.UP).duration(0.4).delay(0.15)])
scene.play([title.animate.scale_by(1.15).rotate_by(-0.05).duration(0.4)])
# La columna gana un punto: los demás se deslizan y el título sigue girado.
points.add(L.box("Más tiempo", font_size="44px", padding=("14px", "28px"),
```

```
                radius="16px", background="#fde68a"), duration=0.5)
scene.render()
```


`move_to(x, y)` usa coordenadas de la escena también para un hijo. Lo único reservado a la caja es colocar a un hijo *al declararlo*: `move_to`, `next_to` o `to_edge` inmediatos sobre un hijo lanzan `LayoutOwnershipError`. `scale_to` o `rotate_to` inmediatos sí funcionan y, como las animaciones, no cambian el sitio del hijo; `box.reflow()` lo mide con su escala actual y le hace sitio. Para eso ajusta `item(offset=...)`, mueve la caja entera o saca al hijo con `box.detach(hijo)`. Si algo no encaja, `box.diagnostics()` y `scene.layout.check_layout()` explican qué restricción o texto no se pudo cumplir. En el editor, `O` y después `K` activan el overlay **Layout**: muestra cada caja y zona y, al pasar el cursor por una caja, su padding, sus márgenes y sus gaps, como las herramientas de desarrollo de un navegador.

Consulta la referencia de Layout para todas las propiedades, firmas y errores. Los ejemplos `layout_*` y `ui_*` del repositorio muestran interfaces completas: un panel de métricas, una app de chat y un rótulo de vídeo.

CAPÍTULO

14. Movimiento

Easings, springs, repetición, stagger, ruido y trayectorias para que cada movimiento tenga intención

14.1. Movimiento con intención

En esta guía aprendes a decidir **cómo** se mueve algo, no solo adónde va: elegir un easing por lo que comunica, usar springs sin pensar en física, repetir y hacer bucles, convertir una rejilla en una onda, dar vida a objetos quietos con ruido determinista y mover objetos por trayectorias.

Todo lo que ves aquí es una función pura del tiempo: la vista previa, un seek a cualquier instante y la exportación producen el mismo fotograma.

```
from gaanim import CYAN, Easing, Scene, stagger

scene = Scene(frame=(16, 9), background="#0f172a")
dots = [
    scene.geometry.dot(0.22).fill(CYAN).move_to(x - 4.5, y - 1.5)
    for x in range(10)
    for y in range(4)
]
scene.play(stagger(
    *[d.animate.grow_from_center().duration(0.6).easing(Easing.BOUNCY) for d in dots],
    total=0.8,
    origin="center",
))
scene.wait(0.2)
scene.render()
```

Una onda desde el centro con un spring con rebote: dos decisiones de movimiento, dos argumentos.

14.1.1. Elegir un easing

El easing decide cómo se reparte el recorrido dentro de la duración. Si no indicas ninguno, las animaciones de propiedades (`move_to`, `scale_to`, `fill`…) usan `Easing.SMOOTH`, que acelera al salir y frena al llegar.

Easing	Cuándo usarlo
`Easing.SMOOTH`	Mover algo entre dos posiciones de reposo. Es el tono editorial por defecto.
`Easing.ease_out(curva)`	Entradas: el objeto llega rápido y se asienta. Se siente responsivo.
`Easing.ease_in(curva)`	Salidas: el objeto arranca despacio y se va acelerando.
`Easing.ease_in_out(curva)`	Como `SMOOTH`, con la intensidad de la curva elegida.
`Easing.LINEAR`	Velocidad constante: giros continuos, barras de progreso, recorridos uniformes.
`Easing.THERE_AND_BACK`	Ir y volver en una sola animación, por ejemplo un pulso.
`Easing.steps(n)`	Saltos discretos: relojes, contadores, animación «a tirones».

`EasingCurve` fija la intensidad: `QUADRATIC` es suave, `CUBIC` es el punto medio habitual, `QUARTIC`, `QUINTIC` y `EXPONENTIAL` son cada vez más marcadas, y `SINE` y `CIRCULAR` dan variantes más orgánicas.

```
from gaanim import CORAL, CYAN, Easing, EasingCurve, Scene

scene = Scene(frame=(16, 9), background="#0f172a")
entrada = scene.geometry.circle(0.5).fill(CYAN).move_to(-9, 1.2)
salida = scene.geometry.circle(0.5).fill(CORAL).move_to(0, -1.2)
scene.play([
    entrada.animate.move_to(0, 1.2).duration(0.8).easing(Easing.ease_out(EasingCurve.CUBIC)),
    salida.animate.move_to(9, -1.2).duration(0.8).easing(Easing.ease_in(EasingCurve.CUBIC)),
])
scene.render()
```

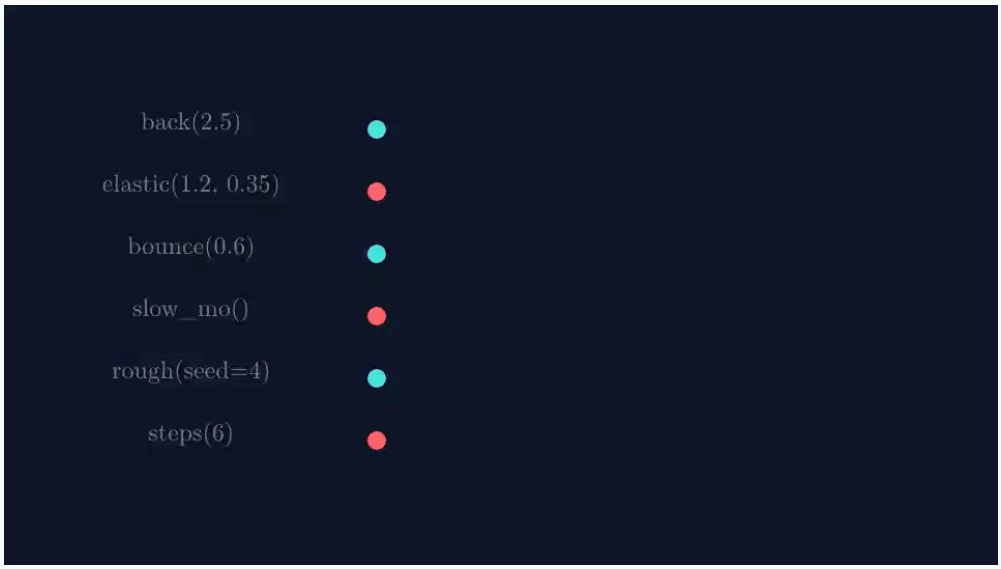
La regla práctica: **ease-out** para lo que entra, **ease-in** para lo que sale y **ease-in-out** para lo que cambia de sitio dentro del cuadro.

14.1.2. Easings con carácter

Las fábricas paramétricas añaden personalidad. Todas son deterministas y validan sus argumentos: un valor fuera de dominio lanza `ValueError`.

```
from gaanim import CORAL, CYAN, GRAY, Easing, Scene

scene = Scene(frame=(16, 9), background="#0f172a")
filas = [
    ("back(2.5)", Easing.back(2.5)),
    ("elastic(1.2, 0.35)", Easing.elastic(1.2, 0.35)),
    ("bounce(0.6)", Easing.bounce(0.6)),
    ("slow_mo()", Easing.slow_mo()),
    ("rough(seed=4)", Easing.rough(0.8, points=16, seed=4)),
    ('steps(6)', Easing.steps(6)),
]
anims = []
for i, (nombre, easing) in enumerate(filas):
    y = 2.5 - i * 1.0
```

```
    scene.text(nombre).fill(GRAY).move_to(-5, y)
    punto = scene.geometry.dot(0.15).fill(CORAL if i % 2 else CYAN).move_to(-2, y)
    anims.append(punto.animate.move_to(5, y).duration(1.4).easing(easing))
scene.play(anims)
scene.wait(0.2)
scene.render()
```



- `back(overshoot)`: se pasa del destino y vuelve. Ideal para tarjetas y botones que «encajan».
- `elastic(amplitude, period)`: oscila como una goma. Úsalo con moderación.
- `bounce(strength)`: rebota al llegar; `strength=0` es un cúbico.
- `slow_mo(linear_ratio, power)`: rápido, cámara lenta y rápido otra vez. Sirve para detener la mirada en el centro del recorrido.
- `rough(strength, points, seed)`: parpadeo y temblor deterministas, para fallos técnicos o luces que titilan.
- `squish(easing, start, end)`: aplica un easing solo dentro de una fracción del clip y mantiene los extremos fuera de ella.
- `from_svg("M0,0 C0.3,0 0.2,1.2 1,1")`: una curva dibujada en cualquier editor, con `x` como tiempo e `y` como progreso.
- `back`, `elastic` y `bounce` aceptan `mode="in"`, `"out"` o `"in_out"`.

14.1.3. Springs sin física

Un spring perceptual se describe por cómo se ve: `bounce` es el sobrepaso máximo, de `0` (sin rebote) a casi `1`, y el resorte se asienta dentro de la duración de la animación. `duration` controla el ritmo y `bounce` el carácter.

```
from gaanim import GOLD, Easing, Scene

scene = Scene(frame=(16, 9), background="#0f172a")
logo = scene.geometry.star(5, 1.2, 0.5).fill(GOLD).scale_by(0.4)
scene.play(logo.animate.scale_to(1.0).duration(0.6).easing(Easing.spring(bounce=0.35)))
scene.render()
```

Los presets con nombre cubren los casos habituales sin ajustar nada:

Preset	Carácter
`Easing.SMOOTH_SPRING`	Amortiguamiento crítico: llega lo antes posible sin pasarse.

`Easing.GENTLE`	Suave y pausado, con un sobrepaso apenas visible.
`Easing.QUICK`	Enérgico, con un sobrepaso pequeño que se asienta pronto.
`Easing.SNAPPY`	Rápido y nítido, con un 20 % de sobrepaso.
`Easing.BOUNCY`	Juguetón, con un 45 % de sobrepaso y rebotes visibles.

Si necesitas el modelo físico, `Easing.spring(stiffness=180, damping=14, mass=1.0, velocity=2.0)` sigue disponible. No mezcles `bounce` con parámetros físicos: la combinación lanza `ValueError`. Todos los springs terminan exactamente en el destino.

14.1.4. Tu propia curva con Easing.custom

`Easing.custom(función, samples=256)` convierte cualquier función de Python en un easing. La función se muestrea una sola vez al crear el easing, así que el render nunca vuelve a Python. Los valores deben ser finitos y estar en `[-1, 2]`.

```
import math
from gaanim import CYAN, Easing, Scene

scene = Scene(frame=(16, 9), background="#0f172a")
pelota = scene.geometry.circle(0.35).fill(CYAN).move_to(0, 3)
salto = Easing.custom(lambda t: 1 - abs(math.cos(3 * math.pi * t)) * (1 - t) ** 2)
scene.play(pelota.animate.move_to(0, -2.5).duration(1.2).easing(salto))
scene.render()
```

Usa `custom` cuando la curva sale de una fórmula. Si la tienes dibujada, usa `from_svg`.

14.1.5. Repetir, alternar y hacer bucles

`anim.repeat(count, yoyo=False, delay=0)` reproduce una animación varias veces: `duration` y `easing` describen un ciclo. Con `yoyo=True` los ciclos alternan de sentido, así que un número par termina donde empezó. `anim.loop(mode, until=segundos)` repite tantos ciclos completos como quepan: `"cycle"` reinicia, `"pingpong"` alterna y `"offset"` acumula, útil para giros continuos. `Composition.repeat(count)` repite un grupo completo.

```
import math
from gaanim import CORAL, CYAN, GOLD, Scene, parallel

scene = Scene(frame=(16, 9), background="#0f172a")
spinner = scene.geometry.square(1.2).fill(CYAN).move_to(-4.5, 0)
badge = scene.geometry.circle(0.7).fill(GOLD).move_to(-1, 0)
flecha = scene.geometry.arrow(1.5, 0, 3, 0).fill(CORAL)
a = scene.geometry.dot(0.2).fill(CYAN).move_to(5, 0.5)
b = scene.geometry.dot(0.2).fill(GOLD).move_to(5, -0.5)
scene.play([
    spinner.animate.rotate_by(math.pi / 2).duration(0.5).repeat(4),
    badge.animate.scale_to(1.2).duration(0.4).repeat(4, yoyo=True, delay=0.1),
    flecha.animate.shift_by(0.8, 0).duration(0.5).loop("pingpong", until=2.0),
    parallel(a.animate.shift_by(0, 0.6), b.animate.shift_by(0, -0.6)).repeat(2),
])
scene.render()
```

La duración total (`count · duration + (count − 1) · delay`) entra en la línea de tiempo, que sigue siendo finita: los bucles se pueden inspeccionar, recortar y exportar como cualquier otra animación.

14.1.6. Stagger: de una lista a una coreografía

`stagger(*anims, each=0.1)` retrasa cada animación por su índice. Con `origin` el retraso crece con la distancia a un origen, y la lista se convierte en una onda:

- `"start"` y `"end"`: desde el primer o el último objeto.
- `"center"`: desde el centro del conjunto hacia fuera.
- `"edges"`: desde los bordes hacia dentro.
- `"random"`: orden aleatorio fijado por `seed`.
- `(x, y)`: desde un punto de la escena, por ejemplo donde «cae» algo.

`each` es el retraso por paso de separación; `total` fija en su lugar la duración de toda la onda, que es más fácil de ajustar al ritmo de la escena. `grid="auto"` (por defecto) mide distancias con las posiciones declaradas y `grid=(filas, columnas)` usa celdas. `easing` da forma a la propia onda.

```
from gaanim import CYAN, GOLD, Easing, EasingCurve, Scene, distribute, stagger

scene = Scene(frame=(16, 9), background="#0f172a")
dots = [scene.geometry.dot(0.2).fill(CYAN).move_to(c - 5.5, 3 - r) for r in range(7) for c in
range(12)]
scene.play(stagger(
    *[d.animate.grow_from_center().duration(0.4) for d in dots],
    total=0.8, origin="center", easing=Easing.ease_in(EasingCurve.QUADRATIC),
))
scene.play(stagger(*[d.animate.fill(GOLD).duration(0.3) for d in dots], total=0.8, origin=(-5.5,
-3.0)))
scene.play(stagger(*[d.animate.opacity(0.3).duration(0.3) for d in dots], total=0.8, origin="random",
seed=7))
tamaños = distribute(dots, 1.0, 0.5, origin="edges")
scene.play([d.animate.scale_to(s).duration(0.5) for d, s in zip(dots, tamaños)])
scene.render()
```

`distribute(items, low, high, origin=...)` usa el mismo orden que `stagger`, pero reparte **valores** (tamaños, opacidades, colores) en lugar de tiempos. Los objetos cuya posición depende de un layout vuelven al orden por índice.

14.1.7. Coreografía con etiquetas

Cuando una secuencia crece, calcular retrasos a mano se vuelve frágil. `label(nombre)` marca un instante dentro de un `sequence`, y `Composition.insert(anim, at=...)` coloca animaciones relativas a él: `"golpe+0.15"`, `"<"` (junto con el anterior), `">-0.1"` (antes de que termine el anterior) o `"-=0.2"` (solapando el final actual).

```
from gaanim import CYAN, GOLD, GRAY, WHITE, Scene, label, sequence

scene = Scene(frame=(16, 9), background="#0f172a")
titulo = scene.text("Etiquetas", role="title").fill(WHITE).move_to(0, 2.4)
subtitulo = scene.text("tiempos relativos").fill(GRAY).move_to(0, 1.4)
logo = scene.geometry.circle(0.9).fill(GOLD).move_to(0, -1)
anillo = scene.geometry.circle(1.3).no_fill().stroke(CYAN, 0.08).move_to(0, -1)
intro = (
    sequence(
        titulo.animate.write().duration(0.8),
        label("golpe"),
        subtitulo.animate.fade_in().duration(0.4),
    )
    .insert(logo.animate.grow_from_center().duration(0.6), at="golpe+0.15")
    .insert(anillo.animate.create().duration(0.6), at="<")
)
print(intro.schedule().labels)
scene.play(intro)
scene.marker("clímax")
scene.render()

{'golpe': 0.8}
```

`scene.marker(nombre)` nombra un instante de la línea de tiempo global. El editor lo muestra en la barra de reproducción y `gaanim export --from clímax` lo acepta en lugar de segundos.

14.1.8. Aleatoriedad reproducible y ruido

`scene.random(seed)` devuelve un generador con semilla (`uniform`, `gauss`, `integer`, `choice`, `shuffle`). La misma semilla da los mismos valores en cualquier plataforma, así que puedes esparcir objetos sin perder reproducibilidad.

`scene.noise(...)` es otra cosa: una señal suave que cambia con el tiempo de la línea de tiempo, evaluada en Rust en cada fotograma. Se conecta con `computed(...)` a cualquier propiedad reactiva.

```
from gaanim import CYAN, GOLD, GRAY, WHITE, Scene, computed

scene = Scene(frame=(16, 9), background="#0f172a")
rng = scene.random(seed=42)
for _ in range(60):
    scene.geometry.dot(rng.uniform(0.02, 0.07)).fill(rng.choice([GRAY, WHITE, CYAN])).move_to(
        rng.uniform(-7.5, 7.5), rng.uniform(-4.2, 4.2)
    )
deriva = scene.noise(frequency=0.8, amplitude=1.0, octaves=3, seed=5)
hoja = scene.geometry.rect(2.4, 0.5).fill(GOLD)
hoja.rotate_to(computed(lambda v: 0.6 * v, inputs=[deriva]))
scene.wait(2.0)
scene.render()
```

`frequency` controla lo rápido que cambia, `amplitude` el rango (`center ± amplitude`) y `octaves` (1 a 8) añade detalle fino.

14.1.9. Dar vida: wiggle y osciladores

Para que un objeto quieto respire sin programar cada movimiento, añade un updater procedural. Ambos se **suman** a las animaciones del objeto en lugar de reemplazarlas, y `remove_updater()` los termina.

- `Updater.wiggle(position, rotation, scale, frequency, octaves, seed)`: temblor orgánico con ruido. Las amplitudes están en unidades de escena, radianes y fracción de escala.
- `Updater.oscillate(canal, waveform, frequency, low, high, phase)`: un valor periódico sobre `"x"`, `"y"`, `"rotation"`, `"scale"` u `"opacity"`, con onda `"sine"`, `"square"`, `"triangle"` o `"saw"`.

```
from gaanim import CORAL, CYAN, GOLD, Scene, Updater

scene = Scene(frame=(16, 9), background="#0f172a")
logo = scene.geometry.star(5, 1.0, 0.45).fill(GOLD).move_to(-4, 0.5)
luz = scene.geometry.circle(0.8).fill(CYAN).move_to(0, 0.5)
boya = scene.geometry.square(1.0).fill(CORAL).move_to(4, 0.5)
logo.add_updater(Updater.wiggle(position=0.2, rotation=0.2, frequency=1.5, seed=1))
luz.add_updater(Updater.oscillate("opacity", frequency=0.8, low=0.3, high=1.0))
boya.add_updater(Updater.oscillate("y", waveform="triangle", frequency=1.0, low=-0.4, high=0.4))
scene.wait(0.5)
scene.play(logo.animate.move_to(-4, -1).duration(1.0))
scene.render()
```

Las amplitudes y la frecuencia de `wiggle`, y la velocidad de `Updater.rotate`, aceptan un `Parameter`. Animarlo cambia el updater mientras corre: `temblor.animate.set(0)` calma un temblor poco a poco en lugar de cortarlo, y una velocidad que sube acelera un giro sin saltos. Un seek cae en el mismo fotograma que la reproducción.

```python
from gaanim import GOLD, Scene, Updater

scene = Scene(frame=(16, 9), background="#0f172a")
circulo = scene.geometry.regular_polygon(6, 2.0).no_fill().stroke(GOLD, 0.04)
velocidad = scene.viz.parameter(0.5)
temblor = scene.viz.parameter(0.0)
circulo.add_updater(Updater.rotate(velocidad))
circulo.add_updater(Updater.wiggle(position=temblor, frequency=4.0, seed=3))
scene.play([velocidad.animate.set(8.0).duration(2), temblor.animate.set(0.12).duration(2)])
scene.play([velocidad.animate.set(0.5).duration(1.5), temblor.animate.set(0.0).duration(1.5)])
scene.render()
```

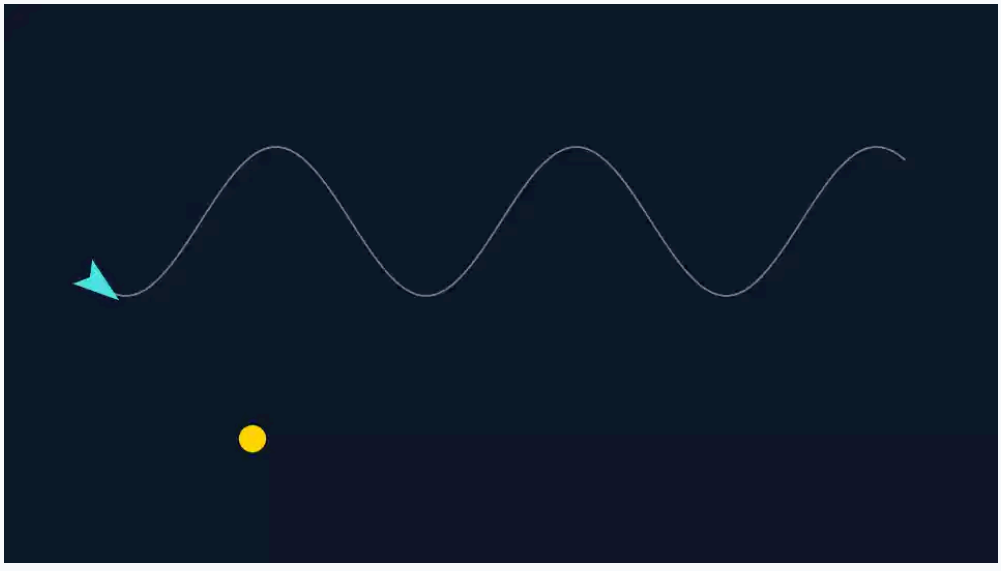
No confundas `Updater.wiggle` con `animate.wiggle()`: el segundo es un énfasis breve para señalar un error, no una capa continua.

14.1.10. Trayectorias y arcos

`animate.move_along(camino)` recorre el contorno de cualquier drawable. Con `orient=True` el objeto gira con la tangente, así un avión apunta hacia donde vuela; `rotate_offset` corrige la orientación de tu dibujo y `start`/`end` eligen el tramo recorrido. Para un recorrido a velocidad constante usa `Easing.LINEAR`.

Un `move_to` en línea recta puede verse mecánico. `.path_arc(ángulo)` lo curva en un arco circular que gira `ángulo` radianes (positivo es antihorario).

```python
import math
from gaanim import CYAN, GOLD, GRAY, Scene

scene = Scene(frame=(16, 9), background="#0f172a")
ruta = scene.geometry.polyline([(x / 10, 1.2 * math.sin(x / 10 * 1.3) + 1) for x in range(-65, 66)])
ruta.no_fill().stroke(GRAY, 0.03)
avion = scene.geometry.polygon([(0.45, 0), (-0.3, 0.25), (-0.15, 0), (-0.3, -0.25)]).fill(CYAN)
avion.move_to(-6.5, 1)
pelota = scene.geometry.dot(0.22).fill(GOLD).move_to(-4, -2.5)
scene.play([
    avion.animate.move_along(ruta, orient=True).duration(2.0),
    pelota.animate.move_to(4, -2.5).path_arc(-math.pi / 2).duration(2.0),
])
scene.render()
```



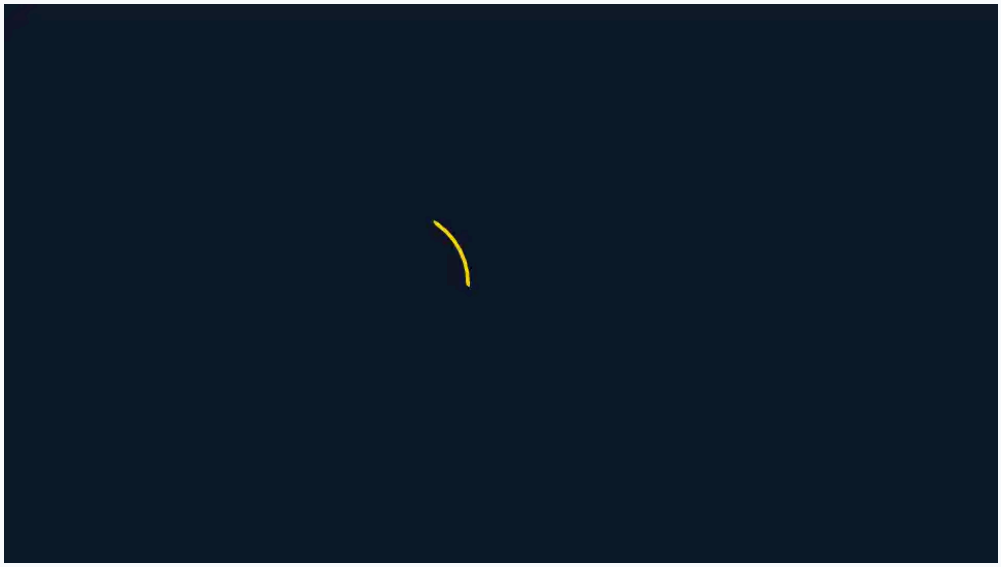
`path_arc` curva el destino de un `move_to` o un `shift_by` de la misma animación; sin una de esas traslaciones lanza `ValueError`.

14.1.11. Trim: dibujar trazos por partes

`drawable.trim(start, end, offset)` muestra solo la ventana `[start, end]` del trazo, en fracciones de longitud. `animate.trim(...)` la anima, y con eso salen los revelados de line art más habituales:

- Dibujar: empieza en `trim(end=0)` y anima a `end=1`.
- Desde el centro: empieza en `trim(start=0.5, end=0.5)` y abre a `0` y `1`.
- Segmento viajero: una ventana corta y un `offset` animado, que da la vuelta en caminos cerrados.
- `mode="sequential"`: los subtrazos aparecen uno tras otro en lugar de a la vez.

```
from gaanim import CORAL, CYAN, GOLD, WHITE, Scene

scene = Scene(frame=(16, 9), background="#0f172a")
anillo = scene.geometry.circle(1.2).no_fill().stroke(CYAN, 0.12).move_to(-5, 0)
anillo.trim(start=0.5, end=0.5)
orbita = scene.geometry.circle(1.2).no_fill().stroke(GOLD, 0.12).move_to(-1.7, 0)
orbita.trim(start=0.0, end=0.15)
guiones = scene.geometry.dashed_line(0.6, 0, 3.4, 0, dash_length=0.3, gap_length=0.2).stroke(CORAL,
0.1)
guiones.trim(end=0.0, mode="sequential")
estrella = scene.geometry.star(5, 1.2, 0.5).no_fill().stroke(WHITE, 0.08).move_to(5.3, 0)
estrella.trim(end=0.0)
scene.play([
    anillo.animate.trim(start=0.0, end=1.0),
    orbita.animate.trim(offset=1.0),
    guiones.animate.trim(end=1.0),
    estrella.animate.trim(end=1.0),
])
scene.wait(0.2)
scene.render()
```



`create`, `write`, `trim` y `show_passing_flash` comparten el mismo canal en un drawable: no los solapes en el mismo `play`; encadénalos con `sequence`.

14.1.12. Repetidores, duplicadores y plexus

Para llenar una escena con muchas copias de una forma no hace falta un bucle de `move_to`: `scene.geometry.repeat(forma, n, ...)` hace `n` copias, cada una un paso más girada, escalada o desplazada (como el Repeater de After Effects), y `scene.geometry.duplicate(forma, Distribution...)` las reparte en una rejilla, un círculo, un camino o una nube aleatoria reproducible. Las dos devuelven un grupo cuyos miembros son las copias, en orden; `animate.count(n)` revela las copias de un repetidor una a una. `scene.geometry.connect(puntos,`

`max_distance)` une con líneas vivas los puntos cercanos (un *plexus*) y las líneas siguen a los puntos mientras se mueven.

```
import math
from gaanim import CYAN, GOLD, PINK, Distribution, Scene

scene = Scene(frame=(16, 9), background="#0f172a")
petalo = scene.geometry.ellipse(0.25, 0.9).fill(PINK).move_to(-4, 1.2)
flor = scene.geometry.repeat(petalo, 12, rotate=math.tau / 12, about=(-4, 0), opacity=(1.0,
0.4)).count(0)
estrella = scene.geometry.star(5, 0.12, 0.05).fill(GOLD)
cielo = scene.geometry.duplicate(estrella, Distribution.random(40, (1, -3, 7, 3), seed=3))
enlaces = scene.geometry.connect(cielo, max_distance=1.6).stroke(CYAN, 0.015)
scene.play([flor.animate.count(12).duration(1.0)])
scene.play([enlaces.animate.fade_in().duration(0.6)])
scene.render()
```

El trazo también se anima sin cambiar la forma: `stroke_profile` da un grosor variable a lo largo del camino y `Updater.dash_flow(speed)` hace correr los guiones de un trazo discontinuo (ver Drawable).

14.1.13. Referencia

- Animaciones: `Easing`, `stagger`, `distribute`, `repeat`, `loop`, `label`, `move_along`, `path_arc` y `trim`.
- Scene: `scene.random`, `scene.noise`, `Updater.wiggle`, `Updater.oscillate` y `scene.marker`.
- Geometría: los drawables que sirven de trayectoria.

CAPÍTULO

15. Personajes con SVG por partes

Dibuja un personaje en Inkscape con un id por parte y anímalo con part, with_pivot y look_at

15.1. Dibujar en un editor, animar por partes

Un personaje ilustrado se dibuja mejor en un editor vectorial que con curvas escritas a mano. En esta guía lo dibujas en Inkscape (vale cualquier editor que guarde SVG), le das un `id` a cada parte que quieras mover y lo animas en Gaanim: inclinas la cabeza desde el cuello, levantas las cejas, saludas con un brazo y haces que el otro apunte a un objeto que se mueve.

15.1.1. Prepara el SVG

1. Dibuja cada parte móvil por separado: cabeza, cejas, brazos, piernas. Agrupa (`Ctrl+G`) lo que se mueve junto, como la cara con sus ojos.
2. Dale a cada parte un `id` en **Objeto › Propiedades del objeto** (`Ctrl+Shift+O`): escribe el nombre en **ID** y pulsa **Establecer**. Usa nombres cortos sin espacios (`cabeza`, `brazo-der`); distinguen mayúsculas.
3. Dibuja las extremidades en reposo y extendidas a lo largo del eje x, desde la articulación hacia fuera. Así un giro de 0 es la pose de reposo y `look_at` apunta la extremidad hacia su objetivo.
4. Si una parte tiene que girar entera hacia un objetivo con `look_at`, haz que sea un solo trazado (**Trazo › Unión**): en un grupo, `look_at` gira cada miembro por separado.
5. Guarda como **SVG plano**. Anota en píxeles los puntos de giro: el cuello, los hombros, las caderas.

El personaje de esta guía (`assets/personaje.svg`, 400 × 300 px) tiene las partes `cabeza`, `cejas`, `ojos`, `nariz`, `boca`, `torso`, `piernas`, `brazo-izq` y `brazo-der`; cada brazo es un solo trazado con su mano.

15.1.2. De píxeles del SVG a la escena

Gaanim importa el SVG a 100 píxeles por unidad, con el centro del documento en el origen y la y hacia arriba. Un punto `(px, py)` de un documento de `ancho × alto` píxeles, escalado por `ESCALA`, queda en

```
((px - ancho / 2) / 100 * ESCALA, (alto / 2 - py) / 100 * ESCALA)
```

Usa `with_pivot(0, 0)` antes de `scale_to`, para que el SVG escale desde el centro del documento y la fórmula siga valiendo. Con ella fijas el pivote de cada parte en su articulación.

15.1.3. Animar por partes

`part(id)` devuelve cada parte como un `Drawable` propio: se anima, se colorea y se mueve sin mover el resto. `with_pivot` fija el punto alrededor del que gira:

```
from gaanim import GOLD, Scene

scene = Scene(frame=(16, 9), background="#0f172a")
ESCALA = 2.0
persona = scene.media.svg("assets/personaje.svg").with_pivot(0, 0).scale_to(ESCALA)


def punto(px, py):
```

```
    """Un punto del SVG, en píxeles, en coordenadas de la escena."""
    return ((px - 200) / 100 * ESCALA, (150 - py) / 100 * ESCALA)


cabeza = persona.part("cabeza").with_pivot(*punto(200, 140))  # el cuello
cejas = persona.part("cejas")
brazo_izq = persona.part("brazo-izq").with_pivot(*punto(164, 159))  # hombro izquierdo
brazo_der = persona.part("brazo-der").with_pivot(*punto(236, 159))  # hombro derecho

scene.play([persona.animate.fade_in().duration(0.4)])
# Inclina la cabeza, levanta las cejas y saluda.
scene.play([
    cabeza.animate.rotate_by(0.15).duration(0.4),
    cejas.animate.shift_by(0, 0.08).duration(0.4),
    brazo_izq.animate.rotate_by(-0.8).duration(0.4),
])
scene.play([
    cabeza.animate.rotate_by(-0.15).duration(0.4),
    cejas.animate.shift_by(0, -0.08).duration(0.4),
    brazo_izq.animate.rotate_by(0.8).duration(0.4),
])
# El brazo derecho sigue a la estrella mientras se mueve.
estrella = scene.geometry.star(5, 0.3, 0.12).fill(GOLD).move_to(5, 3)
brazo_der.look_at(estrella)
scene.play([estrella.animate.move_to(5, -2.5).duration(1.2)])
scene.render()
```

`look_at` sigue al objetivo desde el cursor hasta `clear_drive()`; con `offset=` (radianes) sirve también para partes dibujadas apuntando a otro lado, como un brazo izquierdo extendido hacia la izquierda (`offset=math.pi`).

15.1.4. Trazos que crecen con la cara

Las cejas, la nariz y la boca de este personaje son trazos. Un trazo mide su ancho en unidades de escena, así que al agrandar la cabeza quedarían más finos que el resto.
`scale_stroke_with_object()` hace que crezcan con ella:

```
cabeza = persona.part("cabeza").with_pivot(0, 0.2).scale_stroke_with_object()
scene.play([cabeza.animate.scale_to(1.18).duration(0.5)])
```

15.1.5. Referencia

- Medios: `scene.media.svg`, `part` y `parts`.
- Drawable: `with_pivot`, `look_at`, `scale_stroke_with_object` y las animaciones de `.animate`.

- Movimiento: trayectorias, updaters y temblores para dar vida a un personaje quieto.
- Layout: `scene.layout.scatter` reparte fórmulas o etiquetas alrededor del personaje sin tapar sus partes (`avoid=[persona.part("cabeza"), ...]`).

CAPÍTULO

16. Tipografía cinética

Escritura por unidades, animador de rango, revelados con máscara, máquina de escribir, scramble y resaltado

16.1. Texto que se mueve con intención

En esta guía animas texto por grafemas, palabras o líneas: revelados editoriales detrás de una máscara, olas que recorren un titular, una terminal que escribe sola, un código que se descifra y un resaltador que subraya una idea.

Todas estas animaciones trabajan sobre el `Text` ya compuesto: el párrafo no se vuelve a maquetar en cada fotograma y un seek a cualquier instante coincide con la reproducción.

```
from gaanim import WHITE, Scene

scene = Scene(frame=(16, 9), background="#0f172a")
titular = scene.text("Cada línea sube\ndesde su máscara", role="title").fill(WHITE).scale_by(1.6)
scene.play(titular.animate.reveal(by="line", style="slide_up", stagger=0.15).duration(1.0))
scene.wait(0.3)
scene.render()
```

16.1.1. Las unidades: grafema, palabra, línea y parte

Todas las técnicas de esta guía eligen una unidad con `by`:

- `"grapheme"`: cada carácter visible, incluidos los compuestos como «é».
- `"word"`: palabras según los límites de Unicode; la puntuación se une a su vecina.
- `"line"`: líneas explícitas, separadas con `\n` en el texto.
- `"part"`: las partes semánticas que declaras con `part(...)`.

Son las mismas unidades que `text.graphemes`, `text.words`, `text.lines` y `text.parts`, así que lo que seleccionas y lo que animas coincide siempre.

16.1.2. Escribir por unidades

`text.animate.write()` traza los glifos uno tras otro. Con `by` los glifos de una misma unidad empiezan juntos, con `order` eliges qué unidad va primero (`"forward"`, `"reverse"`, `"center"` o `"random"`) y `stagger` es el desfase entre unidades consecutivas.

```
from gaanim import GOLD, WHITE, Scene, part

scene = Scene(frame=(16, 9), background="#0f172a")
frase = scene.text("Escribir desde el centro hacia fuera").fill(WHITE).move_to(0, 1.5)
ecuacion = scene.text("$", part("e", "E"), " = ", part("m", "m"), " ", part("c", "c^2"),
"$").fill(GOLD)
ecuacion.move_to(0, -1)
scene.play(frase.animate.write(by="word", order="center", stagger=0.08).duration(1.2))
scene.play(ecuacion.animate.write(by="part", stagger=0.1).duration(1.0))
scene.render()
```

`write` es la opción natural para ecuaciones y trazos de pluma. Para texto de lectura, un revelado suele ser más limpio.

16.1.3. Revelados con máscara

`text.animate.reveal(style, by="line", stagger=0.06)` es el movimiento editorial por excelencia: cada unidad entra escalonada. Los estilos:

Estilo	Cuándo usarlo
`"slide_up"`	Titulares y citas: cada unidad sube desde detrás de una máscara recortada a su fila. Es el predeterminado.
`"slide_down"`	Lo mismo, cayendo desde arriba; combina con entradas desde el borde superior.
`"fade"`	Texto largo o secundario, cuando el movimiento distraería.
`"scale"`	Palabras sueltas o etiquetas que «aparecen» con energía.
`"blur"`	Un tono suave y cinematográfico.

La máscara es vectorial, así que el recorte funciona también al exportar a SVG. Con `mask=False` los deslizamientos recorren menos distancia y entran con un fundido. La duración cubre toda la cascada: si el `stagger` no cabe, se comprime. `conceal` es la salida simétrica.

```
from gaanim import GOLD, WHITE, Scene

scene = Scene(frame=(16, 9), background="#0f172a")
lineas = scene.text("Tres líneas\nque suben\ny se van").fill(WHITE).move_to(-4, 0)
palabras = scene.text("palabra por palabra").fill(GOLD).move_to(3.5, 1)
suave = scene.text("con desenfoque").fill(WHITE).move_to(3.5, -1)
scene.play([
    lineas.animate.reveal(by="line", style="slide_up", stagger=0.15).duration(1.2),
    palabras.animate.reveal(by="word", style="slide_down", stagger=0.12).duration(1.2),
    suave.animate.reveal(by="word", style="blur", stagger=0.1).duration(1.2),
])
scene.wait(0.4)
scene.play(lineas.animate.conceal(by="line", style="slide_up", stagger=0.15).duration(1.0))
scene.render()
```

Sobre una selección (`ecuacion["rhs"].animate.reveal("from_below")`), `reveal` hace aparecer solo esos glifos con `"fade"`, `"wipe"` o `"from_below"`; lo verás en la sección «Marcar y anotar», más abajo.

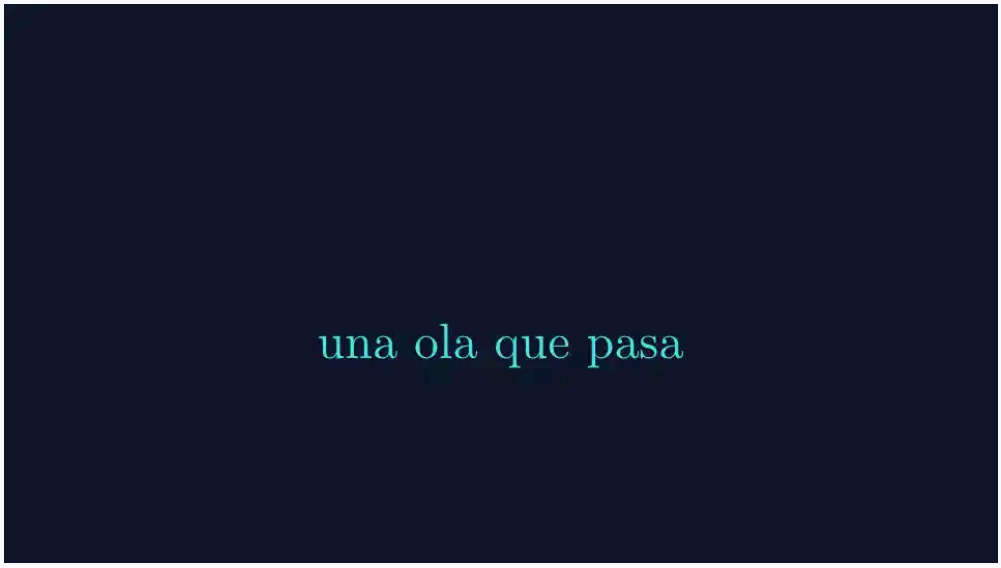
16.1.4. El animador de rango

`reveal`, `conceal` y `blur_in` son presets de una primitiva más general: el animador de rango, al estilo de los Text Animators de After Effects.

1. `text.animator(by, shape, order, seed)` crea el selector.
2. `.set(...)` define el estado «fuera»: `offset`, `opacity`, `scale`, `rotation`, `blur`, `tracking` y `color`.
3. `.animate.sweep()` recorre el rango de `0` (antes de la primera unidad) a `1` (después de la última).

`shape` decide qué pasa en cada unidad. `"square"`, `"ramp"`, `"smooth"`, `"ease_in"` y `"ease_out"` llevan cada unidad del estado «fuera» al reposo: son entradas. `"triangle"` y `"round"` suben al estado «fuera» y vuelven: son olas que pasan.

```
from gaanim import CYAN, GOLD, WHITE, Scene

scene = Scene(frame=(16, 9), background="#0f172a")
cascada = scene.text("Tipografía cinética", role="title").fill(WHITE).scale_by(1.6).move_to(0, 1.2)
entrada = cascada.animator(by="grapheme", shape="smooth")
entrada.set(offset=(0, -0.5), opacity=0.0, scale=0.6, rotation=0.3)
ola_texto = scene.text("una ola que pasa").fill(CYAN).scale_by(2.0).move_to(0, -1.2)
ola = ola_texto.animator(by="grapheme", shape="round", order="center").set(offset=(0, 0.35),
color=GOLD)
scene.play([
    entrada.animate.sweep().duration(1.2),
    ola.animate.sweep().duration(1.2),
])
scene.wait(0.2)
scene.render()
```



`sweep(1, 0)` recorre el rango al revés y `stagger` fija el retraso entre unidades en segundos. Usa el animador cuando ningún preset describe lo que buscas; si uno lo hace, el preset es más corto y más fácil de leer.

16.1.5. Máquina de escribir

`text.animate.typewriter(cps=18, cursor="▌")` escribe el texto a un ritmo de pulsaciones por segundo, con una pequeña irregularidad (`jitter`) fijada por `seed` para que parezca humano. El cursor sigue la última letra y parpadea cuando termina (`blink`). `backspace(n)` borra desde el final y `retype(texto)` conserva el prefijo común y escribe el resto.

```
from gaanim import CYAN, Scene
```

```
scene = Scene(frame=(16, 9), background="#0f172a")
prompt = scene.text("> gaanim render", role="code").fill(CYAN).scale_by(2).move_to(-3, 0)
scene.play(prompt.animate.typewriter(cps=18, cursor="▌", seed=7))
scene.wait(0.5)
scene.render()
```

Sin duración explícita, cada animación dura hasta su última pulsación. Usa el rol `code`: su fuente monoespaciada mantiene alineados el cursor y los caracteres.

```
scene.play(prompt.animate.backspace(6))
scene.play(prompt.animate.retype("> gaanim export --from clímax"))
scene.render()
```

16.1.6. Scramble: descifrar un texto

`text.animate.scramble(charset, reveal_delay, speed, seed)` hace ciclar cada carácter por glifos al azar antes de fijarlo, de izquierda a derecha. Da un tono técnico a títulos, códigos y rótulos. `charset` acepta `"upper"`, `"lower"`, `"digits"`, `"hex"`, `"symbols"` o una cadena literal como `"01"`. `scramble_to(texto)` descifra hacia un texto nuevo.

```
from gaanim import GOLD, Scene

scene = Scene(frame=(16, 9), background="#0f172a")
rotulo = scene.text("ACCESO CONCEDIDO", role="title").fill(GOLD).scale_by(1.6)
scene.play(rotulo.animate.scramble(charset="upper", reveal_delay=0.3, speed=20, seed=1).duration(1.2))
scene.wait(0.3)
scene.render()
```

El ancho del texto final se reserva desde el primer fotograma, así que nada salta mientras los caracteres cambian.

16.1.7. Blur-in y tracking

`blur_in(sigma, by, stagger)` hace entrar cada unidad desde un desenfoque transparente. `tracking(valor)` añade espacio entre glifos sin recomponer el párrafo; animarlo hasta `0` «cierra» un título abierto, un recurso clásico de créditos y cabeceras.

```
from gaanim import WHITE, Scene

scene = Scene(frame=(16, 9), background="#0f172a")
enfoque = scene.text("ENFOQUE", role="title").fill(WHITE)
enfoque.tracking(0.5)
scene.play([
    enfoque.animate.blur_in(sigma=0.3, by="grapheme", stagger=0.05).duration(1.4),
    enfoque.animate.tracking(0.0).duration(1.4),
])
scene.render()
```

`tracking` y un `reveal` que desliza escriben el mismo canal de cada glifo: `scene.play` rechaza esa combinación. Combina `tracking` con animaciones que no mueven glifos, como `blur_in`.

16.1.8. Marcar y anotar

Para señalar una idea dentro de un texto que ya está en pantalla, anima una selección:

- `marker(color, skew, opacity)`: una banda de resaltador que crece desde la izquierda detrás de cada línea seleccionada. `selection.marker(...)` la coloca sin animación.
- `reveal("from_below")`: hace aparecer solo un término de una ecuación.
- `brace("etiqueta")` y `annotate("texto", offset=(x, y))`: una llave o una línea guía con etiqueta, que se quedan en la escena.

```
from gaanim import CORAL, GOLD, GREEN, Scene, part

INK = "#1f2937"
scene = Scene(frame=(16, 9), background="#f8f5ee")
cita = scene.text("Lo que no se mide no se puede mejorar", color=INK, size=0.6).move_to(0, 2)
scene.play(cita.words[5:8].animate.marker().duration(1.0))
scene.play(cita.words[0:2].animate.marker(GREEN, skew=-0.04, opacity=0.4).duration(0.6))
```

```
ecuacion = scene.text("$", part("e", "E"), " = ", part("m", "m"), " ", part("c", "c^2"), "$").fill(INK)
ecuacion.move_to(0, -1)
ecuacion["m"].fill(GOLD)
ecuacion["c"].fill(CORAL)
scene.play(ecuacion["c"].animate.reveal("from_below").duration(0.6))
scene.play(ecuacion["m"].animate.brace("masa").duration(0.7))
scene.play(ecuacion["c"].animate.annotate("velocidad de la luz", offset=(2.5, 1.0)).duration(0.7))
scene.render()
```

Las bandas se dibujan detrás de los glifos pero encima de lo que creaste antes que el texto. Con `blend="multiply"` la banda se multiplica con el fondo o la tarjeta que tiene debajo, como tinta sobre papel: realza sobre fondos claros y desaparece sobre negro.

16.1.9. Referencia

- Texto: unidades y selecciones, `write`, `reveal`, `conceal`, `blur_in`, `tracking`, `Text.animator` y el resaltador.
- Animaciones: `typewriter`, `backspace`, `retype`, `scramble` y `scramble_to`.

CAPÍTULO

17. Transiciones

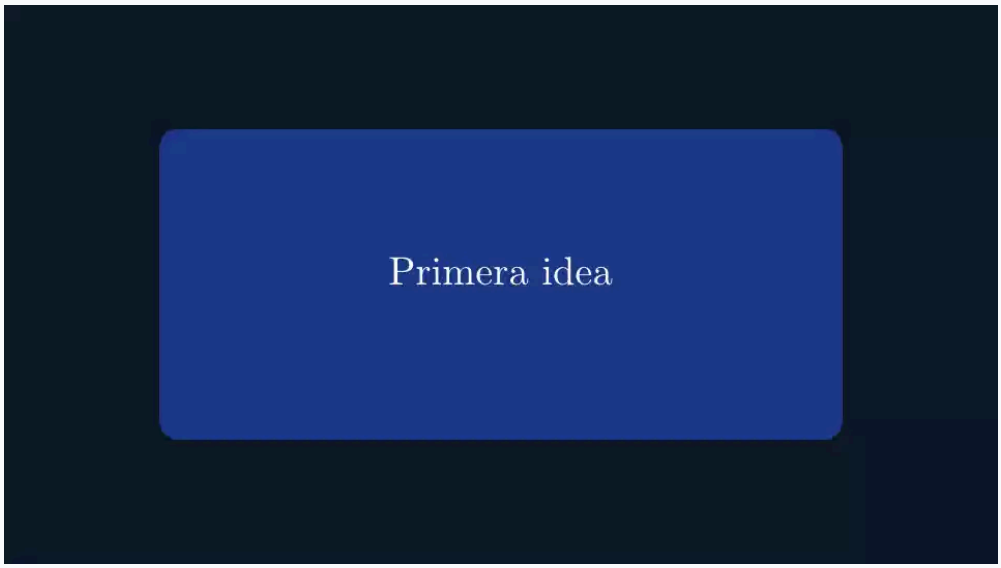
Wipes, iris, push, blinds y morph entre segmentos, overlays sobre el corte y cámara con impacto

17.1. Pasar de una idea a la siguiente

En esta guía unes segmentos con transiciones vectoriales (barridos, iris, persianas, empujes), añades un destello o una fuga de luz sobre el corte, conviertes un objeto en otro a través del corte con `Transition.morph` y das impacto a la cámara con una sacudida por trauma y un zoom que se percibe uniforme.

Una transición pertenece al segmento que **entra**: se declara en `scene.segment(nombre, transición)` y ocupa el principio de ese segmento, sin alargar la escena.

```python
from gaanim import WHITE, Scene, Transition

scene = Scene(frame=(16, 9), background="#0f172a")

def tarjeta(titulo, color):
    scene.geometry.rounded_rect(11, 5, 0.3).fill(color)
    scene.text(titulo, role="title").fill(WHITE)

scene.segment("intro", background="#0f172a")
tarjeta("Primera idea", "#1e3a8a")
scene.wait(0.6)
scene.segment("detalle", Transition.wipe(0.6, direction="left", feather=0.15), background="#3b0764")
tarjeta("Segunda idea", "#7e22ce")
scene.wait(1.0)
scene.render()
```



17.1.1. Elegir una transición

Transición	Cuándo usarla

`Transition.cut()`	Cambio de ritmo seco; con un overlay, un impacto.
`cross_fade(d)`	Continuidad tranquila entre ideas relacionadas.
`fade_through(d, color)`	Cierre de capítulo: la pantalla pasa por un color.
`wipe(d, direction, feather)`	Avanzar en una lista o una línea de tiempo; la dirección sugiere «siguiente».
`clock_wipe(d, start_angle)`	Paso del tiempo, cuentas atrás.
`iris(d, center, shape)`	Entrar en un detalle desde un punto concreto del cuadro.
`blinds(d, count, angle)`	Cambio de sección con textura gráfica.
`push(d, direction)`	Pasar de página: el segmento nuevo empuja al anterior.
`slide(d, direction)`	El segmento nuevo se desliza encima del anterior, que queda quieto.
`zoom_through(d)`	Atravesar una idea para llegar a la siguiente.
`morph(d, pairs=...)`	Un objeto continúa a través del corte y cambia de forma, tamaño o color.
`preset(nombre, d)`	Efectos de vídeo con shader: `cross_zoom`, `directional_warp`, `ripple`, `glitch_displace` y `luma`.
`shader(wgsl, d)`	Tu propia transición en WGSL, como en gl-transitions.

Los revelados vectoriales (`wipe`, `clock_wipe`, `iris`, `blinds`, `push` y `slide`) recortan los dos segmentos con geometría animada: son nítidos a cualquier resolución y funcionan sin texturas. Si los segmentos tienen fondos distintos, el fondo entrante se revela con la misma forma. Los objetos que marcaste con `scene.persist(...)` no se recortan ni se desplazan. Un objeto nuevo que persistes justo después de `scene.segment(...)` entra con ese segmento y su transición lo recorta; créalo y persístelo antes de `scene.segment(...)` para que quede fuera.

```python
from gaanim import WHITE, Scene, Transition

scene = Scene(frame=(16, 9), background="#0f172a")

def tarjeta(titulo):
    scene.text(titulo, role="title").fill(WHITE)
    scene.wait(1.0)

scene.segment("intro", background="#1e3a8a")
tarjeta("Inicio")
scene.segment("reloj", Transition.clock_wipe(0.8, start_angle=90), background="#052e16")
tarjeta("clock_wipe")
scene.segment("iris", Transition.iris(0.7, center=(2, 1), shape="star"), background="#431407")
tarjeta("iris")
scene.segment("persiana", Transition.blinds(0.6, count=8, angle=0), background="#083344")
tarjeta("blinds")
scene.segment("empuje", Transition.push(0.5, direction="up"), background="#422006")
tarjeta("push")
scene.render()
```

`iris` también acepta un `Drawable` como `shape`: su contorno sirve de plantilla. Ese drawable se sigue dibujando en su propio segmento, así que ocúltalo si solo lo usas como forma.

17.1.2. Easing en las transiciones

Todas las transiciones salvo `cut` aceptan `easing=` con cualquier `Easing`, incluidos los springs, que pueden sobrepasar el final y volver. Sin él, los revelados vectoriales usan `Easing.SMOOTH` y `cross_fade`, `fade_through`, `zoom_through` y `morph` avanzan de forma lineal.

```
from gaanim import WHITE, Easing, Scene, Transition

scene = Scene(frame=(16, 9), background="#0f172a")
scene.segment("a", background="#1e3a8a")
scene.text("Antes", role="title").fill(WHITE)
scene.wait(1.0)
scene.segment("b", Transition.slide(0.6, "left", easing=Easing.spring(bounce=0.2)),
background="#7e22ce")
scene.text("Después", role="title").fill(WHITE)
scene.wait(1.0)
scene.render()
```

Un spring con poco `bounce` hace que el segmento nuevo «encaje» en su sitio; úsalo con `slide` y `push`, donde el sobrepaso se lee como inercia.

17.1.3. Overlays sobre el corte

`overlay=` dibuja algo encima de cualquier transición, centrado en su punto medio (en `cut`, en el propio corte), sin cambiar la duración de los segmentos.

- `Overlay.flash(color, duration)`: un destello que sube y se apaga. Da impacto a un corte seco.
- `Overlay.light_leak(seed, hue, duration, intensity)`: manchas de luz suaves que derivan por el cuadro. `hue` es el tono en vueltas: `0.1` es un naranja cálido y `0.6` un azul.

```
from gaanim import WHITE, Overlay, Scene, Transition

scene = Scene(frame=(16, 9), background="#0f172a")
scene.segment("intro", background="#1e3a8a")
scene.text("Intro", role="title").fill(WHITE)
scene.wait(1.0)
scene.segment("impacto", Transition.cut(overlay=Overlay.flash(WHITE, 0.15)), background="#3b0764")
scene.text("¡Impacto!", role="title").fill(WHITE)
scene.wait(1.0)
scene.segment("cálido", Transition.cross_fade(0.6, overlay=Overlay.light_leak(seed=2, hue=0.1)),
background="#052e16")
scene.text("Recuerdo", role="title").fill(WHITE)
scene.wait(1.0)
scene.render()
```

Los overlays son funciones puras del tiempo y de la semilla: la vista previa, un seek y la exportación muestran el mismo fotograma.

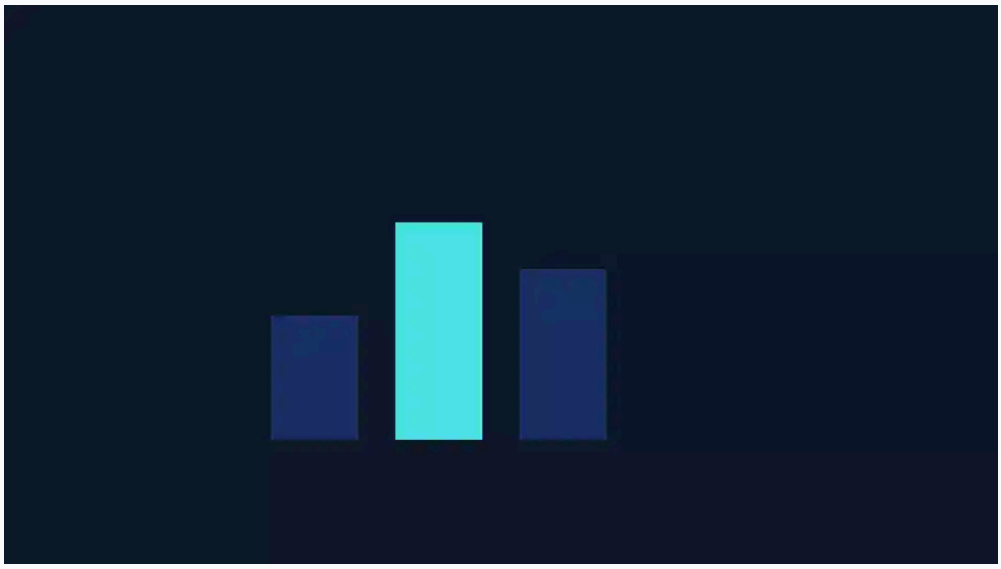
17.1.4. Morph: un objeto que cruza el corte

`Transition.morph(d, pairs=[(origen, destino)])` empareja un drawable del segmento saliente con uno del entrante. La pareja comparte una caja que viaja de la posición y el tamaño del origen a los del destino, mientras uno se desvanece en el otro, así que se lee como un solo objeto que cambia. Lo que no está emparejado se funde.

Las parejas se resuelven al compilar la escena, cuando ya existen los dos segmentos. Por eso morph se declara con `scene.link(saliente, entrante, transición)` después de crear ambos.

```
from gaanim import CYAN, BLUE, WHITE, Scene, Transition

scene = Scene(frame=(16, 9), background="#0f172a")
```

```
barras_seg = scene.segment("barras")
barras = [scene.geometry.rect(1.4, h).fill(c).move_to(x, h / 2 - 2.5) for x, h, c in [(-3, 2, BLUE),
(-1, 3.5, CYAN), (1, 2.75, BLUE)]]
scene.wait(0.6)
detalle_seg = scene.segment("detalle")
panel = scene.geometry.rounded_rect(9, 4.5, 0.3).fill("#16213a").move_to(0, 0)
scene.text("Detalle de la barra", role="heading").fill(WHITE).move_to(0, 1.2)
scene.wait(1.2)
scene.link(barras_seg, detalle_seg, Transition.morph(0.8, pairs=[(barras[1], panel)]))
scene.render()
```



Morph encaja cuando el espectador debe seguir **un** elemento: la barra que se convierte en su ficha, el icono que se convierte en el título de la sección.

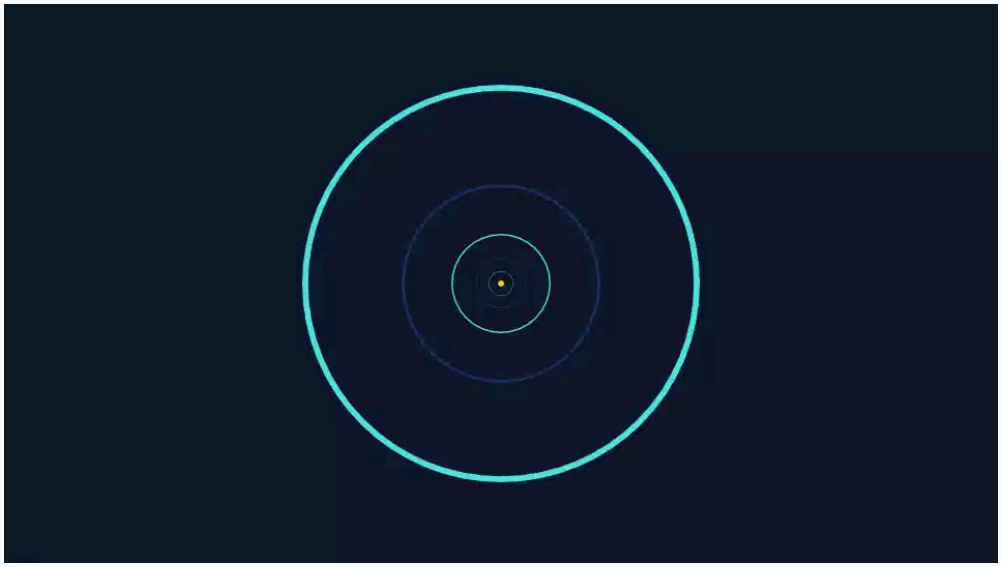
17.1.5. Cámara con impacto

La cámara también marca transiciones dentro de un segmento.

`scene.camera.animate.shake(trauma=0.8, decay=1.5, ...)` sigue el modelo de trauma: la sacudida es proporcional a `trauma²`, así que un golpe suave apenas se nota y uno fuerte sacude de verdad, y el trauma decae `decay` por segundo. El movimiento sale de ruido coherente con semilla (`seed`), `frequency` fija su rapidez y `rotation` el giro máximo.

`scene.camera.animate.zoom_to(z)` interpola el zoom de forma exponencial: el área visible cambia en la misma proporción cada fotograma, así que un zoom de ×8 se percibe a velocidad constante en lugar de acelerar al final. `interpolation="linear"` recupera la interpolación lineal. `frame_to` usa el mismo modelo.

```
from gaanim import BLUE, CYAN, GOLD, Scene

scene = Scene(frame=(16, 9), background="#0f172a")
for i, r in enumerate([3.2, 1.6, 0.8, 0.4, 0.2, 0.1]):
    scene.geometry.circle(r).no_fill().stroke([CYAN, BLUE][i % 2], r * 0.06)
scene.geometry.dot(0.05).fill(GOLD)
scene.play(scene.camera.animate.zoom_to(6.0).duration(1.0))
scene.play(scene.camera.animate.shake(trauma=0.9, decay=1.8, rotation=0.03, seed=2))
scene.render()
```



La duración por defecto de `shake` es `trauma / decay` segundos. Un `shake(0.12, 8)` con solo amplitud y frecuencia conserva la sacudida sinusoidal de versiones anteriores.

17.1.6. Transiciones por shader

`Transition.preset(nombre, d)` y `Transition.shader(wgsl, d)` dibujan cada segmento por separado, cada uno con su fondo, y los mezclan en la GPU con una función WGSL. Así se consiguen efectos que la geometría no puede hacer: un zoom con desenfoque, ondas que deforman la imagen o un glitch. Los objetos de `scene.persist(...)` y los overlays quedan nítidos por encima de la mezcla. Un seek o una exportación desde la mitad dan el mismo fotograma que la reproducción.

```
from gaanim import GOLD, WHITE, Scene, Transition

scene = Scene(frame=(16, 9), background="#0f172a")

def tarjeta(titulo, color):
    scene.text(titulo, role="title").fill(color)
    scene.wait(1.0)

scene.segment("intro", background="#1e3a8a")
tarjeta("Inicio", WHITE)
scene.segment("zoom", Transition.preset("cross_zoom", 0.8, strength=0.6), background="#052e16")
tarjeta("cross_zoom", GOLD)
scene.segment("ondas", Transition.preset("ripple", 1.0), background="#431407")
tarjeta("ripple", WHITE)
scene.segment("glitch", Transition.preset("glitch_displace", 0.6, strength=0.8), background="#083344")
tarjeta("glitch_displace", GOLD)
scene.render()
```

`luma` revela el segmento entrante siguiendo una imagen en escala de grises: primero donde es más oscura (o más clara, con `invert=True`). Con un degradado, unas nubes o una mancha de tinta se obtienen revelados orgánicos; `softness` suaviza el borde.

```
scene.segment("despues", Transition.preset("luma", 1.2, image="assets/tinta.png", softness=0.15))
scene.wait(1)
```

Para escribir la tuya, define `fn transition(uv: vec2<f32>) -> vec4<f32>`. `gaanim_from(uv)` lee el segmento saliente, `gaanim_to(uv)` el entrante y `progress` va de 0 a 1 (lineal salvo que pases

`easing=`). `uv` vale (0, 0) en la esquina superior izquierda del cuadro, `gaanim_resolution()` es su tamaño en píxeles y los `uniforms` son campos de `gaanim_uniforms`:

```
ondas = Transition.shader(
    """
    fn transition(uv: vec2<f32>) -> vec4<f32> {
        let shift = sin(uv.y * 30.0) * gaanim_uniforms.amount * sin(progress * 3.14159);
        let q = uv + vec2<f32>(shift, 0.0);
        return mix(gaanim_from(q), gaanim_to(q), progress);
    }
    """,
    0.8,
    uniforms={"amount": 0.03},
)
scene.segment("despues", ondas)
scene.wait(1)
```

Los paquetes `.gaanim` guardan la transición y la reproducen igual, también en el reproductor web.

17.1.7. Referencia

- Animaciones: `Transition` con todas sus fábricas y `Overlay`.
- Scene: `scene.segment`, `scene.link`, `scene.persist` y la cámara (`zoom_to`, `frame_to`, `shake`).
- Presentaciones: segmentos, paradas y navegación.

CAPÍTULO

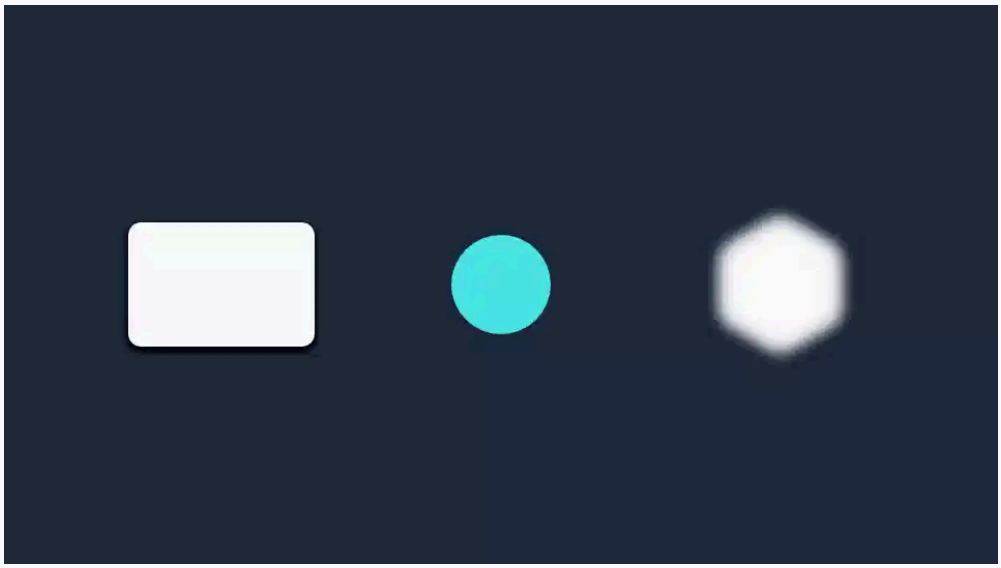
18. Efectos

Glow, blur y sombra animables, postprocesado encadenado, acabados, desenfoque de movimiento y fondos vivos

18.1. Acabado visual

En esta guía añades profundidad y brillo a los objetos con `glow`, `blur` y `shadow`, los animas para levantar una tarjeta, hacer latir una luz o enfocar un objeto, aplicas postprocesados (propios o listos, encadenados y animados) a todo el cuadro, y das movimiento con desenfoque, ecos y fondos vivos.

```
from gaanim import BLACK, CYAN, WHITE, Scene

scene = Scene(frame=(16, 9), background="#1e293b")
tarjeta = scene.geometry.rounded_rect(3, 2, 0.2).fill("#f8fafc").move_to(-4.5, 0)
tarjeta.shadow(BLACK, 0, -0.05, 0.05)
orbe = scene.geometry.circle(0.8).fill(CYAN)
hexagono = scene.geometry.regular_polygon(6, 1.0).fill(WHITE).move_to(4.5, 0)
hexagono.blur(0.15)
scene.play([
    tarjeta.animate.shadow(BLACK, 0, -0.3, 0.35).scale_to(1.06).duration(0.8),
    orbe.animate.glow(CYAN, radius=0.6, intensity=2.0).duration(0.4).repeat(2, yoyo=True),
    hexagono.animate.blur(0.0).duration(0.8),
])
scene.wait(0.3)
scene.render()
```



La tarjeta se levanta, la luz late y el hexágono se enfoca. Los tres efectos usan unidades de escena, así que se ven igual a cualquier resolución.

18.1.1. Glow, blur y sombra

Los tres son setters fluidos de cualquier drawable, texto incluido:

- `glow(color, radius=0.16, intensity=1.0)`: un halo del color indicado. Sirve para destacar un elemento activo o dar aspecto de luz.
- `blur(sigma=0.04)`: desenfoque gaussiano. Aleja un fondo o suaviza una forma decorativa. `blur(0)` lo quita, y sirve de punto de partida nítido para `animate.blur`.
- `shadow(color, x=0.08, y=-0.08, blur=0.06)`: sombra desplazada y desenfocada. El alfa del color regula su opacidad; una sombra sutil separa una tarjeta del fondo.
- `no_effects()` quita los tres sin tocar relleno ni trazo.

Como `fill`, después del primer `play` o `wait` estos setters son cortes en el cursor: el efecto cambia en ese instante y lo anterior se ve como estaba.

- `blend(mode)` cambia cómo se funde el objeto con lo que hay debajo: `"screen"` o `"add"` para luces y destellos, `"multiply"` para resaltadores. No es un efecto, así que `no_effects()` no lo quita.

```
from gaanim import Scene

scene = Scene(frame=(16, 9), background="#0f172a")
titulo = scene.text("Título", role="title").fill("#e2e8f0").move_to(0, 2.5).glow("#38BDF8")
mancha = scene.geometry.circle(2).fill("#1E3A8A").move_to(-4, -1).blur(0.15)
tarjeta = scene.geometry.rounded_rect(4, 2, 0.2).fill("#18202E").move_to(2, -1)
tarjeta.shadow("#00000080", x=0.12, y=-0.12, blur=0.1)
scene.render()
```

Funcionan sobre rellenos y trazos, también con pinceles de gradiente, y los efectos que no cambian se reutilizan desde la caché del renderizador. En un `Text` conservan el handle especializado, así que puedes seguir encadenando métodos de texto.

18.1.2. Animar efectos

`animate.glow(...)`, `animate.blur(sigma)` y `animate.shadow(...)` interpolan desde el estado actual hasta el indicado, y se combinan con otros destinos de la misma animación, como `scale_to`. Un efecto que no existía crece desde cero; `glow(None)`, `blur(0)` y `shadow(None)` lo desvanecen.

Tres patrones cubren la mayoría de usos:

- **Levantar**: una sombra más lejana y difusa junto con un leve `scale_to` hacen que una tarjeta «se despegue» del fondo al seleccionarla.
- **Latir**: `glow(...)` con `.repeat(n, yoyo=True)` hace pulsar una luz y vuelve al estado inicial con un número par de ciclos.
- **Enfocar**: parte de `blur(0.3)` y anima a `blur(0.0)`. Para texto, usa `text.animate.blur_in(...)`, que además escalona las unidades.

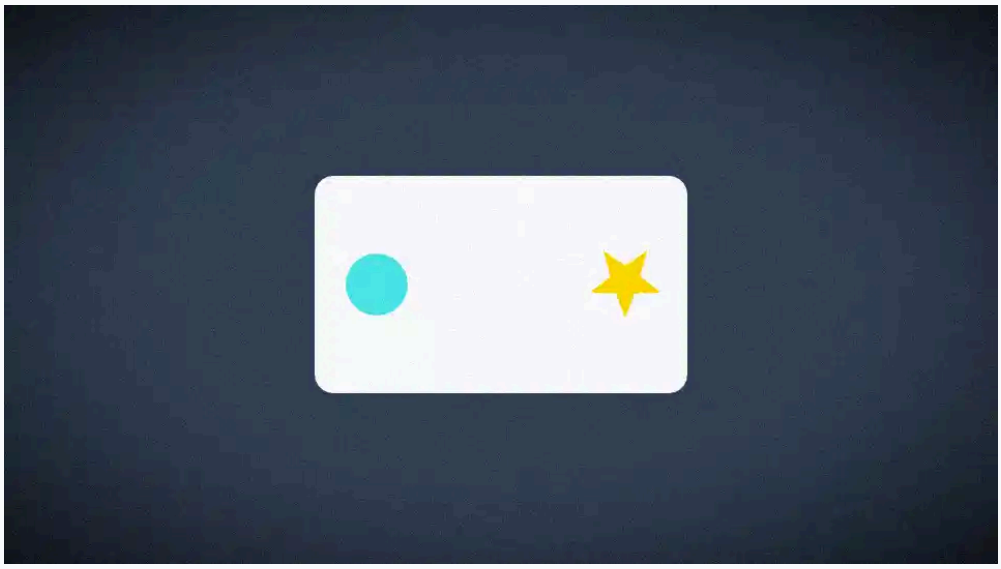
```
from gaanim import BLACK, GOLD, Scene

scene = Scene(frame=(16, 9), background="#1e293b")
tarjeta = scene.geometry.rounded_rect(4, 2.5, 0.2).fill("#f8fafc")
tarjeta.shadow(BLACK, 0, -0.3, 0.35)
estrella = scene.geometry.star(5, 0.8, 0.35).fill(GOLD).move_to(5, 2).glow(GOLD, radius=0.4)
scene.play([
    tarjeta.animate.shadow(BLACK, 0, -0.05, 0.05).scale_to(1.0).duration(0.6),
    estrella.animate.glow(None).duration(0.6),
])
scene.play(tarjeta.animate.shadow(None).duration(0.4))
scene.render()
```

Los efectos no pueden animarse sobre una selección de texto (`texto["x"].animate.glow(...)` lanza `TypeError`); para destacar una parte, usa el resaltador o un cambio de color, descritos en Tipografía cinética.

18.1.3. Postprocesado de todo el cuadro

`PostProcess.shader(fuente)` aplica una función WGSL sobre todo lo que la escena dibuja en 2D: fondo, texto, formas, imágenes y Lottie. La fuente define `gaanim_post(uv, resolution, time)` y lee el color ya renderizado con `gaanim_scene(uv)`. `time` es el segundo exacto de la línea de tiempo, así que un efecto que cambia con el tiempo coincide en la vista previa, los seeks y la exportación.

```
from gaanim import CYAN, GOLD, PostProcess, Scene

acabado = PostProcess.shader("""
fn grano(p: vec2<f32>) -> f32 {
    return fract(sin(dot(p, vec2<f32>(12.9898, 78.233))) * 43758.5453);
}
fn gaanim_post(uv: vec2<f32>, resolution: vec2<f32>, time: f32) -> vec4<f32> {
    let color = gaanim_scene(uv);
    let vineta = smoothstep(0.95, 0.3, distance(uv, vec2<f32>(0.5)));
    let ruido = (grano(floor(uv * resolution) + floor(time * 24.0)) - 0.5) * 0.08;
    return vec4<f32>(color.rgb * vineta + ruido, color.a);
}
""")
scene = Scene(frame=(16, 9), background="#334155", post=acabado)
scene.geometry.rounded_rect(6, 3.5, 0.3).fill("#f8fafc")
punto = scene.geometry.circle(0.5).fill(CYAN).move_to(-2, 0)
scene.geometry.star(5, 0.6, 0.25).fill(GOLD).move_to(2, 0)
scene.play(punto.animate.shift_by(1.5, 0).duration(1.0))
scene.render()
```



Dónde se declara:

- `Scene(post=...)` lo aplica a toda la escena.
- `scene.canvas.post = ...` lo cambia después; `None` lo quita.
- `scene.segment(nombre, post=...)` lo reemplaza mientras el segmento está activo; `post=False` dibuja ese segmento sin postprocesado.

El shader trabaja en píxeles de salida: expresa radios y desplazamientos como fracción de `uv` (o divide por `resolution`) para que el efecto se vea igual a cualquier resolución. También puedes cargar la fuente de un archivo: `PostProcess.shader(Path("assets/grade.wgsl"))`.

Para un fondo procedural o animado, `Background.shader(...)` acepta una función WGSL análoga, `gaanim_background(uv, resolution, time)`, que se dibuja detrás de la escena.

18.1.4. Cadenas de postprocesos y acabados listos

`post` también acepta una lista: cada pasada recibe el cuadro que dejó la anterior, así que el orden importa (el grano va al final para que la viñeta no lo oscurezca). `PostProcess` trae acabados listos que se combinan con tus propios shaders:

- `grain(amount, size)`, `vignette(strength, softness)` y `chromatic_aberration(amount)`: acabado de película.
- `color_grade(exposure, contrast, saturation, temperature)` y `lut(ruta, strength)` con un archivo `.cube`: corrección de color.
- `bloom(threshold, intensity, radius)`: bloom de varios pasos; todo lo que supera `threshold` brilla, igual a cualquier resolución.
- `halftone`, `dither`, `crt`, `pixelate` y `glitch`: estilos.

Un objeto con `.post_process(False)` queda fuera de todas las pasadas: se dibuja encima del cuadro procesado, así que una etiqueta o un HUD se siguen leyendo bajo `pixelate` o `glitch`.

```
from gaanim import CORAL, CYAN, PostProcess, Scene

scene = Scene(frame=(16, 9), background="#0f172a", post=[
    PostProcess.bloom(threshold=0.7, intensity=0.8),
    PostProcess.color_grade(contrast=1.1, temperature=0.2),
    PostProcess.vignette(0.4),
    PostProcess.grain(0.05),
])
scene.geometry.circle(1.2).fill(CYAN).move_to(-2.5, 0)
scene.geometry.star(5, 1.0, 0.45).fill(CORAL).move_to(2.5, 0)
scene.render()
```

Los parámetros de los acabados y los `uniforms` de un shader propio aceptan un `Parameter` (o un `Computed`), así que un efecto se anima como cualquier otro valor: `PostProcess.shader(fuente, uniforms={"amount": amount})` expone `gaanim_uniforms.amount` a la función WGSL.

```
from gaanim import PostProcess, Scene

scene = Scene(frame=(16, 9), background="#1e293b")
golpe = scene.viz.parameter(0.0)
scene.canvas.post = PostProcess.glitch(intensity=golpe, seed=3)
scene.text("¡Corte!", role="title").fill("#f8fafc")
scene.play([golpe.animate.set(1.0).duration(0.2).repeat(2, yoyo=True)])
scene.render()
```

18.1.5. Movimiento: desenfoque, ecos y deformación

- `scene.canvas.motion_blur(180, samples=8)` desenfoca el movimiento en la exportación como el obturador de una cámara de cine: cada fotograma promedia `samples` subfotogramas exactos. La vista previa sigue nítida. `drawable.motion_blur(False)` deja nítido un título o un HUD.
- `drawable.echo(count, delay=0.04, decay=0.6)` deja una estela de copias del objeto tal como estaba antes. Es estado de declaración: cuando el movimiento termina, las copias lo alcanzan.
- `drawable.squash_stretch(amount)` estira el objeto en la dirección de su velocidad y lo aplasta en la perpendicular, conservando el área.

```
from gaanim import CORAL, GOLD, Scene

scene = Scene(frame=(16, 9), background="#0f172a")
scene.canvas.motion_blur(180, samples=8)
bola = scene.geometry.circle(0.4).fill(CORAL).move_to(-5, 1).echo(6, delay=0.05)
gota = scene.geometry.circle(0.4).fill(GOLD).move_to(-5, -1.5).squash_stretch(0.08)
scene.play([
```

```
    bola.animate.move_to(5, 1).duration(1.0),
    gota.animate.move_to(5, -1.5).duration(0.6),
])
scene.render()
```

18.1.6. Fondos vivos

Además de `Background.shader(...)`, hay fondos animados listos, deterministas en cualquier seek: `Background.mesh_gradient(colores)`, `Background.noise_gradient(colores)`, `Background.aurora(colores)` y `Background.dot_grid(spacing)`, con `drift` para desplazar la rejilla.

```
from gaanim import Background, Scene

scene = Scene(frame=(16, 9), background=Background.aurora(["#22d3ee", "#a78bfa", "#34d399"]))
scene.text("Aurora", role="title").fill("#f8fafc")
scene.render()
```

Para repetir formas (`scene.geometry.repeat`, `duplicate`), unir puntos con líneas vivas (`connect`) o animar el trazo (`stroke_profile`, `Updater.dash_flow`), consulta Movimiento con intención.

18.1.7. Límites

- Una transición entre segmentos con shader propio usa `Transition.shader` (o `Transition.preset`), no `shader_effect`: el efecto por objeto no ve los dos segmentos a la vez.
- El postprocesado no se aplica a la exportación SVG ni a otras salidas vectoriales. Con una cámara en perspectiva y en escenas 3D sí se aplica.
- El desenfoque de movimiento solo se ve al exportar y en los snapshots; la vista previa interactiva queda nítida. También afecta al contenido 3D.

18.1.8. Referencia

- Temas y colores: efectos visuales, `PostProcess` y sus acabados, `Canvas.motion_blur`, `Background` y pinceles.
- Animaciones: `animate.glow`, `animate.blur` y `animate.shadow`.
- Texto: `blur_in` y el animador de rango con `blur`.

CAPÍTULO

19. Reactividad

Objetos que dependen de otros: parámetros, puntos y líneas reactivas, trazos y updaters

En esta guía aprenderás a construir objetos que dependen de otros: un punto que recorre un círculo, una línea que une dos objetos en movimiento, un número que muestra un ángulo o una curva que se dibuja sola. Todo sale de una única magnitud animable, igual que en el tutorial.

```
import math
from gaanim import CYAN, GOLD, WHITE, Easing, Scene, computed

scene = Scene(frame=(16, 9), background="#0f172a")
center = (-5, 0)
radius = 2.0

theta = scene.viz.parameter(0.0)
tip = scene.geometry.polar_point(center, radius, theta)
wave_point = scene.geometry.point_ref(
    computed(lambda angle: -2.2 + 1.3 * angle, inputs=[theta]),
    computed(lambda angle: radius * math.sin(angle), inputs=[theta]),
)

orbit = scene.geometry.circle(radius).no_fill().stroke(WHITE, 0.03).move_to(*center)
arm = scene.geometry.tracking_line(center, tip).stroke(GOLD, 0.05)
dot = scene.geometry.dot(0.12).fill(GOLD).follow(tip)
pen = scene.geometry.dot(0.1).fill(CYAN).follow(wave_point)
bridge = scene.geometry.tracking_line(tip, wave_point).stroke(WHITE, 0.02)
wave = scene.geometry.traced_path(pen).stroke(CYAN, 0.06).no_fill()
value = scene.viz.readout(theta, label="$theta$", format=".2f").move_to(-5, 2.9)

scene.play([orbit.animate.create().duration(0.5)])
scene.play([item.animate.fade_in().duration(0.3) for item in (arm, dot, pen, bridge, wave, value)])
scene.play([theta.animate.set(2 * math.pi).duration(3.0).easing(Easing.LINEAR)])
scene.render()
```

19.1. Una sola fuente de verdad

La escena anterior anima un único valor: `theta`. El punto del círculo, el brazo, el punto de la onda, la línea que los une, el trazo y el número se calculan a partir de él en cada fotograma. No hay dos velocidades que puedan desincronizarse.

Esa es la forma recomendada de pensar la reactividad en Gaanim:

1. Crea un `Parameter` con `scene.viz.parameter(valor)` para la magnitud que cambia (un ángulo, un tiempo, una amplitud). Es invisible.
2. Deriva de él puntos, líneas, curvas y números.
3. Anima solo el parámetro con `parameter.animate.set(valor)` dentro de `scene.play([...])`.

Como todo es función del parámetro, Gaanim puede saltar a cualquier instante sin reproducir lo anterior: el editor, la exportación y las capturas muestran exactamente lo mismo.

> **IDEA CLAVE**
>
> Si te descubres sincronizando a mano dos animaciones para que «coincidan», probablemente te falta un parámetro compartido.

19.2. Puntos que dependen de valores

Un `PointRef` es un punto lógico: no se dibuja, pero otros objetos pueden seguirlo o usarlo como extremo. Las fábricas de `scene.geometry` aceptan en cada coordenada un número fijo, un `Parameter` o un valor calculado:

- `polar_point(origen, radio, ángulo)`: un punto a cierta distancia y ángulo (en radianes) de un origen.
- `point_ref(x, y)`: un punto con coordenadas reactivas.
- `offset_point(origen, dx, dy)`: un punto desplazado respecto de otro que puede moverse.
- `point_between(a, b, alpha=0.5)`: un punto entre dos extremos.

Para verlo, haz que un drawable lo siga con `follow`:

```
middle = scene.geometry.point_between(center, tip, alpha=0.5)
marker = scene.geometry.dot(0.08).fill(WHITE).follow(middle)
```

Un objeto con `follow` empieza oculto: aparece cuando incluyes su entrada (`fade_in`, `create`…) en un `scene.play`. Lo mismo vale para las líneas reactivas y los trazos de las secciones siguientes.

19.3. Valores calculados con `computed`

`computed(función, inputs=[...])` crea un valor derivado de otros. La función recibe los valores de `inputs` en el mismo orden y devuelve un número:

```
height = computed(lambda angle: radius * math.sin(angle), inputs=[theta])
readout = scene.viz.readout(height, label="$y$", format=".2f").move_to(-5, -2.9)
```

Un valor calculado puede usarse donde se acepte un parámetro: coordenadas de puntos, radios, extremos de líneas, lecturas numéricas o la cámara. También puede depender de otros valores calculados. Si necesitas el tiempo de la línea de tiempo, decláralo como entrada: `inputs=[scene.time]`.

`scene.viz.readout` acepta además una función con sus entradas directamente:

```
area = scene.viz.readout(lambda r: math.pi * r**2, inputs=[theta], label="$A$")
```

19.4. Líneas y flechas que siguen a sus extremos

`scene.geometry.tracking_line(desde, hasta)` dibuja una línea cuyos extremos se recalculan en el mismo fotograma. Cada extremo puede ser una tupla fija, un drawable, un `PointRef` o un punto de anclaje:

```
from gaanim import Anchor

box = scene.geometry.rect(1.6, 0.9).no_fill().stroke(WHITE, 0.03).move_to(4.5, 2.5)
pointer = scene.geometry.connector(box.anchor_point(Anchor.BOTTOM), tip)
```

`drawable.anchor_point(Anchor.BOTTOM)` es un extremo pegado al borde inferior de `box`, que lo acompaña si el objeto se mueve, rota o escala. `scene.geometry.connector` hace lo mismo que `tracking_line`, pero con punta de flecha; puede aparecer con `grow_arrow()`.

19.5. Dejar un rastro con `traced_path`

`scene.geometry.traced_path(objeto)` registra la posición de un drawable y la dibuja como un trazo. Con `dissipating_time=` los puntos más antiguos desaparecen tras esos segundos:

```
comet = scene.geometry.traced_path(dot, dissipating_time=0.8).stroke(GOLD, 0.04).no_fill()
```

El trazo también se reconstruye al saltar a otro instante, así que no depende de haber reproducido la escena desde el principio.

19.6. Puntos y tangentes sobre una curva

Para recorrer una curva ya dibujada, usa un parámetro entre `0` y `1` con `point_on_curve` y, si quieres, `tangent_on_curve`:

```
t = scene.viz.parameter(0.0)
curve = scene.geometry.polyline([(x / 10, math.sin(x / 10)) for x in range(0, 63)])
runner = scene.geometry.point_on_curve(curve, t)
tangent = scene.geometry.tangent_on_curve(curve, t, length=1.2)
```

`normal_on_curve` y `curvature_on_curve` completan la familia.

19.7. Copiar el movimiento de otro objeto

Cuando una propiedad debe copiar la de otro drawable, usa un **binding**:

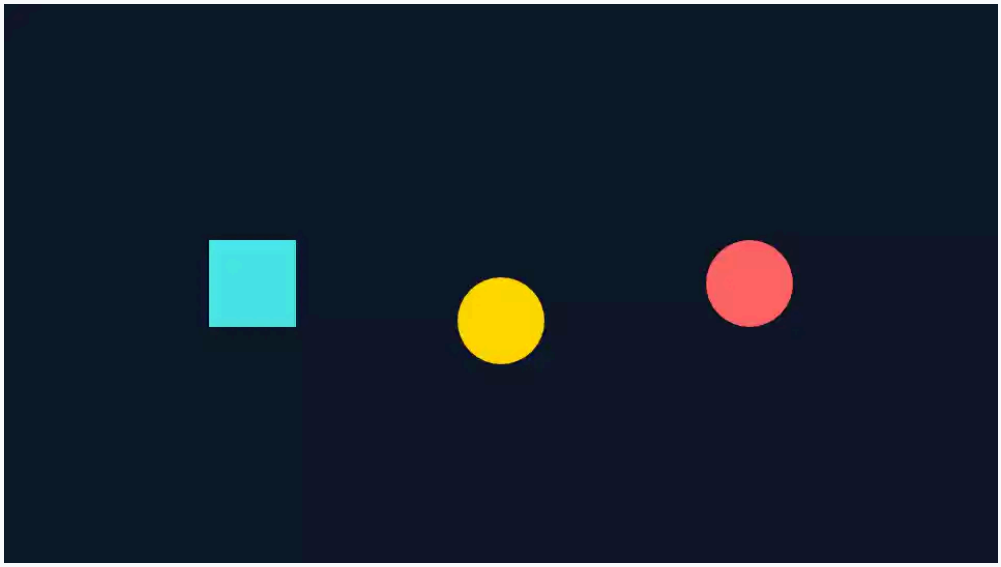
- `bind_x_from(fuente)`, `bind_y_from(fuente)` y `bind_position_from(fuente)` copian la posición.
- `bind_rotation_from(fuente)` copia la rotación, con `ratio=` y `phase=` opcionales; es útil para engranajes.
- `follow(fuente, offset=)` coloca el objeto sobre un extremo, con un desplazamiento opcional.

Para relaciones numéricas (una etiqueta que muestra una distancia, un punto a mitad de camino) prefiere un `PointRef` o `computed`: la dependencia queda explícita.

19.8. Cuándo usar un updater

Un `Updater` es un comportamiento continuo que se añade a un drawable con `add_updater` y se retira con `remove_updater`. Úsalo cuando el movimiento no depende de una magnitud que quieras mostrar o compartir: un logo que respira, un icono que gira, un temblor orgánico.

```
from gaanim import CORAL, CYAN, GOLD, Scene, Updater

scene = Scene(frame=(16, 9), background="#0f172a")
gear = scene.geometry.square(1.4).fill(CYAN).move_to(-4, 0)
buoy = scene.geometry.circle(0.7).fill(GOLD)
badge = scene.geometry.circle(0.7).fill(CORAL).move_to(4, 0)

gear.add_updater(Updater.rotate(1.5))
buoy.add_updater(Updater.oscillate("y", frequency=1.0, low=-0.6, high=0.6))
badge.add_updater(Updater.wiggle(position=0.15, rotation=0.1, frequency=3.0, seed=7))

scene.wait(2.0)
gear.remove_updater()
scene.render()
```



Los presets (`orbit`, `rotate`, `bob`, `pulse`, `advance_x`, `wiggle` y `oscillate`) son funciones del tiempo de la línea de tiempo, así que también son exactos al saltar a cualquier instante. `wiggle` y `oscillate` se suman a las animaciones del objeto en lugar de reemplazarlas.

Si el movimiento sí representa una magnitud (una fase, una posición en una gráfica), vuelve a la primera sección: un `Parameter` es más fácil de mostrar, compartir y controlar.

19.9. Datos medidos

Para reproducir una serie registrada (un sensor, una simulación externa), usa `drive_from_samples` con una lista de tiempos, otra de valores y la propiedad que controlan. Se evalúa sin código Python por fotograma y es exacto al saltar:

```
probe = scene.geometry.dot(0.15).fill(GOLD)
times = [i * 0.05 for i in range(61)]
path = [(math.cos(2 * t), math.sin(3 * t)) for t in times]
probe.drive_from_samples(times, path, "xy", scale=2.5)
```

Los tiempos son relativos al cursor en el que haces la llamada. Un `Parameter` también tiene `drive_from_samples`, y todo lo que dependa de él seguirá la serie.

Para colorear un objeto según una serie, por ejemplo la velocidad de cada auto de una simulación, usa la propiedad `"fill"` con `colors=`. El valor se lee en `domain` (por defecto, el rango de la serie) y elige un color entre ellos:

```
car = scene.geometry.rounded_rect(1.2, 0.6, 0.1)
times = [0.0, 1.0, 2.0, 3.0]
speed = [30.0, 12.0, 2.0, 25.0]
car.drive_from_samples(times, speed, "fill", colors=["#ef4444", "#facc15", "#22c55e"], domain=(0, 30))
```

19.10. Simulaciones con estado

Algunas simulaciones, como una pelota que cae y rebota, acumulan estado de un fotograma al siguiente y no se pueden escribir como una función del tiempo. Para ellas usa `add_updater_fn` con dos argumentos obligatorios en la práctica: `reset`, que devuelve el estado al inicial, y `fixed_dt`, un paso de tiempo fijo. Así Gaanim puede reiniciar la simulación y repetir los pasos al saltar o exportar:

```
from gaanim import GOLD, WHITE, Scene

scene = Scene(frame=(16, 9), background="#0f172a")
floor = scene.geometry.line(-3, -3, 3, -3).stroke(WHITE, 0.03)
ball = scene.geometry.circle(0.3).fill(GOLD).move_to(0, 3)

state = {"y": 3.0, "vy": 0.0}

def reset():
    state["y"], state["vy"] = 3.0, 0.0

def step(position, dt, elapsed):
    state["vy"] -= 9.8 * dt
    state["y"] += state["vy"] * dt
    if state["y"] < -2.7:
        state["y"], state["vy"] = -2.7, -0.8 * state["vy"]
    return (position[0], state["y"], position[2])

ball.add_updater_fn(step, reset=reset, fixed_dt=1 / 240)
scene.play([floor.animate.create().duration(0.3)])
scene.wait(2.2)
```

```
ball.remove_updater()
scene.render()
```

`step(posición, dt, transcurrido)` devuelve la nueva posición local `(x, y, z)`. Sin `reset` y `fixed_dt`, usa `add_updater_fn` solo para comportamientos ligeros que dependan del tiempo transcurrido y no de pasos anteriores.

19.11. Reglas para que todo sea reproducible

Las funciones que pasas a `computed`, `readout`, `plot` o `add_updater_fn` deben ser síncronas y puras: con las mismas entradas devuelven lo mismo. Con el mismo script, las mismas entradas y el mismo entorno de Python, la reproducción, el rebobinado, los saltos y la exportación producen el mismo resultado.

- No leas el reloj del sistema ni archivos dentro de esas funciones.
- Si necesitas azar, usa `scene.random(seed)` o pasa la semilla como entrada; nunca el módulo `random` global.
- No modifiques variables externas desde un `computed`.

Si una función lanza una excepción o devuelve un valor no finito, Gaanim no reutiliza el último valor válido: la geometría muestra un hueco, la lectura numérica muestra `invalid` y los extremos dejan de estar vinculados en ese fotograma. No se garantiza igualdad bit a bit entre plataformas distintas.

19.12. Errores frecuentes

- **El objeto reactivo no aparece:** las líneas reactivas, los trazos y los objetos con `follow` empiezan ocultos. Añade su `fade_in` a un `scene.play`.
- **El movimiento cambia al saltar a otro instante:** hay estado oculto en una función. Conviértelo en entrada de `computed` o usa `reset` y `fixed_dt`.
- **Dos objetos se desincronizan:** derívalos del mismo parámetro en lugar de animarlos por separado.
- **`NameError` o error de aridad en `computed`:** la función debe aceptar exactamente tantos argumentos como elementos tenga `inputs`.

Consulta las firmas completas en la referencia de la escena y en Visualización.

CAPÍTULO

20. Cámara y 3D

Encuadres, zoom, seguimiento y cámaras secundarias en 2D; primitivas, luces y cámara en perspectiva en 3D

EXPERIMENTAL

La cámara 2D es estable. Todo lo 3D de esta página (primitivas, materiales, luces, cámara en perspectiva, ejes y superficies 3D) es experimental: tiene errores conocidos y puede cambiar entre versiones. Úsalo para explorar, no para producción.

En esta guía aprenderás a dirigir la mirada del espectador: acercarte a un detalle, volver al plano general, seguir a un objeto en movimiento, sacudir la imagen y mostrar un detalle ampliado con una cámara secundaria. Al final verás cómo montar una escena 3D sencilla y moverte alrededor de ella.

```
from gaanim import CYAN, GREEN, WHITE, Easing, Scene

scene = Scene(frame=(16, 9), background="#0f172a")
left = scene.geometry.circle(0.8).fill(CYAN).move_to(-4, 0)
right = scene.geometry.square(1.4).fill(GREEN).move_to(4, 0)
label = scene.text("Detalle", role="caption").fill(WHITE).move_to(4, -1.2)

scene.play([
    left.animate.grow_from_center().duration(0.4),
    right.animate.grow_from_center().duration(0.4),
    label.animate.fade_in().duration(0.4),
])
scene.camera.save("general")
scene.play([scene.camera.animate.frame_to([right, label],
margin=0.5).duration(1.0).easing(Easing.SMOOTH)])
scene.play([scene.camera.animate.restore("general").duration(0.8)])
scene.play([scene.camera.animate.shake(trauma=0.6, seed=1)])
scene.render()
```

20.1. La cámara 2D

`scene.camera` controla qué parte de la escena se ve. Trabaja en las mismas unidades lógicas que los objetos (un fotograma de 16 × 9), así que un zoom de 2 muestra la mitad de ancho y de alto.

Cada operación existe en dos formas:

- `scene.camera.zoom_to(2)` aplica el cambio al instante, en el cursor actual.
- `scene.camera.animate.zoom_to(2)` devuelve una animación que pasas a `scene.play([...])` con su duración y su easing.

20.1.1. Mover, acercar y girar

```
scene.play([scene.camera.animate.pan_to(3, 1).duration(0.8)])
scene.play([scene.camera.animate.zoom_to(2.5).duration(1.0)])
scene.play([scene.camera.animate.rotate_to(0.2).duration(0.6)])
scene.play([scene.camera.animate.reset().duration(0.8)])
```

- `pan_to(x, y)` centra la vista en un punto; también acepta un drawable.
- `zoom_to(z)` acerca con valores mayores que 1. Por defecto interpola de forma exponencial para que un zoom grande se perciba a velocidad constante; `interpolation="linear"` recupera la interpolación lineal.
- `rotate_to(ángulo)` gira la vista, en radianes.
- `reset()` vuelve al encuadre inicial.

20.1.2. Encuadrar objetos

`frame_to(objetos, margin=)` calcula el paneo y el zoom necesarios para que los objetos quepan en pantalla con un margen. Es la forma más cómoda de acercarse a un detalle sin calcular coordenadas:

```
scene.play([scene.camera.animate.frame_to([target], margin=0.6).duration(1.0)])
```

Con `dynamic=True` el encuadre se recalcula mientras los objetos se mueven.

20.1.3. Guardar y recuperar encuadres

`scene.camera.save("nombre")` guarda el encuadre actual y `scene.camera.animate.restore("nombre")` vuelve a él. Para definir un encuadre sin aplicarlo, usa `scene.camera.state_2d(...)` con `center`, `zoom` y `rotation`, y anímalo con `scene.camera.animate.to(estado)`:

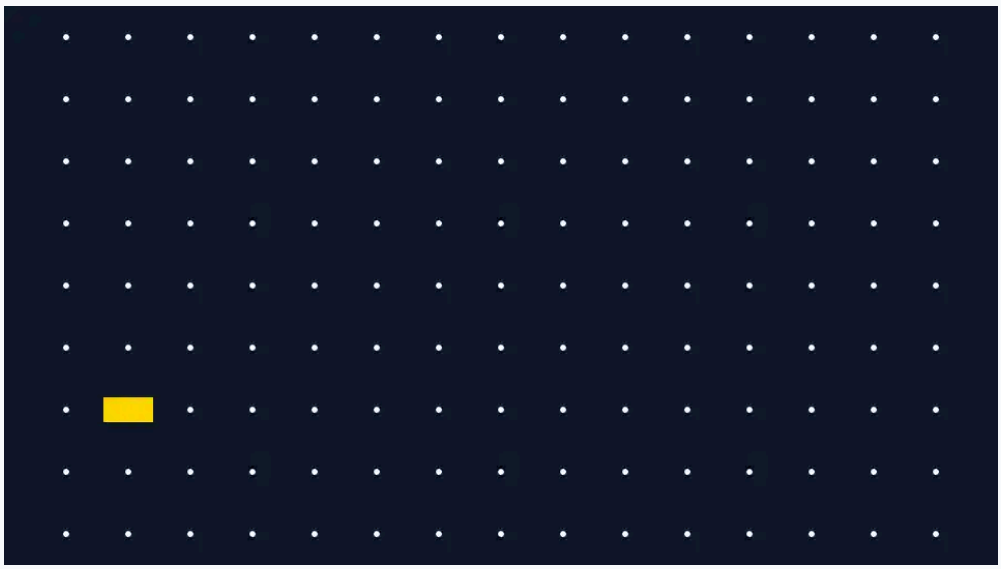
```
overview = scene.camera.state_2d(center=(0, 0), zoom=1.0)
detail = scene.camera.state_2d(center=(3, 1), zoom=3.0)
scene.play([scene.camera.animate.to(detail).duration(1.0)])
scene.play([scene.camera.animate.to(overview).duration(1.0)])
```

20.1.4. Seguir a un objeto

`scene.camera.animate.follow(objeto, lag=)` mantiene el objeto en el centro mientras dura la animación, y al terminar la cámara se queda donde lo dejó. `lag` añade un retraso suave, en segundos:

```
from gaanim import GOLD, WHITE, Easing, Scene

scene = Scene(frame=(16, 9), background="#0f172a")
for x in range(-7, 8):
    for y in range(-4, 5):
        scene.geometry.dot(0.05).fill(WHITE).move_to(x, y)
car = scene.geometry.rect(0.8, 0.4).fill(GOLD).move_to(-6, -2)
```

```
scene.play([
    car.animate.move_to(6, 2).duration(2.0).easing(Easing.SMOOTH),
    scene.camera.animate.follow(car, lag=0.2).duration(2.0),
])
scene.render()
```



Para un vínculo que dure varias animaciones, `scene.camera.bind_2d(center=, zoom=, rotation=)` conecta la cámara a extremos o parámetros y devuelve un `CameraConstraint` que puedes desactivar con `disable()`:

```
zoom = scene.viz.parameter(1.0)
constraint = scene.camera.bind_2d(center=car, zoom=zoom)
scene.play([zoom.animate.set(2.0).duration(0.8)])
constraint.disable()
```

Consulta Reactividad para crear esos parámetros y extremos.

20.1.5. Sacudir la imagen

`shake()` produce un temblor determinista que termina en reposo. `trauma` (entre `0` y `1`) controla la intensidad, `decay` cuánto tarda en apagarse y `seed` fija el patrón:

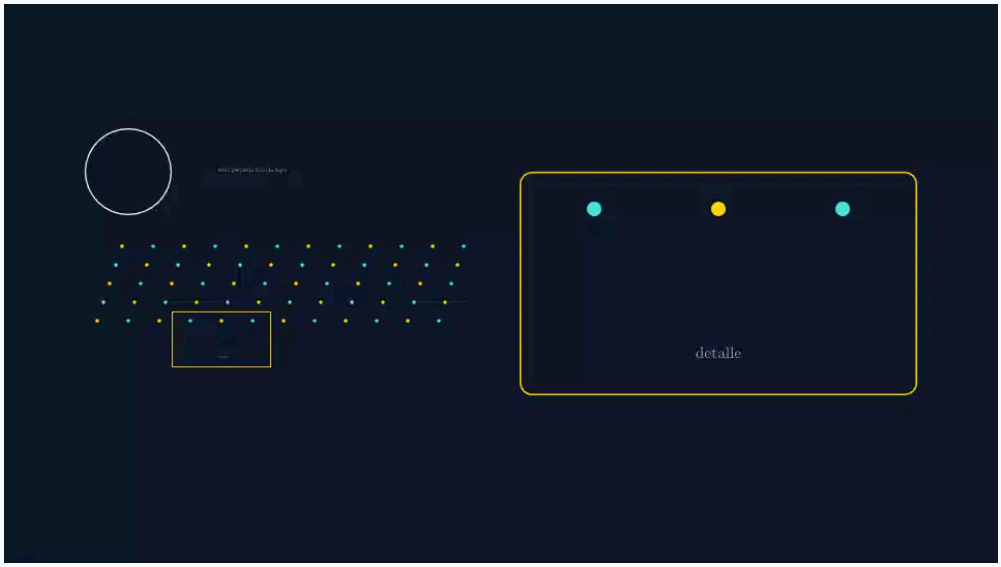
```
scene.play([scene.camera.animate.shake(trauma=0.8, decay=1.5, seed=3)])
```

20.1.6. Cámaras secundarias

`scene.camera` mueve toda la imagen. Para enseñar un detalle ampliado sin perder el plano general, convierte una figura en una pantalla con `pantalla.camera_view(marco)`: la pantalla muestra lo que ve una segunda cámara que encuadra `marco`. Es el *picture-in-picture* de un vídeo o la lupa sobre un mapa.

```
from gaanim import CYAN, GOLD, WHITE, Scene

scene = Scene(frame=(16, 9), background="#0f172a")
for i in range(60):
    scene.geometry.dot(0.03).fill(CYAN if i % 2 else GOLD).move_to(-6.5 + 0.1 * i, 0.3 * (i % 5) - 0.6)
scene.text("detalle", role="caption").scale_to(0.2).move_to(-4.5, -1.2)
scene.text("letra pequeña bajo la lupa", role="caption").scale_to(0.3).move_to(-4, 1.8)

marco = scene.geometry.rect(1.6, 0.9).no_fill().stroke(GOLD, 0.03).move_to(-4.5, -0.9)
```

```
pantalla = scene.geometry.rounded_rect(6.4, 3.6, 0.2).stroke(GOLD, 0.05).move_to(3.5, 0)
pantalla.camera_view(marco)

lupa = scene.geometry.circle(0.7).stroke(WHITE, 0.04).move_to(-6, 1.8)
foco = scene.geometry.circle(0.25).no_fill().no_stroke().move_to(-6, 1.8)
lupa.camera_view(foco)

scene.play([
    marco.animate.move_to(-2, 0).scale_to(0.5).duration(1.5),
    lupa.animate.move_to(-2, 1.8).duration(1.5),
    foco.animate.move_to(-2, 1.8).duration(1.5),
])
scene.render()
```



- El marco es la cámara: moverlo desplaza la vista, reducirlo acerca y girarlo gira la vista en sentido contrario. Se anima con `animate` como cualquier objeto.
- El aumento es el tamaño de la pantalla entre el del marco: una pantalla de 6,4 de ancho con un marco de 1,6 amplía cuatro veces. `fit="cover"` llena la pantalla aunque las proporciones no coincidan y `fit="stretch"` deforma el marco hasta encajarlo.
- La pantalla dibuja su relleno, encima la vista recortada a su contorno y encima su trazo. Cualquier figura cerrada sirve: un círculo hace de lupa.
- La vista no muestra la pantalla ni el marco. Oculta más objetos con `exclude=[...]`; si el marco solo debe hacer de cámara, quítale el trazo.
- El fondo de la vista es el de la escena. Cámbialo con `background=` (un color o un `Brush`), o pasa `background=None` para que se vea el relleno de la pantalla.
- La cámara ve la escena aunque la región quede fuera de la cámara principal, así que puedes mostrar a la vez dos zonas alejadas.

20.1.6.1. Zoom, paneo y pop-out

`camera_view` devuelve un `CameraView` con el mismo vocabulario que `scene.camera`: `pan_to`, `zoom_to` y `rotate_to`, al instante o con `animate`. Con `zoom=` la vista tiene su propio aumento y el tamaño del marco deja de contar; sin marco, la cámara mira a `center` a través de un marco invisible. `animate.zoom_to` cambia ese zoom a velocidad percibida constante, como el de la cámara principal.

`animate.pop_out()` hace brotar la pantalla de la región que ve su cámara: empieza encogida sobre esa región, mostrando la escena a tamaño real, y crece hasta su sitio; si es su entrada, hasta entonces está oculta. `animate.pop_in()` la devuelve y la oculta al llegar.

```
vista = scene.geometry.rounded_rect(5, 2.8, 0.2).stroke(WHITE, 0.05).move_to(-3.5, -2.3)
vista = vista.camera_view(center=(-4.5, -0.9), zoom=3)
scene.play([vista.animate.pop_out().duration(0.8)])
scene.play([vista.animate.pan_to(-1, 0).duration(1.0), vista.animate.zoom_to(6).duration(1.0)])
scene.play([vista.animate.pop_in().duration(0.6)])
```

20.1.6.2. Insets con conectores

`scene.camera.inset(objetivo, zoom=, at=)` monta de una vez la pantalla, un marco sobre el objetivo que recuadra lo que muestra y dos conectores entre ambos. `at` coloca la pantalla en una esquina o un borde del fotograma, o en un punto. Con `follow=True` el marco persigue a un objeto en movimiento, y `view.zoom` es un valor reactivo que puedes mostrar con un readout.

```
from gaanim import CYAN, Anchor, Easing, Scene

scene = Scene(frame=(16, 9), background="#0f172a")
for x in range(-7, 8):
    for y in range(-4, 2):
        scene.geometry.dot(0.04).fill("#334155").move_to(x, y)
bola = scene.geometry.circle(0.12).fill(CYAN).move_to(-6, -3)
view = scene.camera.inset(bola, zoom=5, at=Anchor.TOP_RIGHT, follow=True)
scene.viz.readout(view.zoom, format=".0f", prefix="×").move_to(4.5, 0.4)

scene.play([view.animate.pop_out().duration(0.8)])
scene.play([bola.animate.move_to(5, -2).duration(2.0).easing(Easing.SMOOTH)])
scene.play([view.animate.zoom_to(2).duration(1.0)])
scene.render()
```

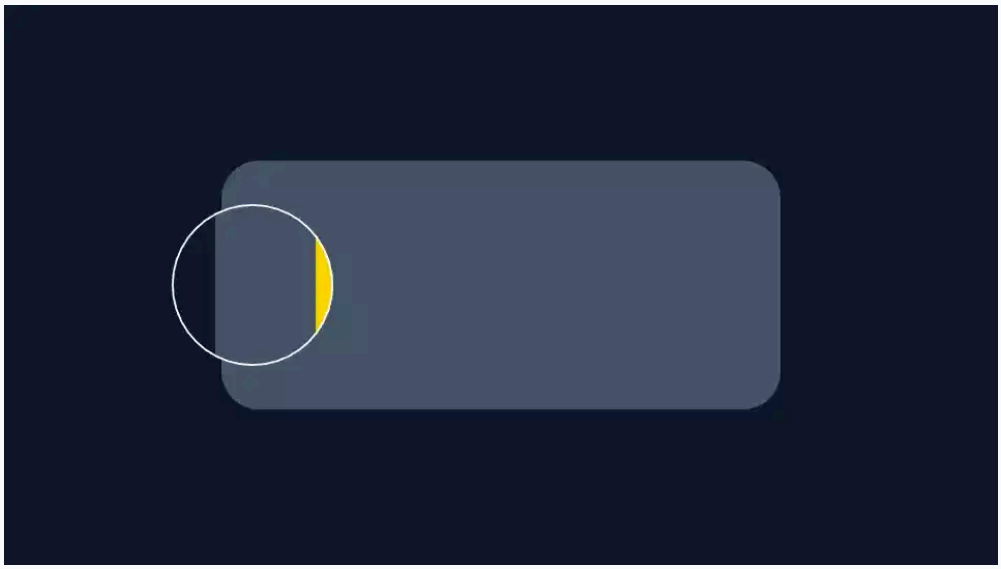
Con `fixed=True` la pantalla se queda fija en la imagen aunque `scene.camera` se desplace o acerque, como cualquier objeto con `.hud()`.

20.1.6.3. Capas: la lupa de rayos X

`drawable.view_layer("nombre")` saca un objeto de la cámara principal: solo lo muestran las vistas que listan esa capa en `layers=`. Así una lupa puede revelar lo que hay debajo de una superficie.

```
from gaanim import GOLD, WHITE, Scene

scene = Scene(frame=(16, 9), background="#0f172a")
scene.geometry.rounded_rect(9, 4, 0.6).fill("#475569").no_stroke()
for x in (-3, -1, 1, 3):
    scene.geometry.rect(0.3, 3).fill(GOLD).no_stroke().move_to(x, 0).view_layer("rayos_x")

lupa = scene.geometry.circle(1.3).stroke(WHITE, 0.06).move_to(-4, 0)
```

```
vista = lupa.camera_view(center=(-4, 0), zoom=1.2, layers=["rayos_x"])
scene.play([
    lupa.animate.move_to(4, 0).duration(3.0),
    vista.animate.pan_to(4, 0).duration(3.0),
])
scene.render()
```



20.2. Escenas 3D

Gaanim puede mezclar objetos 3D con el resto de la escena. Empieza con una composición estática: luz, objetos y una cámara que los vea bien. Cuando sea legible, anima la cámara.

```
from gaanim import CYAN, GOLD, Material3D, Scene

scene = Scene(frame=(16, 9), background="#0f172a")
scene.geometry.lighting_3d("studio", intensity=1.0)

cube = scene.geometry.cube(2.0, material=Material3D.matte(CYAN)).move_to_3d(-1.8, 0, 0)
sphere = scene.geometry.sphere(1.1, material=Material3D.metal(GOLD)).move_to_3d(1.8, 0, 0)

scene.camera.perspective(fov_y=0.7)
scene.camera.look_at(eye=(0, 3, 10), target=(0, 0, 0))

scene.play([cube.animate.create().duration(0.5), sphere.animate.create().duration(0.5)])
scene.play([scene.camera.animate.orbit(delta_yaw=0.8, delta_pitch=0.3).duration(1.2)])
scene.play([scene.camera.animate.dolly(0.8).duration(0.6)])
scene.render()
```

20.2.1. Objetos, materiales y luz

- `scene.geometry` crea primitivas 3D: `cube`, `sphere`, `cylinder`, `cone` y `plane`.
- `Material3D.matte(color)`, `Material3D.metal(color)` y `Material3D.emissive(color, strength)` son puntos de partida; el constructor `Material3D(color, roughness=, metallic=)` permite ajustarlos.
- `scene.geometry.lighting_3d("studio")` ilumina las primitivas con material al estilo de Gaanim: luz del cielo y del suelo, que aclara las caras que miran arriba y oscurece las que miran abajo; una luz principal suave desde arriba a la izquierda, y un leve brillo en los bordes de la silueta. El color del material se mantiene legible: una cara iluminada muestra más o menos su color, y una en

sombra, la mitad. Los materiales pulidos (poca rugosidad) tienen un reflejo blanco y los metálicos reflejan el cielo.
- `scene.geometry.light_3d()` añade una luz puntual (o `"directional"`, como el sol) que se coloca con `move_to_3d` y se anima como cualquier objeto. Mientras hay luces, sustituyen a la luz principal fija, cada una en su color e intensidad; `fade_in` y `fade_out` la encienden y la apagan. Con `lighting_3d("none")` solo alumbran ellas. `"none"` deja solo la emisión del material.
- Las caras planas se pintan de un solo color y las curvas con degradados suaves.
- Los drawables tienen versiones 3D de sus transformaciones: `move_to_3d`, `shift_by_3d`, `rotate_by_3d(eje, radianes)` y `scale_by_3d`, tanto inmediatas como bajo `.animate`.

Los ejes son los de la escena 2D: x hacia la derecha, y hacia arriba y z hacia el espectador.

20.2.2. La cámara en perspectiva

- `perspective(fov_y)` activa la proyección en perspectiva; `fov_y` es el campo de visión vertical en radianes.
- `look_at(eye=, target=, up=)` coloca la cámara en `eye` mirando a `target`.
- `animate.orbit(delta_yaw=, delta_pitch=)` gira alrededor del objetivo.
- `animate.dolly(factor)` se acerca (menor que 1) o se aleja (mayor que 1).
- `orthographic()` vuelve a la proyección ortográfica y `reset()` al encuadre 2D inicial.

20.2.3. Ejes y superficies 3D

`scene.viz.cartesian_3d` crea unos ejes 3D sobre los que puedes dibujar superficies, curvas paramétricas y campos. Una superficie puede depender de parámetros, como cualquier objeto reactivo:

```
import math
from gaanim import Axis, Scene

scene = Scene(frame=(16, 9), background="#0f172a")
space = scene.viz.cartesian_3d(
    Axis.linear(-3, 3).ticks(1),
    Axis.linear(-3, 3).ticks(1),
    Axis.linear(-1, 1).ticks(0.5),
    size=(6, 6, 3),
)
phase = scene.viz.parameter(0.0)
surface = space.surface(lambda x, y, p: math.sin(math.hypot(x, y) - p), inputs=[phase])

scene.camera.perspective(fov_y=0.8)
scene.camera.look_at(eye=(7, -9, 6), target=(0, 0, 0), up=(0, 0, 1))
scene.play([space.animate.create().duration(0.6), surface.animate.fade_in().duration(0.6)])
scene.play([phase.animate.set(2 * math.pi).duration(2.0)])
scene.render()
```

La coordenada z de los datos apunta hacia el espectador, así que para ver la superficie como un relieve se usa `up=(0, 0, 1)` en `look_at`.

20.2.4. Cómo se dibuja el 3D

Gaanim dibuja el 3D con Vello, igual que el 2D: proyecta las mallas de triángulos y las líneas 3D a través de la cámara de la escena, las ordena de atrás hacia delante y calcula la luz en la CPU. De ahí salen sus límites:

- Las primitivas con `Material3D` reciben luz y ocultan sus caras traseras. Las mallas sin material (superficies, gráficos 3D) no reciben luz y se ven por las dos caras; las líneas 3D tampoco reciben luz y miden 1,5 px.
- No hay sombras: `lighting_3d(shadows=True)` se acepta, pero ningún objeto proyecta sombra.
- No hay búfer de profundidad: los triángulos se ordenan por profundidad, así que las superficies que se cruzan pueden mostrar errores de orden.

- Todo el contenido 3D se dibuja debajo del 2D: etiquetas, textos y HUD quedan siempre encima.

Como es dibujo normal de Vello, el 3D aparece en las exportaciones, las capturas, los paquetes `.gaanim` y las vistas previas de Presenter View, y admite desenfoque de movimiento y postprocesado.

20.3. Consejos

- Valida primero una escena quieta; añade el movimiento de cámara al final.
- Los objetos 3D usan las mismas unidades lógicas que el resto de la escena; mide tamaños y distancias de cámara con la misma escala.

Las firmas completas de la cámara están en la referencia de la escena.

CAPÍTULO

21. Presentaciones

Crea, ensaya y presenta diapositivas con notas y pausas

En esta guía aprenderás a preparar una charla con Gaanim: diapositivas con plantillas reutilizables, notas del orador, pausas donde esperar a que hables, el guion de la charla en un archivo aparte, su tiempo grabado diapositiva a diapositiva, ensayos con un teleprompter y la vista del presentador en un segundo monitor.

```python
from gaanim import Scene, lecture, title_slide

scene = Scene(frame=(16, 9), margin=0.9)
scene.canvas.set_theme("presentation")
scene.slides.brand(footer="MI CHARLA", slide_numbers=True, rule=True)

cover = scene.segment("Portada", template=title_slide, notes="Presenta el tema.")
title = scene.text("Una idea clara", role="title")
subtitle = scene.text("Diapositivas reutilizables", role="subtitle")
cover.bind(title=title, subtitle=subtitle)
scene.play([title.animate.write().duration(0.8), subtitle.animate.fade_in().duration(0.6)])
scene.stop("portada")

content = scene.segment("Contenido", template=lecture, notes="Desarrolla la idea.")
heading = scene.text("Contenido", role="title")
body = scene.text("Una sola idea por diapositiva.", size=0.42)
content.bind(title=heading, body=body)
scene.play([heading.animate.fade_in().duration(0.4)])
scene.stop("titulo")
scene.play([body.animate.write().duration(0.8)])
scene.stop("mensaje")
scene.render()
```

La vista previa se reproduce de corrido porque, fuera del modo presentación, las pausas no detienen nada. Al presentar, Gaanim espera en cada `scene.stop(...)` hasta que avanzas.

21.1. Crear el proyecto

```
gaanim init slides mi-charla
gaanim mi-charla
```

También puedes ejecutar `gaanim` sin argumentos, elegir **New presentation**. Si aún no tienes Gaanim listo, sigue Instalación. El proyecto de ejemplo no incluye colores, fuentes ni contenido de ninguna institución: la identidad visual es tuya.

21.2. Diapositivas, pasos y notas

Una presentación de Gaanim es una escena normal organizada así:

- Cada `scene.segment("nombre", ...)` es una **diapositiva**; acepta una plantilla (`template=`) y notas (`notes=`). Su nombre y sus notas aparecen en la vista del presentador. Las notas también pueden vivir en un archivo aparte (ver El guion de la charla).
- Cada `scene.stop("nombre")` dentro de una diapositiva es un **paso**: el punto donde la reproducción espera a que avances. Un paso escrito mientras la diapositiva aún entra con su transición espera a que termine.
- `segment.bind(...)` rellena los huecos de la plantilla con tus objetos.

`scene.slides.brand(...)` configura para toda la presentación el logo, el pie, la numeración y la línea separadora. Las plantillas `title_slide`, `lecture`, `comparison` y `credits` devuelven un layout adaptable y puedes reemplazarlas por funciones de Python propias (ver Layout).

21.3. El guion de la charla

Lo que vas a decir puede vivir fuera del código, en un archivo Markdown que escribes con cualquier editor: una sección `## nombre` por diapositiva, con el nombre de su `scene.segment(...)`, y una línea `[stop]` donde avanzas.

```
# Mi charla

## Portada
Buenos días. Hoy les cuento cómo crecieron las ventas.
[stop titulo]

## Resultados
Primero miren el gráfico de ventas.
[stop grafico]
Como ven, crece cada trimestre.
[stop]
Y la conclusión es clara.
```

```
scene.segment("Portada")
scene.wait(1)
scene.stop("titulo")
scene.segment("Resultados")
scene.wait(1)
scene.stop("grafico")
scene.wait(1)
scene.stop()
```

- El texto antes de la primera marca es lo que dices mientras la diapositiva espera en su primer stop: al terminarlo, avanzas. El texto antes de la segunda va en el segundo stop, y así. Lo que sigue a la última marca va después del último stop.
- `[stop grafico]` nombra el stop. `gaanim check` avisa si no coincide con el de `scene.stop("grafico")`, si una diapositiva tiene más o menos marcas que stops y si una sección no corresponde a ninguna diapositiva ni bloque de voz.
- El archivo es `narration/script.md` dentro de la carpeta de assets del proyecto (`assets_dir` de su `gaanim.toml`, `assets` por defecto); fuera de un proyecto, o si un proyecto anterior ya tiene `narration/` junto al script y no en sus assets, junto al script. `scene.narration_script("guion.md")` elige otro. Es el mismo guion de la narración: una sección puede ser también el texto de un `scene.voiceover(...)`.
- La sección de una diapositiva manda sobre su `notes=`; sin sección, se usan las notas del código, que también admiten líneas `[stop]`. Los encabezados se comparan sin distinguir mayúsculas, los `###` son notas para ti que no se muestran y los comentarios HTML se ignoran.
- Al guardar el archivo, el editor actualiza las notas sin volver a compilar la escena.
- En la barra de tiempo del editor, una línea sobre cada diapositiva dice cómo va su guion: violeta a todo lo ancho si todos sus tramos tienen texto, escrito o dictado, y dorada hasta la mitad si solo algunos. Sin línea, está pendiente. Las pausas son rombos sobre la pista, huecos hasta que llegas a ellas (marcas finas cuando hay demasiadas para verlas como rombos), y cuando los nombres de los segmentos comparten sección (`Fundamentos · densidad`), una fila encima nombra cada sección; un clic en ella salta a su primer segmento.

Gaanim escribe el guion por ti y lo mantiene al día con el código. En el editor, abre **Script & narration** (el botón de micrófono de la barra de reproducción, **More controls** o la paleta de comandos) y pulsa el icono del libro de su cabecera: el archivo nace con una sección por diapositiva, con sus `notes=` como punto de partida, y una marca `[stop nombre]` por cada stop. Cuando el código gana diapositivas o stops, la cabecera lo avisa (`script.md · 3 to write`) y el mismo icono añade las secciones nuevas detrás de la diapositiva anterior y las marcas que faltan al final de cada sección, sin tocar lo que escribiste. El guion se abre en tu editor de código, como el resto de los saltos al código (ver `GAANIM_EDITOR` en la CLI), en lo primero que añadió o, si no faltaba nada, en la primera sección sin texto.

21.3.1. El panel Script & narration

El panel se acopla a la derecha del editor y la escena se encoge a su lado, sin quedar tapada. De arriba abajo:

- **TARGET** es la duración de la charla (`15:00`, `15m`, `1h30m`). Es la misma que fija Presenter View: cambiarla en un sitio la cambia en el otro, y Gaanim la recuerda para ese archivo. A su lado dice de dónde viene (`set here`, `from scene.slides.target`…). Vaciarla quita la fijada en el panel.
- **TAKES** suma los tiempos grabados de las diapositivas y **LAST PRACTICE** da la última práctica, cada uno frente al objetivo (`7:40 left`, `0:35 over`). La barra de debajo muestra lo grabado, una marca blanca para la última práctica y una dorada para el objetivo; con dos prácticas o más, **PRACTICE RUNS** dibuja las últimas doce como columnas.
- **TALK MAP** tiene un bloque por diapositiva, tan ancho como su parte del objetivo y lleno según lo que tardó frente a ella. La diapositiva actual lleva un borde blanco, y un clic lleva el cabezal a cualquiera.
- **SCRIPT** es el guion de la diapositiva en pantalla: cuántas palabras dice y cuánto se tarda en leerlas a tu ritmo (el medido al dictar o el fijado en **Settings → Prompter**), su parte del objetivo (**Share**), su toma y su práctica, y `Script 0:05 too long` si el texto no cabe en su parte. **Rehearse with the prompter** abre el prompter sobre la vista previa y **Open in the editor** abre el guion en su sección.

- La lista de diapositivas enseña los tiempos de las tomas (**Takes**) o los de la última práctica (**Last practice**). **No time** deja solo las que aún no tienen tiempo y **Over** las que se pasan de su parte, y **Filter slides** busca por nombre. Cada fila tiene una barra con su parte marcada, el tiempo, y los botones de escuchar y regrabar (o de practicar desde ahí).
- Abajo quedan los controles: **Record** para grabar o **Practice** para practicar.

La parte de cada diapositiva sale de su `target=`, si lo tiene. Lo que esos objetivos dejan del total se reparte entre las demás según cuántas palabras dice su guion, y una diapositiva sin texto recibe como mínimo la parte de 15 palabras, porque también se ve un momento. Los colores son los de toda la app: violeta si cabe, ámbar si se pasa un poco (un 10 %, al menos 10 segundos) y rojo si se pasa más.

Para recorrer el flujo completo con un proyecto de ejemplo y capturas (practicar, recortar, grabar y convertirlo en vídeo o presentación), ver Ensayar y grabar una charla.

21.3.2. Grabar la charla y ajustar sus tiempos

Graba la charla presentándola en el editor y Gaanim apunta cuánto tardas en cada diapositiva, suma el total y lo compara con lo que quieres que dure. Después vuelves a grabar solo las diapositivas que se pasan hasta que la charla cabe. Funciona aunque el guion esté vacío: si además dictas con Scribe, la misma grabación lo escribe (ver Dictar el guion con Scribe).

Con la lista en **Takes**, cada fila muestra el tiempo grabado de la diapositiva; ▸ despliega sus stops, cada uno con su tiempo y sus botones, y un clic en un nombre lleva el cabezal allí.

1. Fija la duración en **TARGET**, o planifica la charla con `scene.slides.target(...)` o con un `target=` en cada diapositiva (ver El tiempo de la charla). Sin plan también se graban los tiempos, pero no hay con qué compararlos.
2. En **Record**, **Test level** abre el micrófono y muestra su nivel sin grabar.
3. **From the start** presenta desde la primera diapositiva y **Record from here** desde la que está en pantalla. Tras la cuenta atrás, presenta hablando: → o `Espacio` avanzan en cada stop, `Enter` termina y `Esc` lo descarta todo.
4. Cada diapositiva que recorres entera guarda su tiempo, desde que entras hasta que sales, y el de cada tramo entre stops. La diapositiva en la que pulsas `Enter` cuenta si ya estabas en su último stop.
5. Mira el panel: **TAKES** dice cuánto te queda o te pasas, y **Over** deja las diapositivas que se salen de su parte.
6. **This slide**, o el botón de grabar de una fila que ya tiene tiempo, vuelve a presentar solo esa diapositiva: la grabación termina sola cuando sales de ella y su tiempo sustituye al anterior. En una fila sin tiempo, el botón presenta desde ahí hasta que pulsas `Enter`.
7. Para ajustar menos, despliega la diapositiva y pulsa grabar en un stop: la grabación arranca justo al dejar el stop anterior (o al entrar en la diapositiva, para el primero) y termina sola cuando avanzas en ese stop. Su tiempo sustituye al del tramo, y el de la diapositiva se recalcula.

Mientras grabas, el subtítulo del editor (ver El prompter) muestra, además del guion, el reloj de la grabación, el nivel del micrófono, el tiempo que llevas en la diapositiva frente a su objetivo y `talk ≈ 3:40 of 4:00`: lo que duraría la charla con lo que llevas grabado y los tiempos guardados del resto de diapositivas.

Con **Keep the audio** marcado, cada grabación guarda además el audio de cada tramo que cronometra: un archivo WAV por tramo entre stops, nombrado por su diapositiva y su número. Regrabar una diapositiva sustituye todos sus tramos, y regrabar un stop solo el suyo. El botón de reproducir de cada fila lo escucha: la diapositiva entera, tramo tras tramo, o solo ese stop.

Ese audio sirve para escucharte y corregir, no para el vídeo: los tiempos y el audio se guardan en la carpeta de datos de Gaanim (en Windows, `%LOCALAPPDATA%\Gaanim\Gaanim\data\rehearsals`), junto a los ensayos de ese archivo, nunca en el proyecto, así que no acaban en un repositorio ni en una exportación. Para que tu voz forme parte del vídeo exportado, graba en cambio una toma en vivo o bloques de voz en off, que sí se guardan en la carpeta `narration/` del proyecto (ver Narración). Si la escena tiene una toma en vivo (`scene.live_take`), se graba siempre entera: las filas no se regraban por separado.

21.3.3. Practicar contra un objetivo

Practicar es presentar la charla cronometrada sin grabar nada: ni audio, ni guion, ni los tiempos de las tomas. Cada práctica se guarda como un ensayo, y el panel la compara con el objetivo y con las anteriores.

1. Fija la duración en **TARGET**.
2. En los controles de abajo, elige **Practice** y pulsa **Practise from the start**, o **From here** para empezar en la diapositiva en pantalla (el botón de cada fila, con la lista en **Last practice**, también empieza ahí).
3. Tras la cuenta atrás, presenta como en la charla: → o `Espacio` avanzan, `Enter` termina y `Esc` la descarta. El prompter muestra el guion y, en su línea de estado, `PRACTICE`, el tiempo en la diapositiva frente a su parte del objetivo, `0:12 behind` o `0:08 ahead`, y el reloj frente al objetivo (`4:12 / 15:00`). Adelantado o retrasado cuenta enteras las diapositivas ya dejadas, y la actual solo cuando se pasa de su parte.
4. Al terminar, el panel dice cómo fue (`Practice: 14:52 of 15:00 · 0:08 to spare`). **LAST PRACTICE** y **PRACTICE RUNS** se actualizan, y la lista en **Last practice**, con el filtro **Over**, señala dónde recortar.

Funciona en los dos sentidos: las presentaciones en Presenter View también cuentan como prácticas, y el objetivo que fijas en el panel es el que usa Presenter View. Como todo ensayo, una práctica de menos de 30 segundos no se guarda.

21.3.4. Dictar el guion con Scribe

En lugar de escribir el guion, puedes dictarlo mientras grabas la charla: Scribe Live, el servicio de transcripción de Zoom, convierte tu voz en texto y cada frase va a la diapositiva y al tramo que estabas presentando.

1. Consigue un token de Scribe en Zoom Platform Studio (**View JWT Token**). Elige la caducidad más corta que te sirva: si se filtrara, deja de valer antes.
2. En **Settings → Dictation** (`Ctrl+,`, o el icono de ajustes de la cabecera de **Script & narration**), pega el token y pulsa **Save in Windows** (ver El token).
3. En los controles **Record** del panel, deja marcado **Dictate into the script**, que se activa solo en cuanto hay un token, y graba la charla o una diapositiva (ver Grabar la charla).
4. Al terminar con `Enter`, el cabezal se queda donde llegaste. Si Scribe se cortó antes (un error, el límite de minutos o dos minutos sin voz), se queda donde se cortó, para grabar de nuevo lo que falta; `Esc` lo descarta todo y vuelve a donde empezaste.
5. Gaanim espera las últimas frases (como mucho 12 segundos; `Esc` deja de esperar) y las escribe en el guion: lo que dices en un tramo pasa a ser su texto. Si el tramo ya tenía texto, lo sustituye y deja el anterior justo debajo en un comentario `<!-- previous: … -->`, que el guion ignora; al regrabar de nuevo, ese comentario se cambia por el del último texto, así que nunca se acumulan. Tus propios comentarios se quedan donde están.
6. Al regrabar una diapositiva o un stop, solo se escribe en ellos: lo que digas mientras avanzas fuera no cae en el tramo siguiente.

Durante el dictado, el subtítulo del editor muestra el guion del tramo que toca decir; si el tramo aún no tiene guion, muestra en vivo lo que Scribe va escribiendo, frase a frase. Gaanim mide además tu ritmo al hablar (palabras por minuto) para que el prompter ilumine el guion a tu velocidad. Si la escena tiene una toma en vivo, se dicta con sus paradas originales y la toma grabada no cambia; para grabarla y dictar a la vez, marca **Also dictate the script with Scribe** junto a **Record live take**.

- Idiomas: `es-ES`, `en-US`, `pt-BR`, `pt-PT`, `fr-FR`, `de-DE`, `it-IT`, `ja-JP`, `zh-CN`, `ar-SA` y `ar-AE`; el español de Scribe es `es-ES`. Se elige en **Settings → Dictation** o, para todo el proyecto, con `language` en `gaanim.toml` (**Pin in gaanim.toml**, con el proyecto abierto en el editor), que manda sobre el de los ajustes.
- Scribe asigna cada frase completa al tramo en el que empezó a decirse: haz una pausa breve al avanzar para que la frase siguiente caiga en su tramo.
- **Send the names of slides and stops as vocabulary** ayuda a Scribe con nombres propios y términos; está desactivado, porque envía esos nombres a Zoom.

Scribe Live cobra por minuto de audio recibido, así que Gaanim lo limita: una sola sesión a la vez, sin reconexiones automáticas si falla, y la sesión se corta sola al llegar a los **Max minutes per session** de los ajustes (60 por defecto, nunca más de los 120 de Zoom), tras dos minutos sin voz o si el micrófono deja de enviar audio, aunque el editor esté en segundo plano.

21.3.4.1. El token

El token de Scribe es de tu cuenta de Zoom, no del proyecto, y Gaanim nunca lo guarda en la carpeta del proyecto: no puede acabar en un repositorio. Gaanim lo busca en este orden:

1. **Pegado en Settings → Dictation:** vive solo en la memoria de esta sesión del editor.
2. **Guardado en Windows** (recomendado): el Administrador de credenciales de Windows lo cifra para tu usuario, como `gaanim/scribe-token`. Puedes verlo o borrarlo allí, o con **Remove saved token**.
3. **La variable de entorno `GAANIM_SCRIBE_TOKEN`** de tu usuario. Es cómoda en otros sistemas, pero cualquier programa que ejecutes la puede leer, incluido el código Python de las escenas.

No lo escribas en el script, en `gaanim.toml` ni en un archivo `.env` del proyecto. Gaanim no lo muestra en el panel, en los mensajes ni en la consola, y solo lo envía en la cabecera de la conexión cifrada con Zoom. Si crees que se filtró, genera otra clave en Zoom Platform Studio: la anterior deja de valer.

21.4. Comprobar y presentar

```
gaanim check mi-charla --strict
gaanim --present --monitor 1 --duration 20m mi-charla
```

`gaanim check` revisa segmentos, notas, paradas y el formato 16:9. Los índices de monitor empiezan en cero. La exportación ignora las paradas y genera un video continuo.

21.4.1. Ensayar una parte

```
gaanim mi-charla --sections portada,contenido
gaanim --present mi-charla --from contenido
```

`--sections` reproduce solo los segmentos indicados y `--from` empieza en uno y sigue hasta el final; si los combinas, `--from` recorta la lista. Cada nombre selecciona el segmento con ese nombre exacto, todos los segmentos de una `Section` con esa clave o el paso de una `Section` con ese nombre de `SectionStep` (por ejemplo `--sections "Problemática · país sísmico"`). Un nombre con comas se pasa solo o con sus comas escritas como `\,`, sin distinguir mayúsculas. Un nombre desconocido se muestra como error con las opciones disponibles, y entonces se reproduce todo.

El script se ejecuta completo: los objetos que persisten, la cámara y el tema llegan a la primera diapositiva elegida exactamente como en la ejecución entera, porque la selección **salta** a ese segmento en lugar de omitir código. La reproducción y la navegación solo visitan lo elegido; al terminar un tramo se salta al siguiente. Con `gaanim --diff ... --capture-stops` las mismas opciones limitan las pausas capturadas (ver Capturas y comparación visual).

21.5. La vista del presentador

Con `--present`, la audiencia ve la diapositiva a pantalla completa y tú ves **Presenter View**, la vista del presentador. Se lee de arriba abajo:

- **Encabezado:** estado (`PLAYING`, `PAUSED` o `END`), `Slide n of N`, la hora, el tiempo transcurrido con un botón ↺ para reiniciarlo y una barra con un bloque por diapositiva y marcas de pasos; un clic salta a ese punto. Si la charla tiene una duración prevista, también `LEFT` y `PACE` (ver El tiempo de la charla).
- **Now on screen:** nombre de la diapositiva, `Step k of K · nombre`, el tiempo que llevas en ella si tiene objetivo o un ensayo anterior, y una vista grande de lo que ve la audiencia, en vivo: con los votos y los jugadores de la sala cuando hay público. Si la pantalla de la audiencia está en negro o en blanco, la vista lo indica. Debajo, unos botones permiten saltar al inicio de la diapositiva o a cualquiera de sus pasos.
- **Up Next:** una miniatura de la siguiente pausa, indicando si es otro paso de la misma diapositiva o la siguiente diapositiva. Las miniaturas muestran el ensayo, no los votos de la sala.
- **Speaker notes:** las notas de la diapositiva cortadas en sus `[stop]`, con el tramo que toca decir resaltado y `▶ ADVANCE` donde avanzas. `A−`/`A+` ajustan el tamaño y **Prompter** muestra el tramo que toca decir como un subtítulo (ver El prompter).
- **Barra inferior:** primer paso, paso anterior, reproducir/pausa, paso siguiente, último paso, `Overview`, `Black` y `White`, el progreso de las vistas previas y la lista de atajos (icono de teclado).

21.5.1. El tiempo de la charla

Dile a Gaanim cuánto quieres que dure la charla, entera o por diapositivas, y Presenter View te dirá cuánto queda y si vas adelantado o retrasado:

```
scene.slides.target(20 * 60)                                # la charla, en segundos
scene.segment("Portada", template=title_slide, target=30)    # cada diapositiva
scene.segment("Resultados", Transition.slide(0.5, "left"), target=120)
```

La duración de la charla entera también se fija sin tocar el script: en **TARGET** del panel **Script & narration** (ver El panel Script & narration); en Presenter View, con el botón que la muestra junto a `LEFT` (o `Plan time` si aún no hay ninguna), donde eliges `10m` a `60m`, escribes otra (`25m`, `1h10m`, `25:00`) o la quitas; o desde la línea de comandos:

```
gaanim --present --duration 20m mi-charla
```

- `--duration` acepta `20m`, `1h30m`, `90s`, `20:00` o un número de minutos.
- Si hay varias, manda la que fijes en Presenter View o en el panel durante la sesión, luego `--duration`, luego la del script y por último la que fijaste la vez anterior para ese archivo, que Gaanim recuerda. Sin ninguna, la duración es la suma de los objetivos (`target=`), si todas las diapositivas tienen uno. El popup del botón dice de dónde viene.
- `LEFT` cuenta hacia atrás lo que queda de esa duración desde que empezó la presentación (el ↺ del tiempo transcurrido también la reinicia). Pasa a ámbar en el último 10 % y a rojo, con signo menos, cuando te pasas.

- `PACE` aparece si alguna diapositiva tiene objetivo. Suma lo que cada diapositiva ya dejada tardó de más o de menos, y lo que la actual ya se pasó: `+1:05` es un minuto y cinco segundos de retraso, `-0:20` veinte segundos de adelanto. Va en verde si vas a tiempo o adelantado, en ámbar con un retraso de hasta un 10 % de la duración prevista (30 segundos como mínimo) y en rojo con más.
- Bajo el nombre de la diapositiva, `0:45 of 1:30 on this slide` compara el tiempo que llevas en ella con su objetivo; en rojo, cuando lo superas.

Para ajustar la charla a esa duración antes de presentarla, grábala en el editor diapositiva a diapositiva (ver Grabar la charla y ajustar sus tiempos) o practícala contra ella (ver Practicar contra un objetivo).

Cada presentación que dura más de 30 segundos se guarda como un ensayo, con el tiempo de cada diapositiva, en la carpeta de datos de Gaanim. En el siguiente, Presenter View añade `last rehearsal 1:20`: lo que tardaste en esa diapositiva la vez anterior. Se guardan los 20 últimos ensayos de cada archivo, del script o del paquete `.gaanim`, que también conserva los objetivos.

21.5.2. El prompter

El prompter muestra lo que toca decir como un subtítulo: el tramo del guion entre un stop y el siguiente, sobre una franja opaca, con sus palabras iluminándose al ritmo al que deberías ir diciéndolas. Es el mismo en Presenter View y en el editor, con el mismo tamaño de texto.

- **En el editor:** `Shift+T` o **More controls → Prompter** · **Shift+T** lo abren abajo, sobre la vista previa y por encima de la barra de reproducción. No tiene barra ni botones: el clic derecho cambia el tamaño del texto, dice a qué ritmo lee y lo cierra. Mientras grabas o dictas, el mismo subtítulo muestra también la grabación (ver Grabar la charla).
- **En Presenter View: Prompter** (o la tecla `T`) convierte la columna de notas en el subtítulo, que ocupa toda la columna; `A+` y `A–` cambian el tamaño. La columna se ensancha y la miniatura de **Up Next** se reduce para dejarle sitio. **Notes** o `T` vuelven a las notas de la diapositiva.

Cómo se lee:

- Lo ya dicho se apaga y lo que viene queda en blanco. Cuando el tramo no cabe en dos líneas, el texto sube con suavidad para dejar a la vista la línea que toca.
- El avance sigue a la línea de tiempo: cuenta el trozo del tramo que ya recorrió el cabezal más el tiempo que la diapositiva lleva esperando en su stop. Si retrocedes, las palabras vuelven atrás con él.
- El ritmo sale de tus grabaciones: el tiempo que tardaste en ese tramo cuando grabaste la charla, si lo grabaste; si no, tu ritmo medido al dictar con Scribe; y si nunca dictaste, 150 palabras por minuto. Para leer siempre al mismo ritmo, elige **Fixed** en **Settings → Prompter** (`Ctrl+,`) y fija las palabras por minuto; ahí también se olvida el ritmo medido y se cambia el tamaño del texto, que Gaanim recuerda (ver Ajustes).
- A la derecha, `▶ grafico` dice dónde avanzas: el nombre del stop o, al final de la diapositiva, el de la siguiente. Late cuando la diapositiva ya espera en ese stop y el tramo está dicho: toca avanzar.
- Debajo, a la izquierda, el nombre de la diapositiva y, si la grabaste, su tiempo frente a su objetivo.

Para ensayar la charla entera contra su duración, presenta con el prompter abierto: Presenter View cuenta el tiempo de cada diapositiva y guarda el ensayo (ver El tiempo de la charla).

```
gaanim --present --duration 20m mi-charla
```

Para grabarte presentando, llama a `scene.live_take()` al principio del script y pulsa **Record live take** en **Script & narration**. El prompter del editor sigue la grabación, y la toma guarda cuánto esperaste en cada stop: la reproducción y el vídeo exportado siguen tu ritmo (ver Narración).

21.5.3. Atajos de teclado

Tecla	Acción
→, `Enter` o clic en la audiencia	avanzar al siguiente paso
← o `Backspace`	volver al paso anterior
`Space`	pausar o reanudar la animación en curso; nunca salta de paso
`Home` / `End`	ir al primer o al último paso
`O`	vista general: busca por nombre de diapositiva, paso o notas; `Enter` salta a la primera coincidencia
`T`	mostrar el tramo que toca decir como subtítulo en el prompter, o volver a las notas de la diapositiva; en el editor, `Shift+T` abre o cierra el subtítulo del prompter
`B` / `W`	poner la pantalla de la audiencia en negro o en blanco
`P`	volver a abrir la vista del presentador si la cerraste
`Esc`	cerrar la vista general o el negro/blanco; después, salir del modo presentación

Saltar a una diapositiva muestra su estado inicial aunque la anterior termine con una pausa en el mismo instante.

21.5.4. Pantalla de la audiencia

Al llevar el cursor a la parte inferior de la pantalla completa aparece una barra compacta con primer paso, paso anterior, reproducir/pausa, paso siguiente, último paso, el nombre y número de la diapositiva y el progreso. Se oculta al retirar el cursor o al pasar a la vista del presentador. Sus botones y los atajos hacen lo mismo, así que un clic no avanza dos veces.

La pantalla completa, tanto en el editor como al presentar, ajusta el lienzo al monitor sin deformarlo y rellena en negro las franjas sobrantes, sea cual sea el fondo de la escena. En el editor, `Esc` o `F11` salen de la pantalla completa.

21.5.5. Vistas previas de las diapositivas

Las vistas previas se generan en segundo plano sin afectar a la presentación: primero la diapositiva actual y la siguiente, luego el resto. Tras una recarga en caliente, las anteriores siguen visibles, marcadas como `Updating preview…`, hasta que llega su reemplazo. Se conservan al reabrir la vista del presentador, se adaptan al tamaño y la densidad de píxeles de la ventana y solo se regeneran al agrandarla. Si el renderizado falla, la barra ofrece `Retry` y conserva las vistas ya generadas.

Las vistas previas incluyen el contenido 3D, igual que la pantalla de la audiencia.

21.6. Encuestas a la audiencia

Una presentación puede hacer preguntas al público: encuestas, cuestionarios al estilo de Kahoot con clasificación, juegos por equipos y personajes del público dentro de la escena. Todo eso está en Público en vivo: empieza por Tu primera encuesta.

21.7. Repartir las diapositivas en PDF

Para repartir la presentación o tener una copia que no necesite Gaanim, expórtala a PDF: una página por cada pausa, tal como la muestra el Presenter, y otra para el final. Con `--notes`, tras cada

diapositiva va una página con sus notas, donde cada `[stop]` se imprime como una flecha (→ `grafico`).

```
gaanim export mi-charla --output mi-charla.pdf --notes
```

También funciona con un paquete `.gaanim` y desde el diálogo **Export** del editor; la referencia de la línea de comandos explica las opciones.

21.8. Revisar sin pausas

Para revisar una animación de corrido en el editor, activa **Continuous** junto a los controles de reproducción. El ajuste dura toda la sesión y sobrevive a las recargas, pero el modo presentación sigue respetando `scene.stop(...)`. Los saltos, las capturas y la exportación a vídeo también ignoran las pausas.

CAPÍTULO

22. Ensayar y grabar una charla

Del guion a una charla que cabe en su tiempo: practica contra un objetivo, graba tus tiempos y tu voz, y conviértelo en vídeo o en presentación

En esta guía preparas una charla corta de principio a fin con el panel **Script & narration** del editor: le pones una duración, la practicas diapositiva a diapositiva hasta que cabe, grabas tus tiempos y, al final, la conviertes en un vídeo con tu voz o la presentas en directo. Todo con un proyecto de ejemplo que puedes copiar.

El flujo es un ciclo:

1. **Escribir** lo que vas a decir en el guion.
2. **Practicar** contra un objetivo, sin grabar nada.
3. **Recortar** donde te pasas: el texto, o el objetivo de esa diapositiva.
4. **Grabar** tus tiempos (y, si quieres, tu voz para escucharte).
5. **Producir**: un vídeo con la voz grabada, o la presentación en directo.

22.1. El proyecto de ejemplo

Una charla de cinco minutos sobre el ciclo del agua: una portada y cuatro ideas, cada una con dos stops (el título y la idea). Crea una carpeta con estos tres archivos, o empieza con `gaanim init` y sustituye su `main.py`.

`gaanim.toml`:

```
name = "ciclo-del-agua"
kind = "slides"
entry = "main.py"
```

`main.py`:

```
from gaanim import Scene, lecture, title_slide

scene = Scene(frame=(16, 9), margin=0.9)
scene.canvas.set_theme("presentation")
scene.slides.brand(footer="EL CICLO DEL AGUA", slide_numbers=True, rule=True)
scene.slides.target(5 * 60)                    # la charla entera: 5 minutos


def idea(name, heading_text, lines, target=None):
    """Una diapositiva con un título y una línea por stop."""
    slide = scene.segment(name, template=lecture, target=target)
    heading = scene.text(heading_text, role="title")
    body = scene.text("\n".join(lines), size=0.42)
    slide.bind(title=heading, body=body)
    scene.play([heading.animate.fade_in().duration(0.4)])
    scene.stop("titulo")
    scene.play([body.animate.write().duration(1.2)])
    scene.stop("idea")


cover = scene.segment("Portada", template=title_slide, target=20)
```

```
title = scene.text("El ciclo del agua", role="title")
subtitle = scene.text("Cinco minutos, cuatro ideas", role="subtitle")
cover.bind(title=title, subtitle=subtitle)
scene.play([title.animate.write().duration(0.8), subtitle.animate.fade_in().duration(0.6)])
scene.stop("portada")

idea("Evaporacion", "Evaporación", ["El sol calienta mares y lagos.", "El agua sube como vapor."])
idea("Condensacion", "Condensación", ["El vapor se enfría al subir.", "Se forman las nubes."])
idea("Precipitacion", "Precipitación", ["Las gotas crecen y pesan.", "Cae lluvia, nieve o granizo."])
idea("Cierre", "Y vuelta a empezar", ["El agua llega a ríos y mares.", "El ciclo no se detiene."],
target=30)

scene.render()
```

`assets/narration/script.md`: lo que dices en cada diapositiva, con una línea `[stop nombre]` donde avanzas (ver El guion de la charla).

```
# El ciclo del agua

## Portada
Buenas tardes. En cinco minutos les cuento cómo el agua da la vuelta al
planeta una y otra vez.
[stop portada]

## Evaporacion
Todo empieza con el sol.
[stop titulo]
El sol calienta los mares, los lagos y hasta los charcos, y una parte del agua
sube al aire convertida en vapor, sin que la veamos.
[stop idea]

## Condensacion
Ese vapor no se queda arriba para siempre.
[stop titulo]
Al subir, el aire se enfría; el vapor se condensa en gotas diminutas que,
juntas, forman las nubes que vemos pasar.
[stop idea]

## Precipitacion
Las nubes, entonces, empiezan a pesar.
[stop titulo]
Las gotas chocan, se unen y crecen hasta que el aire ya no las sostiene, y
caen como lluvia, nieve o granizo según la temperatura.
[stop idea]

## Cierre
Y aquí el ciclo se cierra.
[stop titulo]
El agua corre por ríos y acuíferos hasta el mar, y el sol vuelve a empezar.
Gracias.
[stop idea]
```

No hace falta escribir el guion a mano: con el panel abierto, el icono del libro de su cabecera lo crea con una sección por diapositiva y sus marcas, y lo abre en tu editor. `gaanim check ciclo-del-agua` confirma que guion y diapositivas coinciden (`Ready to present`).

22.2. Abrir el panel

Abre el proyecto en el editor (`gaanim ciclo-del-agua`, o desde Home) y pulsa el micrófono de la barra de reproducción, o busca **Script & narration** en la paleta de comandos (`Ctrl+Shift+P`). El panel se acopla a la derecha y la escena se encoge a su lado.

Figura 10: El panel con las cinco diapositivas grabadas y un objetivo de 1:00: tres diapositivas se pasan de su parte (en ámbar).

De arriba abajo: el objetivo, cómo van las tomas y la última práctica, el mapa de la charla, el guion de la diapositiva en pantalla, la lista de diapositivas y, abajo, los controles para grabar o practicar. Cada parte se explica en El panel Script & narration.

22.3. Fijar el objetivo

El objetivo es lo que quieres que dure la charla. El ejemplo lo pone en el código con `scene.slides.target(5 * 60)`, y el panel lo muestra en **TARGET** con `from scene.slides.target`. Para probar otra duración sin tocar el código, escríbela en el campo (`4:00`, `4m`, `1h10m`) y pulsa `Enter`: pasa a `set here`, y Presenter View planifica con ella también. Vaciarla vuelve a la del código.

Con el objetivo, cada diapositiva recibe su parte (**Share**). La portada y el cierre tienen `target=` propios (20 y 30 segundos); lo que queda se reparte entre las otras tres según cuánto dice su guion. La tarjeta **SCRIPT** avisa si el texto de una diapositiva, leído a tu ritmo, no cabe en su parte (`Script 0:08 too long`): antes de practicar ya sabes dónde sobra texto.

22.4. Practicar

Practicar es presentar la charla cronometrada sin grabar nada. En los controles de abajo, elige **Practice** y pulsa **Practise from the start** (o **From here**, para empezar en la diapositiva en pantalla). Tras una cuenta atrás de tres segundos, presenta en voz alta:

Tecla	Qué hace
→ o `Espacio`	Avanza al siguiente stop.
`Enter`	Termina y guarda la práctica.
`Esc`	La descarta.

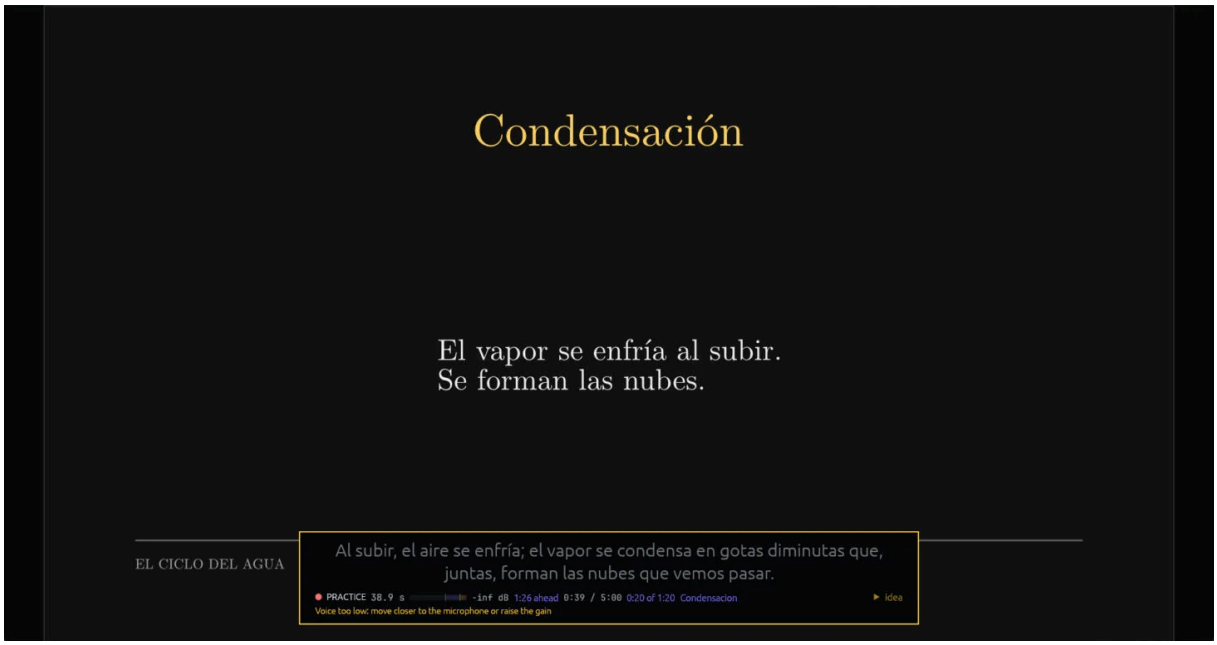


Figura 11: Practicando: el prompter lee el guion y la línea de estado dice cómo vas.

El prompter muestra el tramo del guion que toca decir, con las palabras iluminándose a tu ritmo, y debajo:

- `PRACTICE` y el tiempo desde que empezaste, con el nivel del micrófono.
- `1:26 ahead` o `0:12 behind`: cuánto vas adelantado o retrasado frente a las partes del objetivo. Las diapositivas ya dejadas cuentan enteras y la actual solo cuando se pasa de la suya.
- `0:39 / 5:00`: el reloj frente al objetivo.
- `0:20 of 1:20 Condensacion`: el tiempo en la diapositiva frente a su parte.
- A la derecha, `▸ idea`: el stop donde toca avanzar.

22.5. Leer el resultado

Al terminar, el panel lo resume (`Practice: 1:07 of 5:00 · 3:53 to spare`), **LAST PRACTICE** da el total frente al objetivo y **PRACTICE RUNS** añade una columna por práctica, con una línea dorada en el objetivo: de un vistazo ves si cada ensayo se acerca.

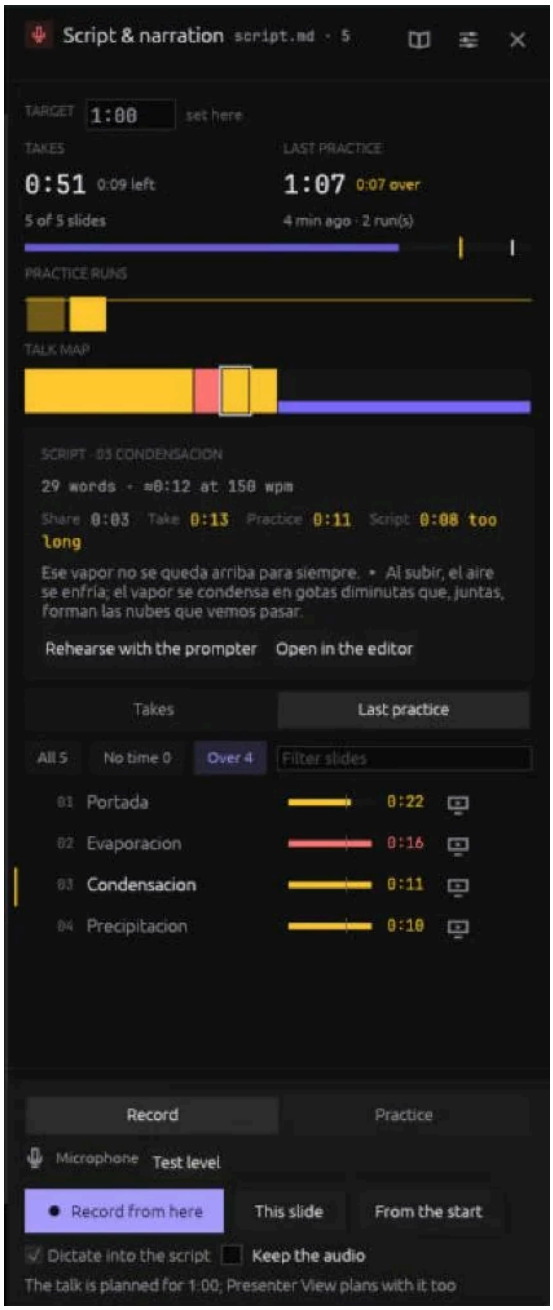


Figura 12: La última práctica, filtrada por las diapositivas que se pasan.

Pon la lista en **Last practice** y pulsa **Over**: quedan solo las diapositivas que se pasaron de su parte, en ámbar si se pasan un poco (un 10 %, al menos 10 segundos) y en rojo si se pasan más. Para cada una tienes dos salidas:

- **Decir menos**: **Open in the editor** abre el guion en su sección; recorta el texto y guarda (el editor actualiza las notas al momento).
- **Darle más tiempo**: súbele el `target=` en el código, y las demás diapositivas se reparten lo que queda.

El botón de cada fila practica desde esa diapositiva, para repetir solo el tramo que corregiste. Repite hasta que **LAST PRACTICE** quede por debajo del objetivo.

22.6. Grabar los tiempos y la voz

Cuando la charla cabe, grábala. En los controles, elige **Record**:

- **Test level** abre el micrófono y muestra su nivel sin grabar: habla como en la charla y busca que los picos caigan en la franja violeta del medidor, entre −18 y −6 dB.
- **From the start** presenta desde la primera diapositiva y **Record from here** desde la que está en pantalla. Cada diapositiva que recorres entera guarda su tiempo y el de cada tramo entre stops.
- **This slide** vuelve a grabar solo la diapositiva en pantalla; el botón de grabar de un stop, solo ese tramo.
- **Dictate into the script** escribe en el guion lo que dices, con Scribe (ver Dictar el guion con Scribe).
- **Keep the audio** guarda tu voz de cada tramo para escucharla con el botón de reproducir de cada fila.

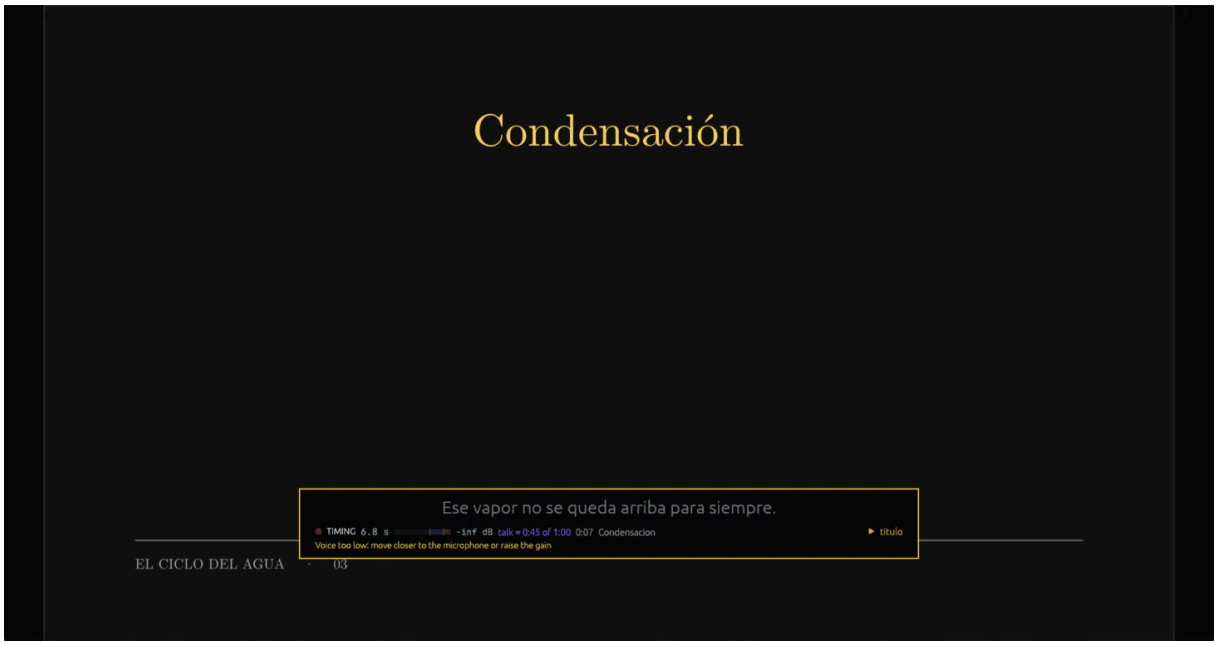


Figura 13: Grabando: `talk ≈ 0:45 of 1:00` estima lo que duraría la charla con lo grabado hasta ahora.

Al grabar, la línea de estado dice `TIMING` y `talk ≈ 0:45 of 1:00`: lo que duraría la charla con esta grabación y los tiempos guardados del resto. Si el micrófono suena bajo o satura, el prompter se enmarca y lo avisa.

Los tiempos y el audio de **Keep the audio** son para ti: se guardan en la carpeta de datos de Gaanim, un archivo WAV por tramo, nunca en el proyecto, y no acaban en una exportación.

22.7. Del ensayo al vídeo

Para que tu voz forme parte del vídeo, grábala como una toma en vivo: añade `scene.live_take("charla")` antes de la primera diapositiva.

```
scene.live_take("charla")
cover = scene.segment("Portada", template=title_slide, target=20)
scene.stop("portada")
```

Con la escena recargada, el panel muestra al final la tarjeta **LIVE TAKE**. **Record live take** presenta la escena desde el principio mientras grabas, igual que en la práctica: → o `Espacio` avanzan, `Enter` termina. La toma se guarda en `narration/charla.wav` del proyecto y cada stop pasa a durar lo que tardaste en él, así que la escena se reproduce de corrido con tu voz. Después exporta:

```
gaanim export ciclo-del-agua -o ciclo-del-agua.mp4
```

Practica antes de grabar la toma: una toma en vivo se graba siempre entera, y el ciclo de practicar y recortar es más rápido que repetirla. Si prefieres grabar bloque a bloque, usa voz en off (ver Narración).

22.8. Del ensayo a la presentación

Para presentar en directo, el objetivo y las prácticas te acompañan:

```
gaanim --present ciclo-del-agua
```

Presenter View (ver El tiempo de la charla) cuenta hacia atrás el objetivo y dice si vas adelantado o retrasado, con el mismo plan que fijaste en el panel. Cada presentación de más de 30 segundos cuenta como una práctica más, y aparece en **PRACTICE RUNS** la próxima vez que abras el panel. Para repartir la charla, `gaanim export ciclo-del-agua -o charla.gaanim` la guarda en un solo archivo que se presenta sin Python, y `-o charla.pdf --notes` da las diapositivas con tu guion (ver Compartir).

CAPÍTULO

23. Proyectos y exportación

Organiza escenas, recursos y salidas en un proyecto; compruébalo y expórtalo

En esta guía aprenderás a trabajar con un proyecto de Gaanim de principio a fin: qué contiene, cómo carga sus recursos, qué pasa cuando guardas un archivo, cómo comprobarlo sin abrir ventana y cómo exportarlo al formato que necesites.

Si todavía no tienes Gaanim instalado ni un proyecto creado, empieza por Instalación: allí están `gaanim init`, la estructura que genera y cómo se prepara el entorno de Python.

23.1. Por qué un proyecto

Gaanim puede abrir un script suelto (`gaanim escena.py`), pero un proyecto mantiene juntos el código, los recursos y las exportaciones, y se abre igual desde cualquier directorio. Hay dos tipos:

- `video`: una escena animada 16:9 pensada para exportarse.
- `slides`: segmentos con nombre, notas y paradas para presentar en vivo (ver Presentaciones).

`gaanim` sin argumentos abre el Inicio, desde donde puedes crear un proyecto, abrir una carpeta o volver a uno de los diez proyectos recientes. Los scripts sueltos no se añaden a esa lista. Los botones junto a **Clear list** cambian entre la lista y una cuadrícula de tarjetas con la imagen de cada proyecto, y Gaanim recuerda la elección. La imagen de un proyecto la guarda el editor al abrirlo: el fotograma de `scene.thumbnail(t)` si el script lo elige, si no el de la primera pausa, y sin pausas el más lleno de unos pocos instantes del primer segmento, el que menos fondo vacío deja ver. Se renueva como mucho cada cinco minutos mientras editas y se guarda en la carpeta de datos de Gaanim, no en el proyecto. Un paquete `.gaanim` muestra su portada.

Un paquete exportado dentro de la carpeta de un proyecto reciente no aparece aparte: va con su proyecto, que dice hace cuánto se grabó el último y lo reproduce con su botón, y presta su portada al proyecto mientras el editor no le haya guardado una. El buscador junto a **Clear list** filtra por nombre o ruta. La chincheta fija un proyecto arriba de la lista: los fijados no salen de ella por antigüedad ni con **Clear list**, solo al quitarlos. Para volver al Inicio desde el editor, usa el botón de la casa de la barra de reproducción.

23.2. El manifiesto `gaanim.toml`

Cada proyecto tiene un `gaanim.toml` en su raíz:

```toml
name = "mi-charla"
kind = "slides"
entry = "main.py"
assets_dir = "assets"
output_dir = "exports"
```

- `entry` es el script que se ejecuta. Debe ser una ruta relativa que quede dentro del proyecto.
- `kind` solo acepta `video` o `slides`.

- `assets_dir` es la carpeta de recursos que carga la escena.
- `output_dir` es la carpeta sugerida para las exportaciones.

La referencia de `gaanim.toml` describe cada clave.

`--force` en `gaanim init` actualiza los archivos conocidos del proyecto sin borrar tus recursos ni otras carpetas.

23.3. Cargar los recursos del proyecto

Para que las rutas de imágenes, SVG, audio o modelos no dependan del directorio desde el que se abre Gaanim, carga el manifiesto al principio de la escena. Después, las rutas se resuelven dentro de `assets_dir`:

```
from gaanim import Scene

scene = Scene(frame=(16, 9))
scene.assets.load_project()  # busca gaanim.toml desde la carpeta del script hacia arriba
logo = scene.media.svg("logo.svg")  # busca assets/logo.svg
scene.play([logo.animate.fade_in().duration(0.5)])
scene.render()
```

Si no usas manifiesto, `scene.assets.assets_dir("assets")` fija la carpeta de recursos directamente. Consulta Recursos para los formatos admitidos.

23.4. Abrir, comprobar y presentar

Puedes pasar la carpeta del proyecto en lugar de la ruta de `main.py`:

```
gaanim mi-charla                    # vista previa con recarga al guardar
gaanim check mi-charla              # ejecuta la escena sin ventana
gaanim check mi-charla --strict     # falla también con avisos
gaanim --present --monitor 1 mi-charla
```

Dentro del proyecto, `.` es la carpeta actual: `gaanim .` o `gaanim check .`.

`gaanim check` ejecuta el script completo, calcula la duración y revisa los segmentos, las notas y las paradas si las hay; en proyectos `slides` también comprueba el formato 16:9 y los marcadores de plantilla sin rellenar. Úsalo antes de exportar o en integración continua.

23.5. Qué pasa al guardar

La vista previa vuelve a ejecutar la escena cada vez que guardas.

23.5.1. Recarga incremental

Los segmentos anteriores al primero que cambió conservan su escena compilada y solo se recompila desde ahí. El aviso de recarga lo indica, por ejemplo `replay 0.12s (reused 37/40 segments)`, y `seek` dice cuánto tardó en volver a la posición donde estabas. La primera edición en una zona nueva recompila todo una vez; las siguientes en esa zona ya son incrementales.

Si nada de lo que se compila cambió, por ejemplo al guardar sin cambios o al editar solo un comentario, la escena se conserva tal cual y el aviso dice `(unchanged, 40 segments)`. Volver a la posición de trabajo cuesta entonces lo mismo que un salto cualquiera.

Se recompila todo cuando:

- cambia algo global (tema, fuentes, tamaño del lienzo);
- un segmento posterior modifica objetos creados antes, por ejemplo al agrupar un logo que aparece desde el principio;
- desde cualquier segmento que use funciones de Python en tiempo de ejecución (animaciones propias, updaters, funciones reactivas o easings propios).

El resultado siempre es idéntico al de una recarga completa. Para forzar esta última, inicia Gaanim con la variable de entorno `GAANIM_INCREMENTAL=0`.

23.5.2. Recursos

Guardar un archivo que no sea Python dentro del proyecto (imágenes, SVG, Lottie, WGSL, fuentes, documentos Typst, datos) vuelve a ejecutar el script leyendo de nuevo ese archivo, no el resto: las cachés olvidan solo lo que guardaban de él. La recarga sigue siendo incremental y recompila desde el primer segmento que lo usa; si la escena no lo lee, queda igual (`unchanged`).

Se ignoran los archivos y carpetas ocultos (`.git`, `.venv`, archivos de intercambio del editor), `venv`, `env`, `__pycache__`, `exports`, `snapshots`, `target`, los temporales (`~`, `.swp`, `.tmp`, `.bak`) y los `.lock`. Un guardado cuenta por el archivo que guarda: las herramientas que escriben un temporal y lo renombran sobre el archivo (como `main.py.tmp.*` o el `sedXXXXXX` de `sed -i`) y la carpeta, que Windows también da por cambiada, no cuentan como cambios.

23.5.3. Módulos de Python del proyecto

Puedes repartir el código en varios módulos. Antes de cada ejecución, Gaanim descarta de `sys.modules` los módulos cuyo archivo está dentro del proyecto (salvo en las carpetas ignoradas), así que guardar un `.py` importado por `main.py` lo vuelve a importar. Los paquetes instalados y el código fuera del proyecto siguen en caché hasta reiniciar Gaanim.

La raíz del proyecto, su carpeta `src/` y la carpeta del script están en `sys.path`, y `__file__`, `sys.argv[0]` y `sys.path` usan rutas normales, sin el prefijo `\\?\` de Windows.

23.6. Exportar

Para exportar no cambias el script: se lo pides al ejecutable.

```
gaanim export mi-video --output mi-video/exports/final.mp4
```

La extensión de `--output` elige el formato:

Extensión	Formato	Necesita FFmpeg
`.mp4`	video H.264, con el audio de la escena	sí
`.webm`	video VP9, con audio; admite transparencia	sí
`.webp`	WebP animado; admite transparencia	sí
`.gif`	GIF animado	sí
`.png`	secuencia de imágenes, una por fotograma; admite transparencia	no

En una secuencia PNG, un `%d` o `%0Nd` en el nombre se sustituye por el número de fotograma (`frames/f_%04d.png` produce `f_0000.png`, `f_0001.png`…); si no, el número se añade al final (`frame_00000.png`).

23.6.1. Opciones

- `--quality draft` exporta rápido a 30 fps para revisar; `standard` (la predeterminada) usa 60 fps y `production`, la mejor codificación.
- `--fps` elige otra frecuencia: `24`, `25`, `50` o una fracción NTSC como `24000/1001`.
- `--width` y `--height` eligen la resolución en píxeles (1920 × 1080 por defecto). La escena no cambia: sus unidades lógicas se escalan.
- `--fit contain` o `--fit cover` permiten una salida con otra proporción que la escena: `contain` añade bandas y `cover` recorta. Sin `--fit`, una proporción distinta es un error.
- `--from` y `--to` exportan solo un tramo. Aceptan segundos o el nombre de un marcador creado con `scene.marker("nombre")`; el audio se recorta al tramo.
- `--transparent` conserva el canal alfa en WebM, MOV, WebP y PNG; la escena necesita un fondo transparente. Un `.mov` es Apple ProRes, para montar la animación en un editor de vídeo.
- `--encoder` elige el codificador de MP4: `auto` (predeterminado) prueba los de hardware y recurre a `libx264`; también acepta `libx264`, `nvenc`, `amf`, `qsv` y `vaapi`, que no recurren a otro si fallan.

`gaanim export --help` y la referencia de la línea de comandos listan todas las opciones. Las paradas (`scene.stop(...)`) no detienen la exportación: el resultado es un video continuo.

23.6.2. Desde el editor

El botón **Export** de la barra de reproducción abre las mismas opciones. Al pulsar **Export**, el diálogo se cierra y la exportación sigue en segundo plano: puedes seguir reproduciendo y navegando la escena. La barra de tiempo marca en violeta, bajo cada segmento, la parte ya exportada; la barra de reproducción muestra el porcentaje (pasa el cursor para ver fotogramas y tiempo restante) y una X para cancelar. La terminal desde la que abriste `gaanim` muestra la misma barra de progreso que `gaanim export` y el resultado. Al terminar aparece un aviso en la esquina con el archivo y un botón para abrirlo.

En **File** escribes solo el nombre: la extensión la pone el formato. Al principio es el del proyecto (su carpeta, si el script es `main.py`, o el nombre del script), en la carpeta `output_dir` de `gaanim.toml`. **Change…** elige otra carpeta con el selector del sistema; pasa el cursor por ella para ver la ruta completa. Si ya existe un archivo con ese nombre, el diálogo avisa de que lo reemplazará y ofrece el siguiente nombre libre (`intro-2`, `intro-3`…). Sin nombre no se puede exportar.

El formato **PNG** exporta un PNG por fotograma (`frame_00000.png`, `frame_00001.png`…) para editar o componer en otro programa. Los fotogramas van a una carpeta con el nombre que escribas, y el aviso final abre esa carpeta.

Para guardar solo el fotograma que estás viendo, pulsa `Ctrl+Mayús+S` (`Cmd+Mayús+S` en macOS) o el botón de captura de la barra de overlays. Se renderiza aparte, sin overlays ni paneles, a 1920 píxeles en el lado largo, y se guarda en `<output_dir>/captures/` con el nombre del script y el instante (`main_2.35s.png`); nunca reemplaza una captura anterior.

23.7. Convertir un script suelto en proyecto

Crea un proyecto con `gaanim init video mi-video`, reemplaza su `main.py` por tu script, añade `scene.assets.load_project()` justo después de crear la `Scene` y mueve los recursos a `assets/`. Funciona igual desde `main.py` que desde un módulo de un paquete del proyecto, como `capitulo4/estilo.py`: busca el `gaanim.toml` más cercano subiendo de carpeta.

Si algo falla al abrir o exportar, consulta Solución de problemas.

CAPÍTULO

24. Compartir sin Python

Graba una escena o presentación en un solo archivo .gaanim que se reproduce, presenta y exporta sin Python

En esta guía aprenderás a grabar una escena o una presentación en un solo archivo `.gaanim` (un *paquete de reproducción*) y a reproducirlo, presentarlo o convertirlo en vídeo en otro equipo que no tiene Python, ni tu proyecto, ni tus recursos.

24.1. Qué guarda un paquete

Un paquete no guarda tu script: guarda lo que la escena dibujó. Gaanim ejecuta la escena igual que para exportar un vídeo y, en cada fotograma, registra lo que el renderizador 2D compone (trazados, pinceles, degradados, imágenes, efectos, máscaras, transiciones, cámara y posprocesado) en lugar de píxeles. Junto a los fotogramas guarda la estructura de la escena: segmentos, notas del orador, pausas con nombre, bucles ambientales, marcadores, escenas, el fondo y el audio.

Por eso cualquier código que la escena ejecuta mientras se reproduce queda *horneado* en el paquete: updaters, animaciones personalizadas, funciones reactivas, `always_redraw`, números aleatorios y cualquier otro callback de Python o de Rust. Al reproducir no se ejecuta nada de tu código; solo se componen los fotogramas grabados. Las animaciones Lottie y dotLottie también se graban así: cada fotograma distinto que dibujan se guarda como escena vectorial, así que el paquete no necesita el archivo Lottie ni sus imágenes.

24.2. Grabar

La extensión `.gaanim` en `--output` graba un paquete:

```
gaanim export mi-charla --output mi-charla.gaanim
gaanim export mi-video --output mi-video.gaanim --fps 30
```

`--fps` elige cuántos fotogramas por segundo se graban (60 por defecto). Un vídeo exportado desde el paquete tiene esa misma frecuencia, así que elige la del vídeo que quieras obtener después. `--from`, `--to`, `--transparent` y `--encoder` no se aplican a un paquete: siempre se graba la escena completa.

Grabar no rasteriza nada, así que suele ser rápido: una presentación de un minuto tarda unos segundos. Una presentación larga con decenas de miles de objetos tarda más, porque cada fotograma recorre la escena completa; una presentación de 41 segmentos, 29 000 objetos y 4½ minutos se graba en unos 8 minutos y ocupa 18 MB. Los paquetes de los ejemplos de Gaanim ocupan entre 20 y 500 KB; una escena con mil instancias animadas, unos 11 MB.

24.3. Reproducir y presentar

```
gaanim mi-charla.gaanim                          # reproducir
gaanim --present --monitor 1 mi-charla.gaanim    # presentar
gaanim mi-charla.gaanim --from resultados        # ensayar desde un segmento
```

Se abre el mismo editor, con la misma barra de reproducción, navegación entre pausas y segmentos, bucles, Presenter View con notas y miniaturas, vista general y atajos de teclado (ver Presentaciones). Lo que no tiene sentido sin el script no está: la recarga al guardar y la narración.

También puedes abrirlo desde el inicio de Gaanim (`gaanim` sin argumentos): la tarjeta **Play .gaanim** (`Ctrl Shift O`) elige el archivo, y también puedes arrastrarlo a la ventana. Los archivos que abres aparecen en **Recent** junto a tus proyectos. En un equipo sin Python, el inicio se abre igual y reproduce archivos `.gaanim`; para crear o abrir proyectos te indica cómo instalarlo. Sin FFmpeg, el paquete se reproduce sin audio y se exporta solo a secuencias PNG.

`gaanim` no carga Python para reproducir, presentar, exportar ni comprobar un paquete: solo lo carga al ejecutar un script. Por eso basta con copiar el paquete de Gaanim de la release, sin instalar Python.

La inspección (`I`) sigue funcionando: los fotogramas son vectoriales, así que el zoom los redibuja nítidos.

24.4. Abrir con doble clic

`gaanim register` asocia los archivos `.gaanim` con Gaanim para tu usuario, sin permisos de administrador:

- el doble clic reproduce el archivo;
- el menú contextual ofrece **Present**;
- el explorador de archivos muestra su portada.

El botón **Associate .gaanim files** de la sección *Environment* del inicio hace lo mismo, y `gaanim unregister` lo deshace. La asociación apunta al `gaanim` desde el que la creas: si mueves la carpeta de Gaanim, vuelve a registrarla.

- **Windows**: escribe en `HKCU\Software\Classes`. Si antes elegiste otra aplicación para los `.gaanim`, Windows pregunta una vez con cuál abrirlos. La portada la dibuja `gaanim_thumbnail_handler.dll`, que viene en el zip junto a `gaanim.exe`; debe quedarse en esa carpeta.
- **Linux**: instala en `~/.local/share` el tipo MIME `application/x-gaanim`, la entrada `gaanim.desktop`, un miniaturizador y los iconos. Las portadas se ven en Nautilus, Nemo, Caja y Thunar (con tumbler); Dolphin (KDE) todavía no.

24.4.1. Portada

Al grabar, Gaanim guarda en el paquete una portada de 512 píxeles de ancho: el fotograma que se ve en la primera pausa (`scene.stop()`), porque el primer fotograma suele estar vacío, o, sin pausas, el fotograma del primer segmento que más muestra. Para elegir otro, llama a `scene.thumbnail()` justo después del plano que quieres, o pásale un instante de la línea de tiempo:

```
scene.play(diagrama.animate.create())
scene.thumbnail()        # la portada es este plano
scene.thumbnail(0.5)     # o el fotograma de los 0,5 s
```

La portada se dibuja con la GPU, como una exportación; sin GPU el paquete se graba igual, sin portada, y el explorador muestra el icono de los `.gaanim`: una hoja con un video animado vectorial y el logo de Gaanim. Para extraerla, sin GPU:

```
gaanim thumbnail mi-charla.gaanim portada.png       # 512 px
gaanim thumbnail mi-charla.gaanim portada.png 256   # lado mayor de 256 px
```

24.5. En el navegador y el teléfono

El reproductor web abre un `.gaanim` sin instalar nada, en la PC o en el teléfono: paololupo.github.io/gaanim/reproductor. Toca o arrastra el archivo; no sale de tu equipo. Es el mismo reproductor de escritorio compilado para la web, con la misma barra de reproducción.

Es experimental: necesita un navegador con WebGPU (Chrome o Edge recientes, Safari 26, Firefox en Windows) y descarga unos 25 MB la primera vez. En las miniaturas del Presenter View web no se ven los postprocesos ni los fondos animados por shader.

El audio del paquete suena en la página de la presentación (el Presenter View queda en silencio). El navegador solo deja empezar el sonido tras un clic o una tecla en la página: si el paquete tiene audio, un aviso lo recuerda. Un audio con `speed` distinto de 1 cambia de tono, cosa que en el escritorio no pasa.

24.5.1. La barra de reproducción

- **Ubicar un momento**: al pasar el cursor por la línea de tiempo, una tarjeta muestra una miniatura de ese momento, el nombre del segmento, su número (`12 / 61`), la pausa y el tiempo. Las miniaturas se preparan la primera vez que pasas el cursor y son las mismas del Presenter View.
- **Buscar un segmento**: haz clic en el nombre del segmento actual, junto al tiempo. Se abre la lista de todos, con su número, sus pausas y cuándo empiezan; escribe parte de un nombre o un número y pulsa `Enter`, o haz clic en uno, para ir a él. Las flechas `↑` `↓` y `Tab` (`Shift+Tab` hacia atrás) recorren los segmentos de la lista, `Enter` va al marcado y `Esc` la cierra. La barra sigue visible mientras la lista, o cualquier menú suyo, esté abierta.
- **Volumen**: el altavoz silencia y el deslizador regula el volumen; solo aparecen si la escena tiene audio. El volumen se recuerda en la próxima visita. En barras estrechas están en el menú **More controls**.

Lo mismo vale para la barra del editor de escritorio.

24.5.2. Compartir un enlace

Para compartir un enlace, publica el `.gaanim` en un sitio que permita abrirlo desde otra página y añade su dirección con `?src=`:

```
https://paololupo.github.io/gaanim/reproductor/?src=https://tu-sitio.com/mi-charla.gaanim
```

El archivo va directamente del sitio donde lo publicaste al navegador de quien abre el enlace: no pasa por ningún servidor de Gaanim, y la página no registra qué archivos se abren. Un `.gaanim` son solo datos, así que abrirlo no ejecuta código de su autor.

Si el sitio lo permite, el reproductor no espera a la descarga completa: primero descarga el final del archivo, donde están el índice y las tablas, y empieza a reproducir; los fotogramas llegan a medida que hacen falta y el resto se descarga en segundo plano (la línea verde de arriba muestra cuánto). Si saltas a una parte que todavía no llegó, verás **Cargando…** un momento. Si el sitio no admite descargas por partes, se descarga el archivo entero antes de empezar, con una barra de progreso.

Alojamiento	¿Funciona?
GitHub Pages	Sí, también por partes.
`raw.githubusercontent.com` (un archivo de un repositorio de GitHub)	Sí, también por partes. GitHub no admite archivos de más de 100 MB en un repositorio.
Tu propio sitio o un CDN	Sí, si envía la cabecera `Access-Control-Allow-Origin` (CORS); por partes si además admite `Range`.
Descargas de GitHub Releases	No: redirigen a otra dirección sin CORS.
Google Drive, Dropbox, OneDrive	Por lo general no: sus enlaces de descarga pasan por páginas intermedias o no envían CORS.

Cuando no se puede abrir, el reproductor explica el motivo: que el sitio no permite abrirlo desde otra página (CORS), que la dirección no existe, que no hay conexión, que no es un `.gaanim` o está dañado, o que se grabó con un formato más nuevo que el del reproductor (recarga la página para actualizarlo).

Para publicarlo en GitHub Pages:

1. Crea un repositorio (o usa uno existente) y sube el `.gaanim`, por ejemplo a `charlas/mi-charla.gaanim`.
2. En **Settings → Pages**, elige publicar desde la rama principal.
3. Al cabo de un minuto, el archivo está en `https://<usuario>.github.io/<repositorio>/charlas/mi-charla.gaanim`.
4. El enlace es `https://paololupo.github.io/gaanim/reproductor/?src=https://<usuario>.github.io/<repositorio>/charlas/mi-charla.gaanim`.

24.5.2.1. Enlazar un momento

Como en los sitios de vídeo, lo que va después de `#` elige dónde empieza:

Fragmento	Empieza
`#segmento=5`	Al principio del quinto segmento.
`#segmento=resultados`	Al principio del segmento que se llama así (sin distinguir mayúsculas). Codifica los espacios como `%20`.
`#segmento=resultados&pausa=2`	En la segunda pausa de ese segmento, detenido ahí.
`#pausa=12`	En la duodécima pausa de toda la presentación.
`#t=83.5` o `#t=1:23.5`	A los 83,5 segundos.

`#present`	Muestra el botón **Present**, que abre el Presenter View; se combina con los demás: `#segmento=5&present`.

El botón de enlace de la barra (dos eslabones) copia el enlace al archivo abierto. **Copy link to this moment** añade el fragmento de lo que se ve: la pausa en la que está detenida la presentación o, si no, el segundo actual. Solo los archivos abiertos con `?src=` tienen enlace; uno que abriste desde tu equipo hay que publicarlo antes.

El botón **Present** existe porque el navegador solo deja abrir otra ventana tras un clic: el enlace no puede abrir el Presenter View por sí solo.

24.5.3. Insertar en un blog o sitio web

Una animación corta puede ir en una entrada de blog como una figura más: se ve nítida a cualquier tamaño y suele pesar menos que el mismo vídeo. Publica el `.gaanim` como para compartir un enlace y pega en la entrada:

```
<iframe src="https://paololupo.github.io/gaanim/reproductor/insertar/?src=https://tu-blog.com/animaciones/circulo.gaanim&amp;loop&amp;controls=0"
  title="Animación: círculo" loading="lazy" allow="fullscreen; clipboard-write"
  style="width: 100%; aspect-ratio: 16 / 9; border: 0"></iframe>
```

`reproductor/insertar/` es un reproductor ligero hecho para esto: solo dibuja el archivo, sin la barra del editor ni el Presenter View, y su primera visita descarga unos 3 MB en lugar de los 24 del reproductor completo. La animación empieza cuando al menos una cuarta parte del marco está a la vista y se pausa cuando sale de ella: quien lee la entrada la ve desde el principio al llegar a ella, no mientras lee lo anterior. Si la pausa a mano, sigue pausada. Una pausa de la escena (`scene.stop()`) la detiene hasta un clic, como en una presentación; si la pausa tiene un bucle (`loop=`), lo repite mientras espera, y el clic sigue desde su final. Estas opciones cambian cómo se reproduce (en HTML, el & que las separa se escribe `&`):

Opción	Efecto
`loop`	Repite la animación entera, como un GIF.
`controls=0`	Oculta la barra. Un clic o un toque en la animación la pausa o la reanuda, y al terminar la repite desde el principio.
`autoplay=0`	No empieza sola: un botón en el centro espera un clic.
`preload=auto`	Descarga el archivo entero aunque no se reproduzca. Sin esta opción, un archivo de más de 2 MB en un sitio que admite descargas por partes baja al principio solo lo que necesita para abrirse, y el resto mientras se reproduce: quien pasa de largo no lo descarga entero.
`full=click`	Si la escena necesita el reproductor completo (ver abajo), no lo abre solo: muestra un aviso y lo abre al hacer clic.
`#t=2`	Empieza a los dos segundos (también `#t=1:05`).

Sin `controls=0`, una barra fina aparece al pasar el cursor por la animación y mientras está pausada: reproducir o pausar, la posición, con una marca en cada pausa de la escena, el tiempo, el bucle, el sonido si la escena lo tiene y la pantalla completa. Sin `loop`, la animación se detiene en su último fotograma con un botón para repetirla.

El reproductor ligero deja fuera el posprocesado, los fondos de shader, las transiciones de shader, los efectos de shader sobre objetos y el cristal, que se dibujan en pasadas propias de la GPU. Cuando la escena usa alguno, abre el reproductor completo con las mismas opciones, así que la animación siempre se ve como la grabaste; solo descarga más. Con `full=click` espera a un clic antes de abrirlo, para que una página con varias animaciones no descargue el reproductor completo en cada una; la página que lo contiene recibe además el mensaje `gaanim:status` de tipo `full`, con la dirección del reproductor completo en `url`, por si prefiere mostrar su propio botón (ver Controlarlo desde la página).

`aspect-ratio` da al marco la proporción de la escena: `16 / 9` para el `frame=(16, 9)` por defecto, `1 / 1` para una cuadrada. Así el marco ocupa el ancho de la columna sin franjas. Si el marco no tiene la proporción de la escena, el reproductor oscurece las franjas y marca el borde de la escena con una línea fina, como el editor; en pantalla completa, las franjas son negras. `loading="lazy"` hace que el navegador no lo cargue hasta que la entrada se acerca a él, y `allow` deja funcionar la pantalla completa.

El playground lo prepara por ti: tras ejecutar la escena, **Insertar** descarga el `.gaanim` con el nombre que elijas, compone el código con la dirección donde lo publicarás y la proporción de la escena, y muestra cómo se verá con las opciones que marques.

Antes de publicarlo, `gaanim pack` puede dejarlo más ligero sin Python:

```
gaanim pack circulo.gaanim animaciones/circulo.gaanim --web
```

Con `--web` guarda los trazados en una rejilla tan fina que, en un vídeo 4K, ningún punto se mueve más de 1/64 de píxel en cada eje (solo cambia el suavizado de algunos píxeles de borde), y deja fuera lo que el reproductor no usa: los resúmenes de `gaanim check` y los subfotogramas del desenfoque de movimiento. En los ejemplos de esta documentación pesa, en la mediana, un 41 % menos, y hasta un 97 % menos en escenas con muchas formas en movimiento. Conserva el original para exportar vídeo o comprobarlo.

Quien lee la entrada necesita un navegador con WebGPU, igual que para el reproductor completo; sin él, el marco explica qué falta. Las demás animaciones de la página, y las de visitas siguientes, toman el reproductor de la caché del navegador.

24.5.3.1. Sin sitio donde publicarla: una página HTML

Si no tienes dónde publicar el `.gaanim`, o el sitio no deja configurar CORS, el playground descarga una página HTML que lleva dentro la animación y el reproductor ligero: en **Insertar**, el botón bajo la vista previa, o la entrada `.html` del menú **Descargar**. Se abre con doble clic, sin conexión ni servidor, y se comparte como cualquier archivo; subida a un sitio, también puede ir en un `<iframe>`, y responde a los mismos mensajes que el reproductor ligero (ver Controlarlo desde la página). Sigue las opciones marcadas en **Insertar**: bucle, inicio al verse y barra de controles.

Pesa unos 4 MB más que la animación: el reproductor va dentro, y la animación, escrita en base64, ocupa un tercio más que su `.gaanim`. El reproductor completo no cabe, así que una escena con posprocesado, fondos, transiciones o efectos de shader, o cristal, se ve sin ellos; el playground avisa al descargarla. La página guarda el reproductor del día en que se descargó, sin actualizarse con el sitio, y quien la abre necesita un navegador con WebGPU.

24.5.3.2. Con el elemento `<gaanim-player>`

En lugar del `<iframe>`, puedes cargar el script del reproductor una vez por página y escribir un elemento por animación:

```
<script type="module" src="https://paololupo.github.io/gaanim/reproductor/insertar/
gaanim-player.js"></script>
```

```
<gaanim-player src="animaciones/circulo.gaanim" loop controls="0"
  label="Animación: círculo"></gaanim-player>
```

El elemento crea el marco por ti. Ocupa el ancho de la columna, toma la proporción de la escena en cuanto la abre (16:9 hasta entonces) y no carga hasta que la página se acerca a él. `src` puede ser relativa a la página. Acepta las opciones de la tabla como atributos (`loop`, `controls="0"`, `autoplay="0"`, `preload="auto"`, `full="click"`), además de:

- `start`: el segundo en que empieza, como `#t=`.
- `label`: lo que anuncia un lector de pantalla.
- `ratio`: la proporción desde el principio, como `ratio="1 / 1"`. Un `aspect-ratio` en el estilo de la página tiene prioridad sobre ambas.

Desde JavaScript, el elemento se maneja como un `<video>`:

- Métodos: `play()`, `pause()` y `seek(segundos)`.
- Propiedades: `currentTime` (al asignarla, salta a ese instante), `duration`, `paused`, `ended`, `loop` y `muted`.
- Eventos: `opened`, `play`, `pause`, `timeupdate`, `stop` (una pausa de la escena), `ended` y `error`. Cada uno lleva en `detail` el mensaje del reproductor (ver abajo).

Los métodos actúan una vez abierta la animación, es decir, después del evento `opened`.

```
const animacion = document.querySelector("gaanim-player");
animacion.addEventListener("ended", () => console.log("Terminó"));
document.querySelector("#otra-vez").addEventListener("click", () => {
  animacion.seek(0);
  animacion.play();
});
```

24.5.3.3. Controlarlo desde la página

Con un `<iframe>`, la página que lo contiene se comunica con el reproductor mediante `postMessage`, sea del sitio que sea.

Lo que avisa el reproductor. Envía mensajes `{ type: "gaanim:status", kind, time, duration, playing, ended, muted }`. Los cinco últimos campos llegan una vez abierta la animación. Cuando la animación pasa al reproductor completo, sus mensajes traen solo `time`, `playing` y `muted`, y no avisan de `stop` ni de `ended`.

`kind`	Cuándo
`ready`	El reproductor ya puede abrir archivos.
`opened`	Abrió la animación. Incluye su tamaño en píxeles (`width`, `height`), `fps` y los instantes de sus pausas (`stops`).
`redirect`	La animación necesita el reproductor completo, que ocupa el marco a continuación, con los mismos datos de tamaño; con `full=click`, el aviso es `full`.
`play`, `pause`	Empezó o se detuvo.
`timeupdate`	Cuatro veces por segundo mientras se reproduce, y tras cada salto.
`stop`	Llegó a una pausa de la escena: `stop` es su instante, y `loop` el bucle que repite mientras espera, si lo tiene.
`ended`	Llegó al final, sin `loop`.
`error`	No pudo abrir o reproducir la animación; `message` dice por qué.

Las órdenes que acepta. Recibe `{ type: "gaanim:command", command }`:

- `"play"`, `"pause"` y `"toggle"`, para reproducir, pausar o alternar.
- `"seek"`, con `time` en segundos.
- `"loop"` y `"mute"`, con `value` igual a `true` o `false`.

Por ejemplo, para dar al marco la proporción de la escena y saltar a los dos segundos:

```
const marco = document.querySelector("iframe");
window.addEventListener("message", (event) => {
  if (event.source !== marco.contentWindow || event.data?.type !== "gaanim:status")
return;
  if (event.data.kind === "opened") {
    marco.style.aspectRatio = `${event.data.width} / ${event.data.height}`;
    marco.contentWindow.postMessage(
      { type: "gaanim:command", command: "seek", time: 2 },
      "https://paololupo.github.io",
    );
  }
});
```

24.5.4. Controles táctiles

En pantallas táctiles, los controles funcionan como en un reproductor de vídeo del móvil:

- Toca la escena para mostrar u ocultar los controles; se ocultan solos tras unos segundos. Mientras se ven, los botones grandes del centro van a la pausa anterior, reproducen o pausan, y van a la pausa siguiente.
- Toca dos veces el lado derecho o el izquierdo para ir a la pausa siguiente o anterior (sin pausas, avanza o retrocede 10 segundos). Cada toque rápido más sigue avanzando.
- Desliza a la izquierda o a la derecha para ir a la pausa siguiente o anterior.
- Arrastra la barra para moverte por la presentación.

24.5.5. Presentar

Pulsa el botón de presentar de la barra: se abre el Presenter View en otra ventana, el mismo que en el escritorio, con la diapositiva actual y la siguiente, las notas y el cronómetro. Lleva la página de la presentación al proyector y ponla en pantalla completa: el aviso que aparece tiene el botón **Pantalla completa**, y también sirven `F11` y el botón del panel inferior. El navegador solo concede la pantalla completa a un clic o una tecla en esa página, y el clic que abrió el Presenter View ya se usó, por eso hace falta uno más. Deja el Presenter View en tu pantalla. Las teclas, los clics y el panel inferior mueven las dos ventanas a la vez, desde cualquiera de ellas; si cierras el Presenter View, `P` lo vuelve a abrir. Si el navegador bloquea la ventana, permite las ventanas emergentes de la página y pulsa `P`.

Un paquete con encuestas recibe votos en vivo también en la web: ver Presentar con público.

24.6. Exportar el paquete a vídeo

```
gaanim export mi-charla.gaanim --output mi-charla.mp4
gaanim export mi-charla.gaanim --output clip.webm --from intro --to cierre
gaanim export mi-charla.gaanim --output vertical.mp4 --width 1080 --height 1920 --fit
cover
```

Admite los mismos formatos y opciones que exportar un script (ver `gaanim export`), incluido `--from`/`--to` con nombres de marcadores y el audio de la escena. La frecuencia del vídeo es siempre la del paquete; `--quality` elige la compresión. El botón **Export** del editor hace lo mismo con el paquete abierto.

24.7. Desde el editor

Con un script o proyecto abierto, el botón **Export** ofrece el formato **Gaanim**: graba la escena en un `.gaanim` sin salir del editor. La calidad elige la frecuencia (Borrador, 30 fps; Estándar y Producción, 60 fps), salvo que elijas otra en **Fotogramas por segundo**; la resolución no se aplica, porque el paquete es vectorial.

24.8. Comparar capturas

`gaanim --diff --example mi-charla.gaanim` captura el fotograma de cada pausa del paquete y lo compara con la versión aprobada, sin Python (ver Capturas y comparación visual). Las capturas son idénticas a las que `--capture-stops` obtiene del script.

24.9. Reproducibilidad

Un paquete reproduce la escena tal como la exportación la dibuja, fotograma a fotograma:

- **Mismos fotogramas que un vídeo.** Los fotogramas se graban en la misma rejilla de tiempos que usa la exportación, con los mismos pasos. Un vídeo exportado desde el paquete es idéntico, píxel a píxel, al que exporta el script con la misma frecuencia y el mismo tamaño.
- **Callbacks con estado.** Algunos updaters y callbacks acumulan estado según los instantes que visita la línea de tiempo. La rejilla se graba en un mundo que visita exactamente los instantes que visita una exportación, así que su estado evoluciona igual. Los instantes que caen entre fotogramas de la rejilla (el final, los límites de segmento, las pausas, los finales de los bucles ambientales y los marcadores) también se graban, para que una presentación en pausa muestre exactamente el instante de la pausa. Si la escena tiene updaters, trazos acumulados, ecos, squash o animaciones personalizadas, cuyo estado depende de los instantes visitados, esos instantes se graban aparte, en un segundo mundo que sigue la misma rejilla, sin alterar los demás fotogramas; si no, se graban en el mismo mundo y la grabación tarda la mitad.
- **Cualquier resolución.** Los fotogramas son vectoriales y guardan qué márgenes dependen de la resolución de salida, así que exportar el paquete a cualquier tamaño da los mismos píxeles que exportar el script a ese tamaño.
- **Motion blur.** Cada fotograma de una escena con `scene.canvas.motion_blur(...)` guarda también los subfotogramas que la exportación promedia, y el vídeo del paquete los promedia igual. La reproducción en el editor muestra el fotograma sin desenfoque, como la vista previa del script.
- **Integridad.** Cada archivo interno lleva su hash BLAKE3 en el manifiesto, y cada fotograma, un resumen de lo que compone. Un paquete dañado o modificado se rechaza en lugar de mostrarse mal. `gaanim check`
  `mi-charla.gaanim` vuelve a componer todos los fotogramas y los compara con sus resúmenes, sin Python:

```
▸ check     mi-charla.gaanim · mi-charla
            Frames:     262 at 60 fps · 4.35 seconds · 1920×1080
            Structure:  5 segments · 8 stops · 0 markers · 0 audio tracks
✓ pass      every frame composes as it was recorded
```

Entre dos fotogramas grabados, el reproductor mantiene el último. A velocidad normal eso equivale al vídeo; a cámara lenta verás los fotogramas de la rejilla repetidos, porque no hay nada que interpolar sin ejecutar la escena.

24.10. Límites

La grabación se niega, con un mensaje, cuando la escena contiene clips de vídeo (`scene.media.video`), que el paquete aún no puede reproducir exactamente. En ese caso exporta un vídeo. El contenido 3D sí se graba: el paquete guarda sus mallas, materiales y luces, y el reproductor las proyecta en cada fotograma como la vista previa.

Los paquetes son de un solo uso: no se pueden editar ni volver a convertir en script. Guarda el proyecto original para seguir trabajando.

24.11. Dentro del archivo

Un `.gaanim` es un ZIP con estas entradas:

Entrada	Contenido
`manifest.json`	Formato y versión, generador, frecuencia, duración, tamaño, bloques de fotogramas con los segmentos que usa cada uno, y el hash BLAKE3 de cada entrada.
`scene.bin`	Título, fondo, color de borrado, posprocesados, segmentos con notas y pausas, marcadores, escenas y pistas de audio.
`segments/NNNNNN.bin`	Trazados, imágenes, recetas de fragmentos, cadenas, mallas y líneas 3D, y los shaders de las transiciones y de los efectos por objeto, cada uno guardado una sola vez. Un segmento se cierra al llegar a unos 2 MB y va justo antes del bloque de fotogramas que lo completó.
`frames/NNNNNN.bin`	Bloques de 60 fotogramas; cada bloque empieza con un fotograma completo y el resto guarda solo lo que cambió.
`scenes/NNNNNN.bin`	Cada fotograma distinto de una animación Lottie, como escena vectorial de Vello.
`index.bin`, `digests.bin`	Instante y resumen BLAKE3 de cada fotograma. Un fotograma de la rejilla de la exportación ocupa un byte en el índice.
`media/*`	Archivos de audio, direccionados por contenido.
`thumbnail.png`	Portada opcional de 512 píxeles de ancho, que el explorador de archivos lee sin decodificar el resto.

Los números se guardan sin pérdida: los reales conservan todos sus bits, de modo que el paquete compone exactamente lo que compuso la escena.

Salvo `manifest.json`, `media/*` y `thumbnail.png`, cada entrada guarda sus datos comprimidos con Zstandard. En una presentación larga ocupa menos de la mitad que con la compresión propia del ZIP; en un paquete pequeño, alrededor de un 10 % menos.

Un reproductor abre el paquete con el manifiesto, `scene.bin`, el primer bloque y los segmentos que usa, y descarga los demás bloques con sus segmentos a medida que los reproduce. Así, una presentación larga empieza a verse sin esperar a descargar los trazados de todas sus diapositivas.

Este es el formato 6. Gaanim sigue leyendo el formato 5 (versiones 0.9 a 0.12), así que los paquetes ya publicados siguen abriéndose. Los grabados con versiones anteriores a la 0.9 ya no se abren: vuelve a grabarlos desde su script.

CAPÍTULO

25. Capturas y comparación visual

Fotogramas exactos de tu escena, comparados contra una versión aprobada

En esta guía aprenderás a comprobar que un cambio en el código no alteró lo que se ve. `gaanim --diff` captura fotogramas exactos de una escena, los compara con una versión que ya aprobaste y genera un informe HTML para revisar las diferencias en el navegador.

25.1. Dónde se guardan las capturas

Cada escena tiene su propia carpeta dentro del proyecto:

```
tests/visual/<nombre-de-la-escena>/
  baseline/  # capturas aprobadas y manifest.json: guárdalas en git
  current/   # captura de la versión actual: local
  report/    # informe HTML y JSON y mapas de calor: local
```

Por ejemplo, `escenas/intro.py` usa automáticamente `tests/visual/escenas/intro/`. No hace falta escribir las rutas de `baseline`, `current` ni `report`.

25.2. Flujo normal

1. Elige los instantes que quieres capturar. El script llama a `scene.snapshots(...)` cuando existe la variable de entorno `GAANIM_SNAPSHOTS`; `gaanim --diff` la define y ejecuta la captura sin abrir ventana:

   ```python
   import os
   if os.environ.get("GAANIM_SNAPSHOTS"):
       scene.snapshots(os.environ["GAANIM_SNAPSHOTS"], [0.0, 0.5, 1.0])
   ```

2. Cuando la escena se vea como quieres, apruébala. `--bless` guarda la captura como versión aprobada (`baseline/`):

   ```
   gaanim --diff --example escenas/intro.py --bless
   ```

3. Después de cada cambio, captura otra vez y compara con la versión aprobada. El comando imprime la ruta de `report/index.html`; ábrela en el navegador:

   ```
   gaanim --diff --example escenas/intro.py
   ```

`--bless` sobrescribe la versión aprobada: úsalo solo cuando hayas revisado y aceptado el cambio visual.

Si prefieres capturar las pausas del script, `scene.stops` devuelve cada `scene.stop()` con su instante y `scene.cursor` da el instante actual de la escena:

```python
if os.environ.get("GAANIM_SNAPSHOTS"):
    scene.snapshots(os.environ["GAANIM_SNAPSHOTS"], [stop.time for stop in scene.stops])
```

25.3. Capturar cada pausa de una presentación

Para revisar una presentación, `--capture-stops` captura el fotograma que se ve en cada `scene.stop()` sin tocar el script: no hace falta llamar a `scene.snapshots` ni leer `GAANIM_SNAPSHOTS`. El script solo debe terminar con `scene.render()`.

```
gaanim --diff --example mi-charla --capture-stops --capture-only
gaanim --diff --example mi-charla --capture-stops --stops 12,30 --capture-only
```

- Cada pausa se captura en su instante exacto: las animaciones anteriores ya terminaron, y una pausa al final de un segmento conserva ese segmento en pantalla.
- Los archivos se llaman `stop_0001.png`, `stop_0002.png`… según la numeración global de las pausas (desde 1), así que la versión aprobada y la actual se emparejan por pausa aunque cambien los tiempos.
- `--stops` acepta números y rangos (`12,30`, `3-7`) de esa misma numeración.
- `--sections resultados,cierre` y `--from resultados` capturan solo las pausas de esos segmentos, claves de `Section` o nombres de `SectionStep` (ver Presentaciones); se combinan con `--stops` y conservan la numeración global.
- Además de `manifest.json`, escribe `stops.json` con el número, el instante, el segmento, el nombre de la pausa y el archivo de cada captura.
- Funciona con `--bless` y `--capture-only`, pero no con `--no-capture`.
- Solo compara si la versión aprobada tiene su propio `stops.json`. Si no hay versión aprobada, o se hizo con `scene.snapshots`, captura, avisa de que no hay nada que comparar y termina con éxito. `--bless` con `--capture-stops` reemplaza el `manifest.json` aprobado: no lo uses sobre una versión de `scene.snapshots` que quieras conservar.

25.3.1. Paquetes de reproducción

Un paquete `.gaanim` (ver Compartir sin Python) también se compara, sin Python: `--example` con un paquete captura siempre sus pausas, así que no hace falta `--capture-stops`, y admite `--stops`, `--sections` y `--from`.

```
gaanim --diff --example mi-charla.gaanim --bless
gaanim --diff --example mi-charla.gaanim
```

Las capturas de un paquete son idénticas a las de `--capture-stops` sobre el script que lo grabó, así que puedes comparar la versión aprobada de uno con la actual del otro usando `--baseline` y `--current`, por ejemplo para comprobar que un paquete compartido muestra lo mismo que el proyecto.

25.4. Automatizar y tolerar diferencias

```
gaanim --diff --example escenas/intro.py --pixel-threshold 4 --max-changed-ratio
0.0001
```

- `--no-capture` compara los PNG que ya están en `current/`.
- `--capture-only` escribe `current/` (o la carpeta de `--current <DIR>`) y termina sin comparar ni modificar la versión aprobada.
- `--capture-stops` y `--stops <LISTA>` capturan cada `scene.stop()` en lugar de usar `scene.snapshots` (ver arriba).
- `--tests-root <DIR>` cambia la carpeta raíz, que por defecto es `tests/visual`.
- `--pixel-threshold` y `--max-changed-ratio` fijan cuánto puede cambiar cada píxel y qué proporción de píxeles puede cambiar sin que cuente como diferencia.

Para comparar dos carpetas cualesquiera, indica las rutas a mano con `--baseline`, `--current` y `--output`:

```
gaanim --diff --baseline capturas/antes --current capturas/despues --output capturas/informe
```

25.5. Leer el resultado

El informe JSON incluye, para cada captura, el instante, el porcentaje de píxeles modificados, el error medio, la diferencia máxima y el rectángulo que contiene el cambio. El comando termina con código 0 si no hay diferencias, 1 si las hay y 2 ante errores, así que sirve tal cual en integración continua.

`report/index.html` funciona sin conexión y muestra lo mismo que el JSON:

- La lista lateral filtra por fallos, cambiados o ausentes y busca por nombre.
- Siete modos de comparación: diff, versión aprobada, actual, lado a lado, cortina, superposición con opacidad y parpadeo. Las teclas `1` a `7` los alternan.
- Un recuadro marca el área cambiada; `B` lo oculta.
- ←/→ recorren las capturas, `F` ajusta a la ventana, `0` muestra el tamaño real y +/− (o `Ctrl` + rueda) cambian el zoom. `?` lista todos los atajos.
- La dirección de la página incluye la captura y el modo, para compartir un enlace a un fotograma concreto.

PARTE IV

Público en vivo

Encuestas, cuestionarios, equipos y personajes del público: la API te da los datos y tú diseñas el juego

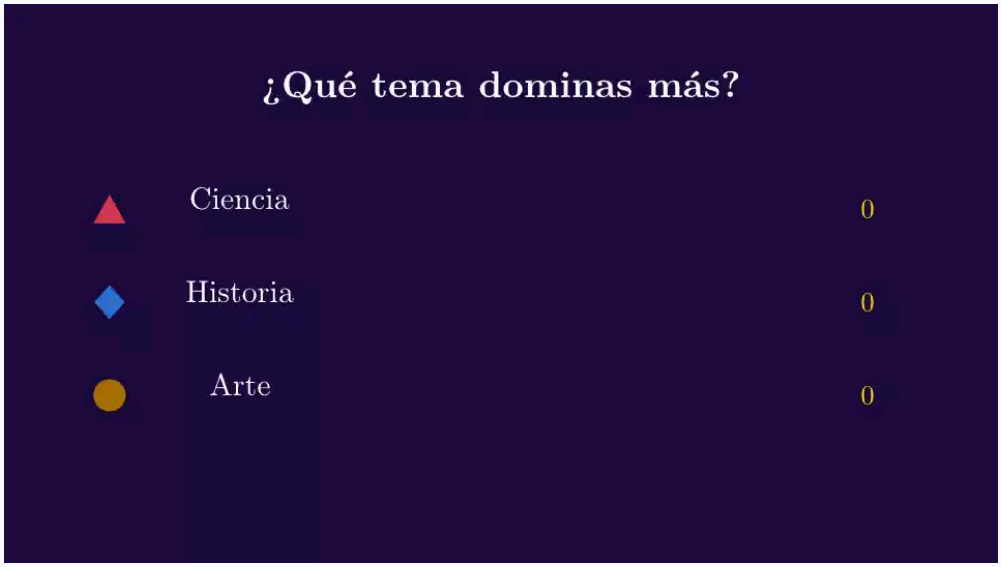
CAPÍTULO

26. Público en vivo

Encuestas, cuestionarios, equipos y personajes del público dentro de tu escena

Mientras presentas, el público puede responder desde su teléfono: votar en una encuesta, competir en un cuestionario al estilo de Kahoot, jugar por equipos y hasta entrar a la escena como el personaje que armó. Escanean un QR, escriben un apodo y la escena reacciona a lo que hacen.

```
from gaanim import GOLD, WHITE, Scene

scene = Scene(frame=(16, 9), background="#1c0b3f")
poll = scene.poll("¿Qué tema dominas más?", ["Ciencia", "Historia", "Arte"],
                  rehearse=[4, 3, 2])

scene.text(poll.question, size=0.6, color=WHITE, weight=700).move_to(0, 3)
for i, answer in enumerate(poll.options):
    y = 1.2 - 1.5 * i
    poll.icon(i, 0.55).move_to(-6.3, y)
    scene.text(answer, size=0.5, color=WHITE, text_box="cap").move_to(-4.2, y)
    bar = poll.bar(i, length=7, thickness=0.6, radius=0.3).fill(poll.color(i)).no_stroke()
    bar.move_to(1.6, y)
    scene.viz.readout(poll.votes(i), format=".0f", color=GOLD, font_size=0.45).move_to(5.9, y)
scene.wait(4)
scene.stop()
scene.render()
```



En la vista previa vota un público inventado, el ensayo, así que puedes diseñar todo el juego sin teléfonos. Al presentar, las mismas barras siguen los votos reales.

26.1. Cómo funciona

Hay cuatro piezas:

- **La escena**, que escribes en Python como cualquier otra. Abre preguntas con `scene.poll` y `scene.quiz` y decide cómo se ven.
- **El relay**, un pequeño servicio web gratuito que despliegas una vez en tu cuenta de Cloudflare. Lleva las preguntas a los teléfonos y los votos a la presentación.
- **Los teléfonos** del público, que abren la página del relay con el QR. No instalan nada ni crean cuentas.
- **El ensayo**, un público inventado que juega la escena en la vista previa, al exportar y en las capturas, siempre igual.

Al presentar con `gaanim --present`, Gaanim abre cada pregunta en el relay cuando llegas a ella y lee los resultados mientras el público responde. Un paquete `.gaanim` exportado también recibe votos en vivo, sin Python.

26.2. Datos, no diseño

Gaanim no trae una pantalla de Kahoot ya hecha. Te da los datos y piezas para dibujarlos, y el diseño es tuyo:

- **Números en vivo**: votos, porcentajes, respuestas, segundos que quedan, puntos de cada jugador y de cada equipo. Son `Parameter`, así que los usas en lecturas, en `computed` o para mover y escalar cualquier cosa.
- **Piezas listas**: el QR, barras que siguen a una respuesta, los apodos como texto en vivo, la forma y el color que cada respuesta tiene en el teléfono.
- **Momentos**: dónde se abre y se cierra una pregunta, dónde se revela la respuesta y cuándo una pausa avanza sola porque el público ya respondió.
- **Personajes**: con una zona viva, cada jugador entra a la escena como su personaje y una función de Python decide cómo se mueve.

Con esas piezas puedes hacer un Kahoot clásico, una carrera, una batalla entre equipos, un podio o algo que todavía no existe. El recetario de estilos tiene varios para copiar.

26.3. Por dónde seguir

CAPÍTULO

27. Tu primera encuesta

Despliega el relay, abre una encuesta y preséntala con teléfonos reales

En esta guía pondrás una encuesta en una diapositiva y la presentarás con teléfonos reales. Lo haces en tres pasos: despliegas el relay una sola vez, escribes la encuesta y presentas.

27.1. 1. Despliega el relay

Los votos viajan a través de un **relay**, un pequeño servicio web que despliegas tú, gratis, en tu propia cuenta de Cloudflare. Como teléfonos y presentación solo se conectan hacia fuera, por HTTPS, funciona en redes universitarias que aíslan a los dispositivos entre sí y con datos móviles. La página de votación está pensada para el teléfono: respuestas grandes con color, letra y forma, tema claro u oscuro y español o inglés según el teléfono.

La forma más corta es el botón Deploy to Cloudflare: copia el relay a un repositorio tuyo y lo publica desde ahí. También puedes desplegarlo desde tu equipo, con Node.js instalado:

```
gaanim relay init            # escribe el relay en ./gaanim-relay
cd gaanim-relay
npx wrangler login           # una cuenta gratuita de Cloudflare
npx wrangler deploy          # imprime https://gaanim-relay.<tú>.workers.dev
```

Después dile a Gaanim dónde está:

```
gaanim relay use https://gaanim-relay.<tú>.workers.dev
```

`gaanim relay` sin argumentos muestra el relay en uso y si habla el mismo protocolo que tu versión de Gaanim; al presentar, Gaanim avisa si no. Para actualizarlo, vuelve a escribirlo sobre su carpeta con `gaanim relay init --force gaanim-relay` y despliégalo otra vez.

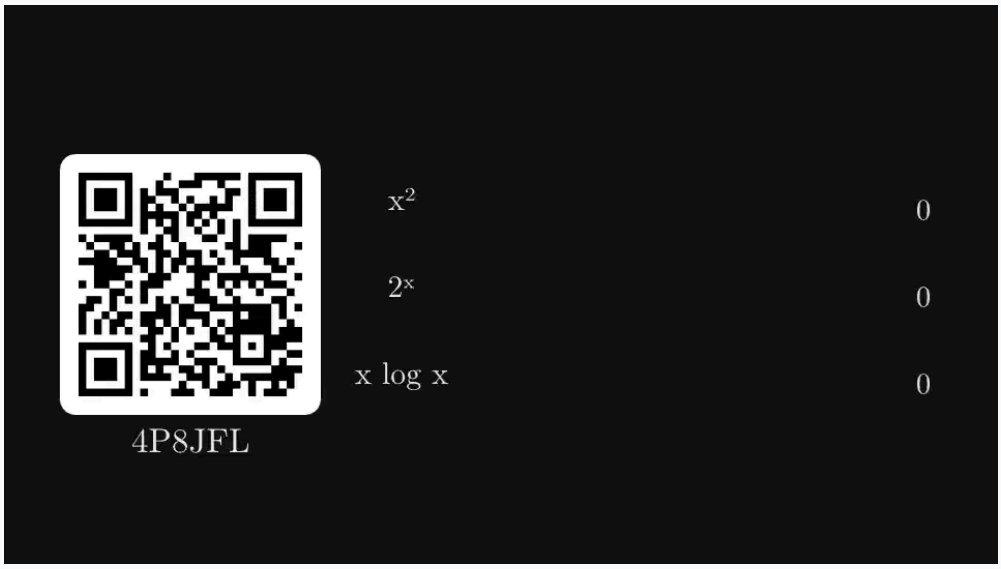
Un proyecto puede usar otro relay con `[polls] relay` en su `gaanim.toml`, y la variable `GAANIM_POLL_RELAY` tiene prioridad sobre ambos. Sin relay, `scene.poll` avisa y el QR no lleva a ninguna parte. La dirección queda grabada en el QR, así que si cambias de relay vuelve a exportar.

27.2. 2. Escribe la encuesta

`scene.poll` hace una pregunta al público y te da los datos para que tú decidas cómo presentarla:

```
from gaanim import RED, WHITE, Scene

scene = Scene(frame=(16, 9))
poll = scene.poll("¿Qué curva crece más rápido?", ["x²", "2ˣ", "x log x"], rehearse=[3, 7, 2])

card = scene.geometry.rounded_rect(4.2, 4.2, 0.25).fill(WHITE).no_stroke().move_to(-5, 0)
qr = poll.qr(3.6).move_to(-5, 0)                          # el QR, un drawable más
code = scene.text(poll.code, size=0.55).move_to(-5, -2.7)  # por si no pueden escanear

for i, answer in enumerate(poll.options):
    y = 1.2 - 1.4 * i
```

```
    scene.text(answer, size=0.5, text_box="cap").move_to(-1.6, y)
    poll.bar(i, length=6, thickness=0.6, radius=0.3).fill(RED).no_stroke().move_to(3.2, y)
    scene.viz.readout(poll.votes(i), format=".0f").move_to(6.8, y)

scene.wait(3)
scene.stop()
scene.render()
```



La encuesta no dibuja nada por sí misma. Te da:

- `poll.qr(tamaño)`: el código QR como drawable, relleno de negro. Ponlo sobre un fondo claro con algo de margen para que los teléfonos lo lean.
- `poll.code` y `poll.url`: el código de seis caracteres y la dirección de votación, para mostrarlos como texto.
- `poll.votes(i)`, `poll.share(i)` (de 0 a 1), `poll.percent(i)` (de 0 a 100) y `poll.total()`: `Parameter` que siguen los votos. Úsalos como cualquier parámetro: en `readout`, en `computed`, en puntos o en líneas reactivas (ver Reactividad).
- `poll.bar(i, ...)`: una barra lista cuya longitud sigue a la respuesta `i`. Con `direction` crece hacia la derecha, la izquierda, arriba o abajo; con `scale="leader"` (por defecto) la respuesta que va ganando llena su barra, y con `"total"` cada barra mide su porcentaje. Sus límites son los de la barra completa, así que el layout no se mueve al llegar votos.
- `poll.icon(i, tamaño)` y `poll.color(i)`: la forma y el color que la respuesta `i` tiene en el teléfono, para que la pantalla y el teléfono se parezcan.
- `poll.close()`: deja de recibir votos en ese punto. Sin él, la encuesta se cierra al terminar el segmento donde se abrió. Los valores conservan el último conteo, así que un paso posterior puede comentar el resultado.

Cada teléfono cuenta una vez y puede cambiar su voto mientras la pregunta sigue abierta. Si vuelves a una encuesta, conserva sus votos. `rehearse` da al ensayo un peso por respuesta (`[3, 7, 2]` hace la segunda más popular), para que la vista previa se parezca a lo que esperas.

El código de la sesión es fijo para cada proyecto y el QR es contenido normal de la escena: se ve igual en la previsualización, en un vídeo exportado o en un paquete `.gaanim`, y los teléfonos lo escanean una vez por presentación. La clave que permite abrir preguntas y leer votos se guarda solo en tu equipo. Para fijar el código, por ejemplo si varias personas presentan el mismo proyecto, usa la variable `GAANIM_POLL_SESSION`.

27.3. 3. Presenta

```
gaanim --present main.py
```

Mientras la presentación está dentro de la encuesta, los votos llegan en vivo y las barras crecen. Prueba antes con dos o tres teléfonos tuyos: escanea el QR, vota y mira cómo cambia la pantalla. Para seguir, lee Presentar con público, que cuenta qué ves en la vista del presentador y qué hacer si algo falla.

Los votos son anónimos: el teléfono guarda un identificador al azar y el relay no pide nombres ni cuentas. El relay borra la sesión y sus votos doce horas después de su última actividad.

CAPÍTULO

28. Anatomía de un juego

Cuestionarios, revelación, clasificación, sala de espera, equipos y el ensayo

En esta guía verás las piezas con las que se arma un juego: una sala de espera mientras el público entra, preguntas con cuenta atrás, el momento en que revelas la respuesta, la clasificación y los equipos. Cada pieza te da datos y momentos; cómo se ven lo decides tú.

Un juego típico sigue este ritmo, una diapositiva por paso:

1. **Sala de espera**: el QR y los apodos de quienes van entrando (`scene.audience`).
2. **Pregunta**: `scene.quiz` abre la pregunta y el reloj empieza a correr.
3. **Pausa**: `scene.stop(until=...)` espera a que respondan, o a que se acabe el tiempo, y avanza sola.
4. **Revelación**: `quiz.reveal()` muestra en cada teléfono si acertó y cuántos puntos ganó; en la pantalla, tú resaltas la correcta.
5. **Clasificación**: `scene.leaderboard` ordena a los jugadores para un podio, una carrera o una lista.

28.1. Cuestionarios

`scene.quiz` es una encuesta con respuesta correcta, al estilo de Kahoot. Los teléfonos se unen al juego con un apodo, ven una cuenta atrás, responden una sola vez y una respuesta correcta gana más puntos cuanto antes llega: todos los puntos al instante y la mitad en el último segundo. `quiz.reveal()` marca el punto en que la presentación revela la respuesta: cada teléfono muestra si acertó, cuántos puntos ganó y su puesto. Justo después de un `scene.stop()`, espera mientras la presentación descansa en la pausa y revela al avanzar.

```
quiz = scene.quiz("¿Cuál es la derivada de x²?", ["x", "2x", "x²/2", "2"],
                  correct=1, time=20, rehearse=0.7)
clock = scene.viz.readout(quiz.remaining(), format=".0f")   # la cuenta atrás
scene.stop()        # el público responde
quiz.reveal()       # al avanzar, los teléfonos ven el resultado
scene.wait(2)
scene.stop()

board = scene.leaderboard()
for rank, x in [(1, -4), (0, 0), (2, 4)]:
    board.bar(rank, length=4, thickness=2.6, direction="up").fill(GOLD).move_to(x, -1.2)
    board.name(rank, size=0.55, align="center").move_to(x, 1.9)
    scene.viz.readout(board.points(rank), format=".0f", suffix=" pts").move_to(x, 1.3)
```

`quiz.remaining()` sigue el reloj del relay al presentar; en la previsualización llega a 0 en la pausa donde la presentación espera las respuestas. `quiz.correct` es el índice de la respuesta correcta (o la lista, en selección múltiple), para que resaltes la ficha que corresponde.

Ver cómo van los votos mientras la pregunta sigue abierta influye en las respuestas. `quiz.revealed()` vale 0 hasta el `reveal()` y 1 después, y con él decides qué se ve antes: barras ocultas, porcentajes tapados o un «?».

```
quiz = scene.quiz("¿2 + 2?", ["3", "4"], correct=1)
shown = quiz.revealed()
for i in range(2):
    quiz.bar(i, length=3).move_to(4 * i - 2, 0).opacity(shown)
scene.stop()
quiz.reveal()       # desde aquí las barras se ven
```

28.2. La clasificación

`scene.leaderboard()` da a los jugadores de mejor a peor. Cada método recibe un puesto, 0 para el primero:

- `board.name(i)`: el apodo como texto en vivo, con `size`, `weight`, `font` y `align` para alinearlo por la izquierda, el centro o la derecha.
- `board.points(i)`, `board.correct(i)`, `board.answered(i)`, `board.streak(i)` (aciertos seguidos) y `board.players()`: `Parameter`.
- `board.bar(i, ...)`: una barra que mide los puntos del puesto `i` frente a los del primero.

Un puesto sin jugador muestra un apodo vacío, 0 puntos y una barra vacía, así que puedes dibujar un podio de tres aunque jueguen dos. Todo sigue a los resultados en vivo, también al presentar un `.gaanim`.

28.3. Avanzar por sí sola

`scene.stop(until=...)` hace que una pausa avance sola cuando el público cumple una condición: al presentar, en cuanto se cumple, la presentación sigue como si hubieras pulsado siguiente.

```
quiz = scene.quiz("¿Cuál es la derivada de x²?", ["x", "2x", "x²/2", "2"], correct=1)
scene.stop(until=quiz.answered(share=0.8) | quiz.time_up())
quiz.reveal()
```

- `poll.answered(at_least=10)`: al menos 10 respuestas.
- `poll.answered(share=0.8)`: respondió el 80 % del público, los jugadores en un cuestionario y los teléfonos conectados en una encuesta.
- `quiz.time_up()`: se acabó el tiempo del cuestionario.
- `audience.at_least(5)`: entraron al menos 5 jugadores.

Se combinan con `|` (cualquiera) y `&` (todas). Solo avanza sola una pausa a la que llegaste avanzando: si vuelves atrás a una cuya condición ya se cumple, se queda ahí hasta que pulses siguiente. La vista del presentador muestra cuánto falta, y en la previsualización y la exportación es una pausa normal. Funciona igual al presentar un paquete `.gaanim`.

28.4. Sala de espera

`scene.audience` da los jugadores en el orden en que entran, para llenar una sala de espera mientras el público escanea el QR. Una escena que lo usa pide el apodo en cuanto el teléfono abre la página, en vez de esperar al primer cuestionario.

```
audience = scene.audience()
audience.qr(3).move_to(-5, 0)
for slot in range(12):
    x, y = (slot % 4) * 2.2 - 1, 1.5 - slot // 4 * 1.5
    pop = computed(lambda a: min(a / 0.4, 1), inputs=[audience.age(slot)])
    audience.name(slot, size=0.4).move_to(x, y).scale_to(pop)
players = scene.viz.readout(audience.count(), format=".0f", suffix=" jugadores")
scene.stop("sala", until=audience.at_least(5))
```

`audience.name(i)` es el apodo de quien entró en el puesto `i`, vacío mientras nadie lo ocupa. `audience.joined(i)` vale 1 cuando el puesto está ocupado y `audience.age(i)` cuenta los segundos desde que entró, hasta 60, para animar su llegada. Si quitas a un jugador, los siguientes suben un puesto. Para que entren como personajes que corren, saltan o se estrellan contra la pantalla, usa una zona viva.

28.5. Equipos

`scene.teams` hace que el juego se juegue por equipos, por ejemplo una batalla entre dos grupos del curso. Cada jugador entra a un equipo: el relay lo pone en el que tiene menos jugadores, o con `choose=True` el teléfono le pregunta a cuál quiere unirse (hasta que responda su primera pregunta puede cambiar). El teléfono muestra su equipo con su color y, al final, qué equipo ganó. Los puntos de cada jugador suman para su equipo.

```
teams = scene.teams(["Rojo", "Azul"], colors=["#ff4f8b", "#2fb8ff"])
scene.rehearsal(20, skill=[0.7, 0.5])     # un ensayo con un equipo más fuerte

for team, x in [(0, -4), (1, 4)]:
    scene.text(teams.names[team], size=0.6).move_to(x, 3)
    scene.viz.readout(teams.score(team), format=".0f", suffix=" pts").move_to(x, 2)
    teams.bar(team, length=5, direction="up").fill(teams.colors[team]).move_to(x, -1)
```

- `teams.score(i)`, `teams.players(i)` y `teams.average(i)` (puntos por jugador, más justo si los equipos quedan desparejos) son `Parameter`.
- `teams.leader()` es el índice del equipo que va ganando.
- `teams.bar(i, ...)` es una barra que mide los puntos del equipo frente al que va primero.
- En una zona viva, cada jugador sabe su equipo: `p.team`, su orden de llegada dentro del equipo `p.team_index`, cuántos son `p.team_count`, los puntos del equipo `p.team_score` y su puesto `p.team_rank` (0 para el que va ganando). Sin equipos, todos están en el equipo 0.

```
def battle(p):
    side = -1 if p.team == 0 else 1         # cada equipo en su lado
    x = side * (2 + p.team_index // 4 * 1.2)
    y = -2 + p.team_index % 4 * 1.1
    if p.team_rank == 0:                      # el que va ganando salta
        y += 0.5 * abs(math.sin(3 * p.t))
    return pose(x, y, flip=side > 0, express="winner" if p.team_rank == 0 else None, loop=True)
```

28.6. Las respuestas de cada jugador

Además de los totales, puedes saber qué respondió cada jugador, por su puesto en la sala (`audience`) o en la clasificación (`board`):

```
quiz = scene.quiz("¿2 + 2?", ["3", "4"], correct=1)
audience = scene.audience()
board = scene.leaderboard()
eligio = audience.chose(0, quiz, 1)     # 1 si el primero en llegar eligió la 2.ª
acerto = audience.right(0, quiz)        # 1 si acertó
tardo = audience.answer_time(0, quiz)   # segundos que tardó
racha = board.streak(0)                 # aciertos seguidos del primero
```

`responded`, `chose`, `right`, `earned` y `answer_time` hablan de una pregunta; `correct`, `answered` y `streak` (y `audience.score`) del juego hasta ahora. Todos son `Parameter`. En las zonas vivas cada personaje lo sabe de sí mismo: `p.answer`, `p.answer_mask`, `p.answer_time`, `p.answer_points`, `p.answers`, `p.correct` y `p.streak`.

28.7. Ensayo

Fuera de una presentación en vivo, un público inventado juega la escena: en la previsualización, la exportación y las capturas los jugadores entran a la sala, responden cada pregunta, el reloj baja y los puntos suben, como en una sesión real. Todo sale del mismo grupo, así que los números cuadran en todas partes: los votos de una pregunta suman los jugadores que había, y la clasificación suma los puntos que ganaron sus respuestas. Las zonas vivas también lo usan.

```
scene.rehearsal(24, seed=3, skill=0.65, speed=0.5)

audience = scene.audience()                 # entran uno tras otro
quiz = scene.quiz("¿Capital de Australia?", ["Sídney", "Canberra", "Perth"],
                  correct=1, rehearse=0.4)  # solo el 40 % acierta
poll = scene.poll("¿Te gustó?", ["Sí", "No"], rehearse=[5, 1])
board = scene.leaderboard()                 # puntos de sus respuestas
```

- `players`: cuántos jugadores (se llaman Ana, Beto, Caro…) o una lista de apodos, en el orden en que entran. Sin llamar a `scene.rehearsal` ensayan 12.
- `skill` (0 a 1): cuántas preguntas aciertan en promedio; cada jugador es algo mejor o peor que el resto, así que la clasificación se reparte. Con equipos, acepta un valor por equipo para ensayar una batalla despareja.
- `speed` (0 a 1): qué tan pronto responden.
- `arrive`: en cuántos segundos entran todos; por defecto, entre `scene.audience()` y la primera pausa de la sala.
- `seed`: otro grupo, con otros tiempos, respuestas y personajes.

Por pregunta, `rehearse=` en `scene.quiz` es la parte que acierta, y en `scene.poll` (o en un cuestionario) un peso por respuesta.

El ensayo sigue el ritmo de la escena: las respuestas de una pregunta llegan entre que se abre y la pausa donde la presentación esperaría por ellas, así que la previsualización es una versión acelerada de lo que pasará en vivo. Con los mismos argumentos el ensayo es siempre el mismo, en cada vista previa y en cada exportación. Diseña con un ensayo parecido a tu público: con `scene.rehearsal(40)` verás si tu sala de espera aguanta un curso grande.

CAPÍTULO

29. Preguntas

Imágenes, selección múltiple y preguntas escritas en Markdown o en una hoja de cálculo

En esta guía verás qué clases de pregunta puedes hacer y cómo escribirlas fuera de Python, en un archivo que cualquiera del equipo puede editar.

29.1. Tipos de pregunta

Además de una sola respuesta, una pregunta puede llevar una imagen y aceptar varias respuestas:

```
quiz = scene.quiz("¿De qué país es esta bandera?", ["Japón", "Bangladés", "Palaos"],
                  correct=1, image="assets/bandera.png")
paises = scene.quiz("¿Qué países están en Sudamérica?", ["Perú", "Chile", "México", "Brasil"],
                    correct=[0, 1, 3])          # selección múltiple
gustos = scene.poll("¿Qué te gustó?", ["Mapas", "Banderas", "El juego"], multiple=True)
```

- `image=` muestra la imagen sobre la pregunta en los teléfonos, reducida a 1024 píxeles. En la escena la pones tú, con `scene.media.image`.
- Con `correct=[...]` el cuestionario es de selección múltiple: el teléfono deja marcar varias respuestas y enviarlas, y solo suma puntos quien marcó todas las correctas y ninguna otra.
- `multiple=True` en una encuesta deja votar por varias. `votes(i)` cuenta cada respuesta elegida, y `total()` y `share(i)` cuentan a los teléfonos que respondieron.
- `poll.icon(i, tamaño)` es la forma que tiene la respuesta `i` en el teléfono (triángulo, rombo, círculo, cuadrado, estrella, hexágono), con su color, y `poll.color(i)` el color. En un cuestionario los teléfonos ordenan y colorean las respuestas a su manera para que no se copien, así que ahí la forma de la pantalla no coincide con la del teléfono.

Una pregunta tiene de 2 a 6 respuestas de hasta 120 caracteres, y la pregunta, hasta 300.

29.2. Preguntas en un archivo

Las preguntas se pueden escribir fuera de Python, en un Markdown o en una hoja de cálculo, y abrirlas con `scene.question`:

```
# ¿De qué país es esta bandera?
![](assets/bandera.png)
tiempo: 15
acierta: 45%

- Japón
- [x] Bangladés
- Palaos

# ¿Qué países están en Sudamérica?

- [x] Perú
```

```
- [x] Chile
- México
- [x] Brasil

# ¿Qué te gustó más?
modo: varias
votos: 3, 2, 4

- Los mapas
- Las banderas
- El juego
```

Cada `#` empieza una pregunta y cada elemento de la lista es una respuesta; `[x]` marca las correctas (varias la hacen de selección múltiple) y sin ninguna es una encuesta. El texto antes de la primera pregunta se ignora, así que puedes dejar ahí instrucciones para quien edite el archivo. Las líneas `clave: valor` ajustan:

Clave	Qué ajusta
`tiempo`	Segundos para responder un cuestionario, de 5 a 300.
`puntos`	Puntos máximos de una respuesta correcta, de 100 a 10000.
`modo`	`una` o `varias`: si una encuesta acepta varias respuestas.
`imagen`	Una imagen, como también la acepta `![](ruta)`.
`acierta`	La parte del ensayo que acierta, como `70%` o `0.7`.
`votos`	Un peso por respuesta para el ensayo, como `3, 1, 2`.
`notas`	Notas del orador para la diapositiva.

En una hoja de cálculo (`.csv`) va una pregunta por fila con las columnas `pregunta`, `respuesta 1` a `respuesta 6` y `correcta` (los números de las correctas desde 1, como `2` o `1, 3`), y opcionalmente las mismas claves; la plantilla de Kahoot sirve tal cual.

```python
from gaanim import load_questions

for q in load_questions("preguntas.md"):
    scene.segment(q.text, notes=q.notes)
    poll = scene.question(q)                  # cuestionario o encuesta
    scene.text(q.text, size=0.6).move_to(0, 3.5)
    if q.image:
        scene.media.image(q.image, width=5, height=3).move_to(4, 0)
    for i, respuesta in enumerate(q.options):
        poll.icon(i, 0.5).move_to(-6, 2 - i * 1.2)
        scene.text(respuesta, size=0.45).move_to(-3.5, 2 - i * 1.2)
    scene.stop()
    if q.is_quiz:
        poll.reveal()
        scene.wait(1)
        scene.stop()
```

`load_questions` lee la ruta relativa al script y devuelve objetos `Question` con `text`, `options`, `correct` (los índices de las correctas, vacío en una encuesta), `multiple`, `time`, `points`, `image`, `rehearse`, `notes` e `is_quiz`. El diseño de cada diapositiva sigue siendo tuyo: puedes elegir un diseño para las preguntas con imagen y otro para las demás, o intercalar una diapositiva propia después de una pregunta concreta. Un archivo con un error lanza `QuestionError` con el nombre del archivo y la línea.

CAPÍTULO

30. Zonas vivas

El público entra a la escena como sus personajes y tú decides cómo se mueven, con funciones normales de Python

En esta guía aprenderás a poner al público dentro de tu escena: mientras presentas, cada persona que se une desde su teléfono aparece como el personaje que armó y tú decides cómo se mueve con una función de Python. Sirve para una sala de espera, una carrera con los puntajes, un podio o una batalla entre equipos.

```
from gaanim.live import pose, smoothstep

scene = Scene(frame=(16, 9))
audience = scene.audience()

def fila(p):
    llegada = smoothstep(min(p.t / 0.8, 1))      # entra en 0,8 s
    x = -6 + p.index * 1.5
    return pose(x, -2 + 4 * (1 - llegada), express="happy")

scene.live_zone(audience, fila, names=True)
scene.wait(6)
```

`scene.live_zone(audience, comportamiento)` abre una zona en el cursor que dura hasta `zone.close()` o el final del segmento. El comportamiento es una función normal de Python que recibe al jugador `p` y devuelve `pose(x, y, ...)`: dónde están sus pies y cómo se ve. Gaanim la compila al crear la escena, así que un `.gaanim` exportado la ejecuta sin Python. Fuera de una presentación en vivo, la zona reproduce el ensayo, el público inventado de la escena.

Abre la zona al empezar el segmento y deja tiempo antes de la pausa, como el `scene.wait(6)` del ejemplo. Al presentar, la escena descansa en la pausa y los personajes siguen moviéndose, pero en la vista previa y en una exportación el tiempo no pasa en la pausa: una zona abierta justo antes de un `scene.stop()` se ve sin nadie dentro.

30.1. El jugador

`p` trae lo que el motor sabe del jugador en cada cuadro:

- `p.t`: segundos desde que llegó a la zona; `p.time`, desde que se abrió.
- `p.joined`: cuándo llegó; `p.index`: su orden de llegada, desde 0; `p.count`: cuántos hay.
- `p.rank` (0 para el que va primero), `p.score`, `p.leader` (el puntaje del primero), `p.previous_rank`, `p.rank_since`, `p.previous_score` y `p.score_since`, para animar los cambios de puesto.
- `p.team`, `p.team_index`, `p.team_count`, `p.team_score` y `p.team_rank` en un juego con equipos.
- `p.answer` (la respuesta que eligió en la pregunta abierta, o la última; −1 si ninguna), `p.answer_mask` (todas las que eligió, un bit cada una), `p.answer_time`, `p.answer_points`, y en el juego `p.answers` (respondidas), `p.correct` (acertadas) y `p.streak` (aciertos seguidos).
- `p.random(k)`: un número entre 0 y 1 que depende solo del jugador y de `k`, el mismo en cada cuadro y en cada equipo.

- `p.state.<nombre>`: los números que la zona guarda para él (ver Estado entre cuadros).

`pose` acepta `rotation`, `scale`, `sx` y `sy` (aplastar y estirar), `lean`, `look_x` y `look_y` (hacia dónde miran los ojos), `flip`, `visible`, `show_name` y `express` (`"happy"`, `"sad"`, `"hurt"`, `"winner"`, `"surprised"`) con `loop` y `since`.

`gaanim.live` trae ayudas para animar con los principios de la animación: `lerp`, `clamp`, `progress`, `smoothstep`, `ease_in`, `ease_out`, `ease_in_out`, `ease_out_back`, `spring`, `wobble`, `impact` (el rebote al aterrizar), `anticipate` (agacharse antes de saltar), `ballistic` y `landing` (tiros parabólicos). Además, la zona estira a cada personaje en la dirección en que se mueve, lo inclina al correr, balancea orejas y sombreros y mueve sus ojos hacia donde va; `squash`, `lean`, `follow` y `look` en `live_zone` regulan cuánto, y 0 lo apaga.

30.2. Estado entre cuadros

Un comportamiento es una función del tiempo: no recuerda nada del cuadro anterior. Cuando necesitas memoria por jugador (vidas, energía que se carga, peldaños subidos), declárala con `state` y calcula los siguientes valores con `update`:

```
from gaanim.live import STEP, pose, state

scene = Scene(frame=(16, 9))
audience = scene.audience()

def cargar(p):                                  # cada 1/60 s
    energia = p.state.energia + 0.7 * STEP
    if energia >= 1:
        return state(energia=0, peldanos=p.state.peldanos + 1)
    return state(energia=energia)               # peldanos sigue igual

def subir(p):
    return pose(-6 + p.index * 2, -3.5 + 1.2 * p.state.peldanos + 0.4 * p.state.energia)

scene.live_zone(audience, subir, state={"energia": 0.0, "peldanos": 0}, update=cargar)
scene.wait(6)
```

`state` da los números (hasta 8) y dónde empieza cada uno cuando el jugador llega. `update(p)` corre cada `STEP` segundos (1/60) para cada jugador y devuelve `state(...)` con los que cambian; los que no nombra conservan su valor. El comportamiento y la actualización los leen como `p.state.energia`. La zona los calcula paso a paso desde que se abre, así que saltar en la línea de tiempo y exportar dan siempre lo mismo.

30.3. Diferencias con Python

El comportamiento se escribe en Python, pero no se ejecuta con Python: Gaanim lo traduce a instrucciones que corren en Rust, en la presentación, en el reproductor y en la web. Por eso acepta un subconjunto del lenguaje, y algunas cosas se comportan distinto.

30.3.1. Lo que puedes usar

- Números, `True`/`False`, `None` y las cadenas de las expresiones.
- `+ - * / // % **` con las reglas de Python (`-7 // 2` es `-4`, `-7 % 2` es `1`, `round` redondea las mitades al par), comparaciones encadenadas (`0 < x < 1`), `and`, `or`, `not`, `in` sobre tuplas e `is None`.
- `if`/`elif`/`else`, el condicional `a if c else b`, asignaciones normales, aumentadas (`x += 1`) y con desempaquetado (`x, y = ...`), `return`.

- `for i in range(N)` con un `N` fijo (se desenrolla, hasta 256 vueltas).
- Tuplas (una lista escrita entre corchetes se trata como tupla), `len`, `sum`, `min`, `max`, `abs`, `round`, `int`, `float`, `bool` y todo el módulo `math`.
- Tus propias funciones, definidas a nivel de módulo, con parámetros normales y valores por defecto (puedes llamarlas pasando argumentos por nombre); y constantes del módulo.

30.3.2. Lo que no

- `while`, `break`, `continue`, recursión, `try`, `with`, clases, `global`.
- Modificar colecciones (`append`, asignar a `xs[i]`), diccionarios, conjuntos, comprensiones, `lambda` y f-strings.
- Parámetros solo por nombre (`def f(a, *, b)`), `*args` y `**kwargs`.
- Otras bibliotecas (`numpy`, `random`…): usa `math` y `p.random(k)`.
- `p.name` y otras cosas del jugador que no estén en la lista de arriba.

Si escribes algo que no se puede compilar, `scene.live_zone` lanza `gaanim.live.BehaviorError` señalando la línea y explicando qué usar.

30.3.3. Lo que se comporta distinto

- **Las constantes se congelan al compilar.** El comportamiento lee el valor que tenían las variables del módulo al llamar a `scene.live_zone`; cambiarlas después no tiene efecto.
- **Las dos ramas de un `if` se calculan** y se elige una. No hay efectos secundarios que puedan notarse, pero una operación inválida en la rama que no se elige tampoco falla.
- **Los errores son números.** Donde Python lanzaría un error (dividir por cero, `math.sqrt(-1)`, `math.log(0)`), el comportamiento obtiene `inf` o `nan`, como en coma flotante; una pose con `nan` no se dibuja bien, así que protege esos casos con `max` o un `if`.
- **Todos los números son `float`.** `int(x)` trunca, pero el resultado sigue siendo un número de coma flotante: los enteros son exactos hasta 2^{53}.
- **El tiempo es discreto para lo que recuerda.** Las expresiones que empiezan y el estado avanzan en pasos de 1/60 s desde que se abre la zona; la pose, en cambio, se calcula para el instante exacto de cada cuadro.

Los mismos casos se comprueban en las pruebas del proyecto: cada comportamiento de ejemplo se ejecuta en CPython y en Rust con jugadores de muestra y ambos deben dar las mismas poses.

CAPÍTULO

31. Recetario de estilos

Escenas completas para copiar: fichas, encuestas, sala de espera, carreras, batallas y podios

Cada receta es una escena completa que puedes guardar como `main.py`, presentar tal cual y cambiar a tu gusto. Todas usan el ensayo para que la vista previa muestre un juego en marcha; al presentar, los mismos dibujos siguen al público real.

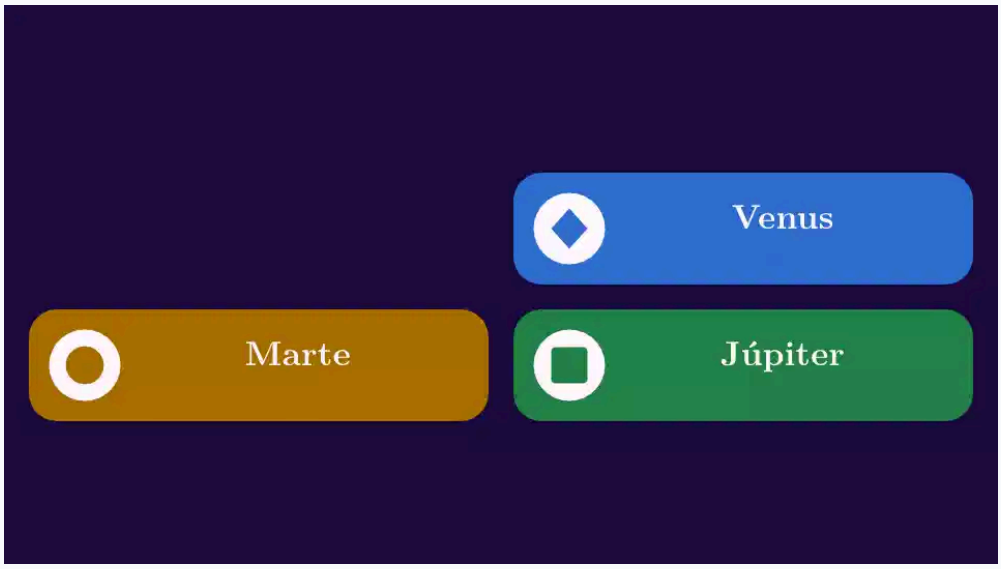
Las recetas se combinan: una trivia completa puede empezar con la sala de espera, seguir con fichas de colores en cada pregunta, pasar por una batalla de equipos y terminar en el podio. Usa un segmento por diapositiva (`scene.segment`) y una pausa donde el presentador hable.

31.1. Fichas de colores

El estilo clásico: cuatro fichas con el color y la forma de cada respuesta en el teléfono, una cuenta atrás y una pausa que avanza sola cuando todos respondieron. Al revelar, las fichas se encogen a una fila con `magic_move`, aparecen los votos y la correcta brilla.

```
from gaanim import Scene, magic_move

INK, WHITE, GOLD = "#1c0b3f", "#fff8ff", "#ffd23f"
scene = Scene(frame=(16, 9), background=INK)
scene.rehearsal(16, seed=4)
quiz = scene.quiz("¿Qué planeta es el más caliente?", ["Mercurio", "Venus", "Marte", "Júpiter"],
                  correct=1, time=20, rehearse=0.45)


def tile(i, width, height, size):
    """Una respuesta con su color y su forma, con nombre para magic_move."""
    box = scene.geometry.rounded_rect(width, height, height / 4).fill(quiz.color(i)).no_stroke()
    x = -width / 2 + height / 2
    badge = scene.geometry.circle(height * 0.32).fill(WHITE).no_stroke().move_to(x, 0)
    icon = quiz.icon(i, height * 0.36).move_to(x, 0)
    label = scene.text(quiz.options[i], size=size, color=WHITE, weight=700, text_box="cap")
    label.move_to(height * 0.35, 0)
    return scene.geometry.group([box, badge, icon, label]).named(f"respuesta{i}")


title = scene.text(quiz.question, size=0.6, color=WHITE, weight=700).move_to(0, 3.4)
clock = scene.viz.readout(quiz.remaining(), format=".0f", color=GOLD, font_size=0.7).move_to(7, 3.4)
tiles = [tile(i, 7.4, 1.8, 0.55).move_to(-3.9 + i % 2 * 7.8, 0.9 - i // 2 * 2.2) for i in range(4)]
scene.play([title.animate.fade_in(), clock.animate.fade_in()]
           + [t.animate.grow_from_center().duration(0.4).delay(0.08 * i).settle()
              for i, t in enumerate(tiles)])
scene.stop("respondiendo", until=quiz.answered(share=1.0) | quiz.time_up())
quiz.reveal()

# Las fichas se encogen a una fila y aparecen los votos.
COLUMNS = [-5.7, -1.9, 1.9, 5.7]
row = [tile(i, 3.6, 0.9, 0.3).move_to(COLUMNS[i], 0.6) for i in range(4)]
before, after = scene.geometry.group(tiles), scene.geometry.group(row)
scene.play([magic_move(before, after).duration(0.8), clock.animate.fade_out()])
reveal = []
for i, x in enumerate(COLUMNS):
    bar = quiz.bar(i, length=2.8, thickness=0.16, radius=0.08, scale="total")
```

```
    bar.fill(WHITE).no_stroke().move_to(x - 0.3, -0.2)
    votes = scene.viz.readout(quiz.votes(i), format=".0f", color=WHITE, font_size=0.3)
    reveal += [bar.animate.create(), votes.move_to(x + 1.45, -0.2).animate.fade_in()]
    if i == quiz.correct:
        reveal.append(row[i].animate.glow(quiz.color(i), radius=0.4, intensity=1.3))
    else:
        reveal.append(row[i].animate.opacity(0.3))
scene.play(reveal)
scene.wait(2)
scene.stop("respuesta")
scene.render()
```



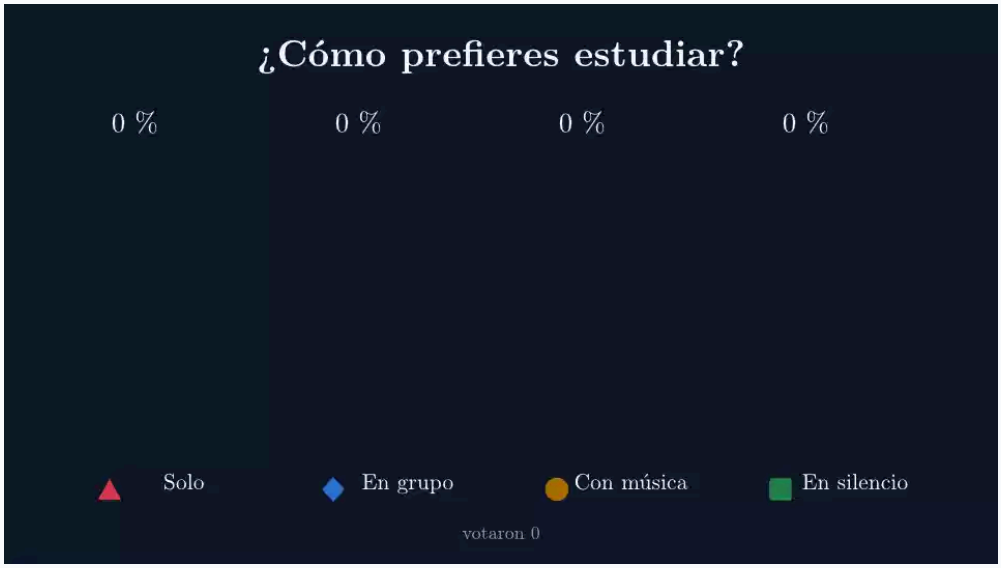
Los teléfonos ordenan y colorean las respuestas de un cuestionario a su manera para que nadie copie al vecino, así que en un cuestionario la ficha de la pantalla no coincide con la del teléfono; en una encuesta sí.

31.2. Columnas de una encuesta

Una encuesta sin respuesta correcta: columnas que crecen con el porcentaje de cada respuesta y el total de teléfonos que votaron.

```
from gaanim import Scene

INK, WHITE, MUTED = "#0f172a", "#f8fafc", "#94a3b8"
scene = Scene(frame=(16, 9), background=INK)
scene.rehearsal(30, seed=2)
poll = scene.poll("¿Cómo prefieres estudiar?", ["Solo", "En grupo", "Con música", "En silencio"],
                  rehearse=[2, 5, 4, 1])

scene.text(poll.question, size=0.6, color=WHITE, weight=700).move_to(0, 3.5)
for i, answer in enumerate(poll.options):
    x = -5.4 + 3.6 * i
    poll.bar(i, length=5, thickness=2.4, direction="up", radius=0.2) \
        .fill(poll.color(i)).no_stroke().move_to(x, -0.4)
    scene.viz.readout(poll.percent(i), format=".0f", suffix=" %", color=WHITE,
                      font_size=0.45).move_to(x - 0.5, 2.6)
    poll.icon(i, 0.4).move_to(x - 0.9, -3.3)
    scene.text(answer, size=0.36, color=WHITE, text_box="cap").move_to(x + 0.3, -3.3)
scene.viz.readout(poll.total(), format=".0f", prefix="votaron ", color=MUTED,
                  font_size=0.3).move_to(0, -4)
scene.wait(4)
```

```
scene.stop("votando")
scene.render()
```



Con el valor por defecto de `scale`, `"leader"`, la respuesta que va ganando llena su columna y las demás miden en proporción, lo que se lee bien aunque voten pocos; con `scale="total"` cada columna mide su porcentaje. Una lectura (`readout`) se ancla por la izquierda para no saltar cuando cambian los dígitos, así que para centrarla bajo algo córrela un poco a la izquierda. `poll.percent(i)` sigue en vivo también al presentar un `.gaanim`.

31.3. Sala de espera con personajes

Mientras el público escanea el QR, cada jugador cae desde arriba como el personaje que armó en su teléfono y rebota al aterrizar. La pausa avanza sola cuando hay al menos 8.

```
from gaanim import Scene
from gaanim.live import ease_in, impact, lerp, pose

INK, WHITE, GOLD = "#1c0b3f", "#fff8ff", "#ffd23f"
scene = Scene(frame=(16, 9), background=INK)
scene.rehearsal(14, seed=5, arrive=5)
audience = scene.audience()

scene.geometry.rounded_rect(3.6, 3.6, 0.3).fill(WHITE).no_stroke().move_to(-5.6, 0.6)
audience.qr(3.1).fill(INK).move_to(-5.6, 0.6)
scene.text(audience.code, size=0.7, color=GOLD, weight=800).move_to(-5.6, -1.8)
scene.viz.readout(audience.count(), format=".0f", suffix=" en la sala", color=WHITE,
                  font_size=0.35).move_to(-6.8, -2.6)


def llegar(p):
    x = -1.8 + p.index % 6 * 1.7
    y = 1.4 - p.index // 6 * 2.0
    caida = ease_in(min(p.t / 0.5, 1))       # cae medio segundo
    sx, sy = impact(p.t - 0.5)               # y se aplasta al tocar el suelo
    return pose(x, lerp(6, y, caida), sx=sx, sy=sy, express="happy", since=0.5)


scene.live_zone(audience, llegar, size=1.1, names=True, name_size=0.22)
scene.wait(7)
```

```
scene.stop("sala", until=audience.at_least(8))
scene.render()
```

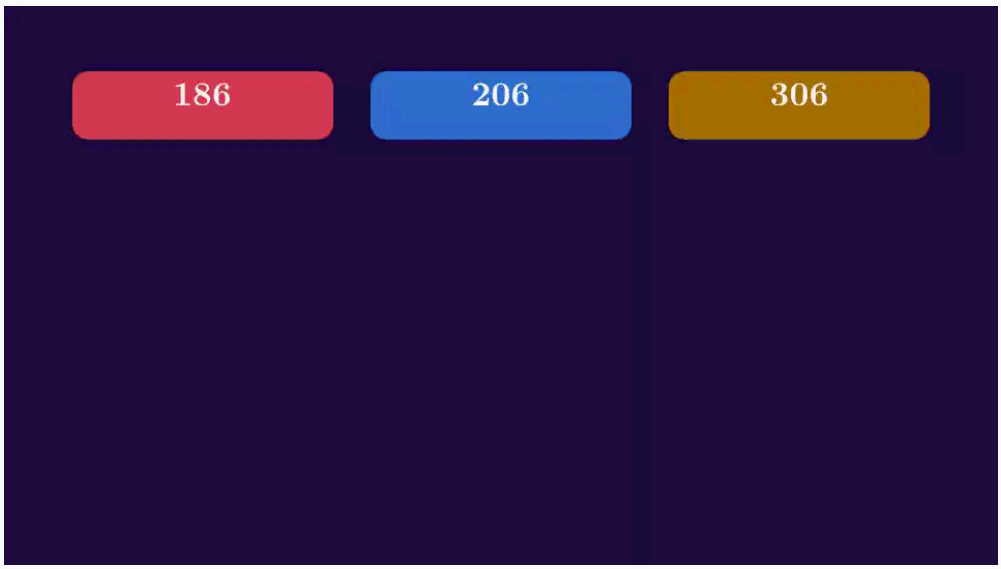
`p.index` es el orden de llegada y `p.t` los segundos desde que llegó, así que cada uno hace su entrada a su tiempo. Con `names=True` la zona dibuja el apodo bajo los pies. Para más movimientos, mira Zonas vivas.

31.4. Cada uno corre a su respuesta

Después de revelar, cada personaje corre bajo la respuesta que eligió: los que acertaron saltan, los demás se lamentan y quienes no respondieron se quedan en una esquina.

```
import math

from gaanim import Scene
from gaanim.live import ease_out_back, impact, lerp, pose, progress, smoothstep

INK, WHITE = "#1c0b3f", "#fff8ff"
COLUMNS = (-4.8, 0, 4.8)
scene = Scene(frame=(16, 9), background=INK)
scene.rehearsal(18, seed=3)
audience = scene.audience()
quiz = scene.quiz("¿Cuántos huesos tiene un adulto?", ["186", "206", "306"], correct=1,
                  rehearse=0.55)
for i, x in enumerate(COLUMNS):
    scene.geometry.rounded_rect(4.2, 1.1, 0.25).fill(quiz.color(i)).no_stroke().move_to(x, 2.9)
    scene.text(quiz.options[i], size=0.55, color=WHITE, weight=700, text_box="cap").move_to(x, 2.9)
scene.wait(1)
scene.stop("respondiendo")
quiz.reveal()


def correr(p):
    if p.answer < 0:                                    # no respondió
        return pose(-7.2 + 0.4 * p.index, -4, scale=0.6, express="sad", loop=True)
    x = COLUMNS[0] + (COLUMNS[1] - COLUMNS[0]) * p.answer + 3 * (p.random(1) - 0.5)
    y = 0.8 - 3.8 * p.random(2)
    salida = 0.4 * p.random(3)
    carrera = progress(p.t, salida, 0.8)
    if carrera < 1:                                     # corre desde abajo
        return pose(lerp(16 * p.random(4) - 8, x, ease_out_back(carrera)),
```

```
                    lerp(-6, y, smoothstep(carrera)) + math.sin(math.pi * carrera),
                    express="surprised", since=0)
    llego = p.t - salida - 0.8
    if p.answer_points > 0:                              # acertó: salta sin parar
        beat = (llego + p.random(5)) % 0.8 / 0.8
        return pose(x, y + 1.6 * beat * (1 - beat), express="winner", loop=True)
    sx, sy = impact(llego, amount=0.4)
    return pose(x, y, sx=sx, sy=sy, express="sad", since=salida + 0.8)


scene.live_zone(audience, correr, size=0.95)
scene.wait(4)
scene.stop("respuesta")
scene.render()
```



`p.answer` es la respuesta que eligió (-1 si ninguna) y `p.answer_points` lo que ganó con ella; en selección múltiple, `p.answer_mask` tiene un bit por respuesta elegida. `p.random(k)` da a cada jugador un número propio y fijo, para que no corran todos igual.

31.5. Batalla de equipos

Dos equipos tiran de una cuerda: el que va ganando la arrastra hacia su lado. Las barras de arriba comparan el promedio de puntos por jugador, más justo si los equipos quedaron desparejos.

```
import math

from gaanim import Scene
from gaanim.live import ease_out_back, pose

INK, MUTED = "#1a1033", "#cbb8ff"
scene = Scene(frame=(16, 9), background=INK)
teams = scene.teams(["Rojo", "Azul"], colors=["#ff4f8b", "#2fb8ff"])
scene.rehearsal(16, seed=2, skill=[0.85, 0.4])
audience = scene.audience()

# Aquí van tus preguntas; el ensayo las responde y suma los puntos.
for question, options, correct in [("¿7 × 8?", ["54", "56", "64"], 1),
                                   ("¿Capital de Perú?", ["Lima", "Cusco", "Puno"], 0)]:
    quiz = scene.quiz(question, options, correct=correct)
    scene.wait(0.5)
    scene.stop()
```

```
    quiz.reveal()

scene.segment("Batalla")


def side(p):
    return -1 if p.team == 0 else 1


def tirar(p):
    x = side(p) * (1.6 + p.team_index % 5 * 0.95)
    y = -0.6 - p.team_index // 5 * 1.25
    adelante = p.team_rank == 0
    arrastre = 1.4 * ease_out_back(min(p.time / 1.2, 1)) * (1 if adelante else -1)
    tiron = 0.15 * math.sin(2 * math.pi * (p.t * 1.4 + p.team_index * 0.13))
    return pose(x + side(p) * (arrastre + tiron), y, lean=side(p) * 0.25, flip=side(p) > 0,
                express="winner" if adelante else "hurt", loop=True)


scene.geometry.rect(15, 0.12).fill("#c89b6d").no_stroke().move_to(0, -0.2)
for team, x in [(0, -4.5), (1, 4.5)]:
    scene.text(teams.names[team], size=0.7, color=teams.colors[team], weight=800).move_to(x, 3.4)
    teams.bar(team, length=5.5, thickness=0.35, radius=0.17,
              direction="left" if team == 0 else "right") \
        .fill(teams.colors[team]).no_stroke().move_to(x, 2.5)
    scene.viz.readout(teams.average(team), format=".0f", suffix=" pts por jugador", color=MUTED,
                      font_size=0.28).move_to(x - 1.3, 1.9)
scene.live_zone(audience, tirar, size=0.9, lean=0.0)
scene.wait(5)
scene.stop("batalla")
scene.render()
```

`p.team_rank` es 0 para el equipo que va ganando y `p.team_index` el orden de llegada dentro de su equipo, que sirve para formar filas en cada lado. Con `choose=True` en `scene.teams`, cada uno elige su equipo en el teléfono.

31.6. Carrera de puntajes

Los cinco primeros corren por carriles: avanzan hasta su puntaje, cambian de carril cuando alguien los adelanta y celebran una racha de aciertos. El puntaje que muestran sube poco a poco gracias al estado entre cuadros.

```
import math

from gaanim import Scene
from gaanim.live import STEP, ease_out_back, lerp, pose, spring, state

INK, SEA, MUTED, GOLD = "#1c0b3f", "#4a2a8c", "#cbb8ff", "#ffd23f"
scene = Scene(frame=(16, 9), background=INK)
scene.rehearsal(12, seed=8)
audience = scene.audience()
board = scene.leaderboard()

for question, options, correct in [("¿Símbolo del oro?", ["Ag", "Au", "Or"], 1),
                                   ("¿Año de la llegada a la Luna?", ["1965", "1969", "1972"], 1),
                                   ("¿Qué significa PDF?", ["Portable Document Format",
                                                            "Printed Data File"], 0)]:
    quiz = scene.quiz(question, options, correct=correct)
    scene.wait(0.5)
    scene.stop()
    quiz.reveal()

scene.segment("Carrera")
LANES, START, LENGTH = 5, -5.2, 10.4


def lane(rank):
    return 2.3 - 1.3 * rank


def contar(p):                          # el puntaje mostrado persigue al real
    return state(mostrado=p.state.mostrado + (p.score - p.state.mostrado) * min(1, 2.5 * STEP))


def correr(p):
    y = lerp(lane(p.previous_rank), lane(p.rank), ease_out_back(p.rank_since / 0.6))
    meta = START + LENGTH * p.state.mostrado / max(p.leader, 1)
    x = lerp(START - 1.2, meta, spring(p.t, frequency=0.9, damping=0.45))
    paso = abs(math.sin(2 * math.pi * 1.6 * p.t))
    animo = "winner" if p.streak >= 3 else ("happy" if p.rank == 0 else None)
    return pose(x, y + 0.16 * paso, express=animo, loop=True, visible=p.rank < LANES)


scene.text("Así va la carrera", size=0.6, color="#fff8ff", weight=700).move_to(0, 3.7)
for rank in range(LANES):
    y = lane(rank)
    scene.geometry.rect(LENGTH + 0.6, 0.07).fill(SEA).no_stroke().move_to(START + LENGTH / 2, y - 0.04)
    board.name(rank, size=0.28, weight=700).fill(MUTED).move_to(-6.9, y + 0.35)
    scene.viz.readout(board.points(rank), format=".0f", color=GOLD, font_size=0.28).move_to(6.2, y +
0.35)
scene.live_zone(audience, correr, size=0.85, state={"mostrado": 0.0}, update=contar)
scene.wait(5)
scene.stop("carrera")
scene.render()
```

`p.rank`, `p.previous_rank` y `p.rank_since` (segundos desde el último cambio de puesto) animan los adelantamientos; `p.leader` es el puntaje del primero, para medir la pista. `state` y `update` guardan un número por jugador entre cuadros (ver Estado entre cuadros).

31.7. Podio

Los tres primeros suben a su escalón y el ganador salta de alegría mientras llueve confeti; el resto aplaude desde abajo.

```
import math

from gaanim import Scene
from gaanim.live import anticipate, impact, pose

INK, WHITE, GOLD = "#1c0b3f", "#fff8ff", "#ffd23f"
scene = Scene(frame=(16, 9), background=INK)
scene.rehearsal(10, seed=6)
audience = scene.audience()
board = scene.leaderboard()
for question, options, correct in [("¿Planeta con anillos?", ["Marte", "Saturno"], 1),
                                   ("¿Autor de «La noche estrellada»?", ["Van Gogh", "Monet"], 0)]:
    quiz = scene.quiz(question, options, correct=correct)
    scene.wait(0.5)
    scene.stop()
    quiz.reveal()

scene.segment("Podio")
PLACES = ((0.0, 0.2, 3.6), (-3.6, -0.8, 2.6), (3.6, -1.4, 2.0))   # x, arriba, alto
MEDALS = (GOLD, "#d7dbe8", "#e39b5f")


def celebrar(p):
    if p.rank < 3:
        x, y, _ = PLACES[p.rank]
        if p.rank > 0:
            return pose(x, y, scale=1.3, lean=0.08 * math.sin(2 * math.pi * p.t / 1.3), look_y=0.6)
        beat = p.t % 1.3                                  # agacharse, saltar, aterrizar
        sx, sy = anticipate(beat, at=0.35, amount=0.3, duration=0.35)
        alto = 0.0
        if 0.35 <= beat < 0.9:
            u = (beat - 0.35) / 0.55
```

```
            alto = 4.4 * u * (1 - u)
        elif beat >= 0.9:
            sx, sy = impact(beat - 0.9, amount=0.35)
        return pose(x, y + alto, sx=sx, sy=sy, scale=1.4, express="winner", loop=True)
    x = -6.3 + (p.rank - 3) % 8 * 1.8
    return pose(x, -4.0, scale=0.7, look_x=-x / 4, look_y=1)


heading = scene.text("¡Campeones!", size=0.8, color=WHITE, weight=800).move_to(0, 3.5)
steps = []
for rank, (x, top, height) in enumerate(PLACES):
    block = scene.geometry.rect(2.8, height).fill(MEDALS[rank]).no_stroke().move_to(x, top - height /
2)
    name = board.name(rank, size=0.36, weight=700, align="center").fill(INK).move_to(x, top - 0.5)
    points = scene.viz.readout(board.points(rank), format=".0f", suffix=" pts", color=INK,
                               font_size=0.3).move_to(x - 0.6, top - 1.0)
    steps.append(scene.geometry.group([block, name, points]).shift_by(0, -8))
scene.play([step.animate.shift_by(0, 8).duration(0.6).delay(0.2 * (2 - rank)).settle(overshoot=0.1)
            for rank, step in enumerate(steps)])
scene.live_zone(audience, celebrar, size=1.1)
confetti = scene.fx.confetti((0, -4.6), count=0, seed=21, spread=1.2, speed=(8, 12))
scene.play([confetti.animate.burst(110).duration(1.2), heading.animate.indicate().duration(0.6)])
scene.wait(2)
scene.stop("podio")
scene.render()
```

Al terminar la presentación, los teléfonos también muestran el podio y una despedida.

31.8. Más ideas

- **Sonidos**: cualquier animación lleva un sonido con `.sound(ruta)`, y `Transition.magic_move(..., sound=)` suena al cambiar de diapositiva. Los paquetes de efectos de Kenney son de dominio público (CC0).
- **Un dato entre preguntas**: después de revelar, una `scene.viz.bar_race` cuenta la historia detrás de la respuesta.
- **Un anfitrión**: `scene.character(...)` pone en la escena un personaje como los del público, que celebra con `express("winner")` cuando quieras.
- **Transiciones que continúan**: nombra las piezas que se repiten en cada diapositiva (el logo, el marcador de equipos) con `.named()` y usa `Transition.magic_move` en `scene.segment` para que se muevan en lugar de aparecer de nuevo.

- **Partidas de verdad**: el ejemplo `examples/showcase/trivia/` del repositorio junta todas estas recetas en una trivia completa con preguntas en Markdown, sonidos e imágenes.

CAPÍTULO

32. Presentar con público

La vista del presentador, partidas nuevas, paquetes .gaanim y qué hacer si algo falla

En esta guía verás qué pasa cuando presentas una escena con público: qué te muestra la vista del presentador, cómo empezar una partida nueva, qué sigue en vivo en un paquete `.gaanim` y cómo resolver los problemas más comunes.

32.1. Antes de la charla

1. Comprueba el relay con `gaanim relay`: debe decir `up to date`.
2. Revisa la escena con `gaanim check` y recórrela con la vista previa, donde juega el ensayo.
3. Presenta una vez con dos o tres teléfonos tuyos, con datos móviles y con la red del lugar si puedes: escanea el QR, únete, responde y revela.
4. Antes de empezar, una partida nueva (`R` dos veces) deja la sesión limpia.

32.2. La vista del presentador

La vista del presentador tiene una sección **Audience** con el código de la sesión, los teléfonos conectados y los jugadores con sus puntos. Desde ahí puedes quitar a un jugador (su teléfono ya no puede volver a unirse) o empezar una partida nueva con **New game**, que borra votos, respuestas y jugadores y pide una segunda pulsación para confirmar. Pulsar `R` dos veces seguidas hace lo mismo desde cualquier ventana de la presentación, y `gaanim relay reset [ruta]` lo hace desde la terminal.

En una pausa con `until=`, la vista del presentador muestra cuánto falta para que avance sola; siempre puedes avanzar antes.

Al terminar la presentación los teléfonos muestran el podio y una despedida, y la siguiente presentación empieza una partida nueva. Si la presentación se cae a mitad de la charla, al volver a abrirla la partida sigue donde estaba. Si pierde la conexión con el relay, la presentación sigue y la terminal avisa mientras reintenta.

32.3. Guardar los resultados

Cada presentación con público guarda sola los resultados de la partida, para que puedas revisarlos o poner notas después. Van a `resultados/` dentro de la carpeta de salida del proyecto (`exports/` por defecto), o junto al paquete `.gaanim` que presentas, en una carpeta por partida con la fecha y hora en que empezó:

```
exports/resultados/2026-10-01_15-30/
├── jugadores.csv     un jugador por fila: puesto, puntos, aciertos y cada respuesta
├── respuestas.csv    una fila por respuesta: pregunta, respuesta, si acertó, puntos
y segundos
```

```
└── preguntas.csv     una fila por pregunta: cuántos respondieron, cuántos acertaron
y los votos
```

Los archivos se actualizan mientras llegan respuestas, a lo sumo cada cinco segundos, y una vez más al terminar la presentación, así que una presentación que se cierra de golpe también los deja. Son CSV en UTF-8, que Excel, Google Sheets y LibreOffice abren con las tildes bien. En la vista del presentador, la sección **Audience** dice cuándo se guardaron por última vez, tiene **Save now** para guardarlos al instante y **Open folder** para abrir la carpeta. Si tienes un archivo abierto en Excel mientras presentas, Excel no deja reemplazarlo: la vista del presentador lo avisa y se guarda en cuanto lo cierras.

Si la presentación se cerró antes de guardarlos, el relay conserva la partida doce horas y la descargas con:

```
gaanim relay results            # en la carpeta del proyecto
```

32.3.1. Saber quién es quién

Los apodos son libres. Para que cada fila tenga el código o el nombre real del alumno, pídelo al unirse con `scene.roster`:

```
scene.roster("Código de alumno")    # o "Nombre y apellido", "Correo"…
```

El teléfono muestra ese campo junto al apodo y no deja unirse sin él (con `required=False` es opcional). El dato nunca llega a la pantalla ni a los demás teléfonos: solo aparece, como una columna más, en los resultados guardados.

32.4. Los teléfonos

La página de votación funciona en cualquier navegador moderno, sin instalar nada. Muestra la pregunta en cuanto la presentación llega a ella, una cuenta atrás en los cuestionarios y, al revelar, si el jugador acertó, sus puntos y su puesto. Para que nadie copie, cada teléfono ordena las respuestas de un cuestionario a su manera y no muestra el puntaje hasta la revelación.

Los apodos admiten de 2 a 20 letras, números o espacios y no se repiten en la sesión. Si el teléfono se bloquea o pierde la señal, vuelve a la pregunta abierta al regresar. En redes que bloquean WebSockets, la página pregunta por HTTP cada pocos segundos.

32.5. Paquetes .gaanim

Un paquete `.gaanim` exportado se presenta sin Python y también recibe votos en vivo, pero el paquete reproduce lo que se grabó al exportar. Siguen al público real:

- las barras de `poll.bar`, `board.bar` y `teams.bar`;
- los apodos de `board.name` y `audience.name`, cuyos glifos guarda el paquete;
- las lecturas que muestran directamente un valor del público, como `scene.viz.readout(poll.votes(0))`, `poll.percent(0)` o `teams.score(1)`;
- las zonas vivas, cuyos comportamientos se compilan dentro del paquete;
- las pausas con `until=`.

Lo que pase por un `computed` de Python muestra lo que se grabó con el ensayo: si quieres que algo siga en vivo en un paquete, usa directamente el `Parameter`.

32.5.1. En el reproductor web

El reproductor web también presenta un paquete con encuestas: al presentar, la página de la presentación se conecta al relay que se grabó en el paquete y los teléfonos votan como con la aplicación. El Presenter View muestra los teléfonos y los jugadores, su diapositiva sigue los votos de la sala como la de la audiencia, y desde él puedes quitar a un jugador o empezar una partida nueva. Los resultados no se guardan solos en una carpeta: **Download results** descarga un ZIP con `jugadores.csv`, `respuestas.csv` y `preguntas.csv`.

Necesita un relay de Gaanim 0.8 o posterior: si el tuyo es anterior, actualízalo con `gaanim relay init --force <su carpeta>` y `npx wrangler deploy`. Los votos de la sesión anterior no se pierden.

El primer navegador o computadora que presenta una sesión se queda con ella: el relay solo acepta su clave. Si la sesión ya la presentó la aplicación en tu computadora, pásale su clave al navegador con un enlace:

```
gaanim relay key mi-charla.gaanim --src https://tu-sitio.com/mi-charla.gaanim
```

El enlace que imprime termina en `#clave=…`. La página la guarda y la quita de la barra de direcciones. Quien tenga la clave controla las encuestas de la sesión: no la compartas con el público.

32.6. Privacidad y límites

Los votos son anónimos: el teléfono guarda un identificador al azar y el relay no pide nombres ni cuentas; un cuestionario solo pide un apodo, y lo que pida `scene.roster`. Los resultados guardados tienen los apodos, ese dato y las respuestas de cada jugador: trátalos como las notas de tu curso. Los votos de quien no se unió con un apodo solo cuentan en los totales. Lo que cada teléfono votó lo recuerda el relay, no el teléfono, así que una partida nueva empieza limpia en todos. El relay borra la sesión y sus votos doce horas después de su última actividad.

Para que nadie con el código pueda inflar una sesión, una partida admite 500 jugadores y una encuesta 1000 teléfonos, y cada teléfono puede enviar unos pocos mensajes por segundo. Una sala de unos cientos de teléfonos entra con holgura en el plan gratuito de Cloudflare.

32.7. Si algo falla

Síntoma	Qué hacer
El QR no lleva a ninguna parte	La escena se creó sin relay. Configúralo con `gaanim relay use <URL>` y vuelve a cargar o exportar: la dirección queda grabada en el QR.
La terminal dice que el relay está desactualizado	Escríbelo de nuevo con `gaanim relay init --force <carpeta>` y despliégalo otra vez con `npx wrangler deploy`.
Los teléfonos no ven la pregunta	Comprueba que la presentación esté dentro de la encuesta (entre que se abre y su cierre) y que la terminal no avise de un relay inaccesible.
Un teléfono dice que el apodo está ocupado	Alguien ya lo usa en esta sesión; los apodos no distinguen mayúsculas.

Quedaron jugadores de un ensayo anterior	Empieza una partida nueva con `R` dos veces o `gaanim relay reset`.
La vista previa no se parece a la sala	Ajusta `scene.rehearsal`: cuántos jugadores, qué tan bien responden (`skill`) y qué tan rápido (`speed`).
Probar contra un relay local va lento en Windows	Usa `http://127.0.0.1:8787` en vez de `localhost`, que primero intenta IPv6.

PARTE V

Ejemplos

Escenas completas listas para copiar y ejecutar

CAPÍTULO

33. Examples

Every example in the repository, by topic: complete scenes ready to run

Each example covers one subject: it shows the variants the API offers for it, with the code that builds each one on screen.

Basics

Scenes, objects, play and wait, groups and segments: the first things to learn.

- **First scene**: A first scene: objects on a stage, and play, wait and immediate changes moving the timeline cursor, drawn as a timeline.
- **Scene tour**: A tour of the scene: the factories that make shapes, text, equations, code, layouts, plots, SVGs and readouts, and the animations that change them.
- **Groups**: Groups that move, turn and animate as one, nested arms that turn about their joints, and groups lined up or repeated.
- **Segments**: Segments as chapters of a scene: notes, reuse, persistent objects, release, stops and markers.

Shapes

Geometry primitives, curves, arrows, strokes, boolean operations and clipping.

- **Primitives**: Every closed shape, line, arc, polyline and diagram mark that scene.geometry draws.
- **Curves**: Quadratic and cubic Bézier curves with their handles, composed curve commands and parametric polylines, one redrawn every frame.
- **Arrows**: Straight, double and curved arrows with their sizes, grow_arrow against create, tips on any stroke and connectors that follow moving boxes.
- **Strokes**: Stroke width and alignment, caps, joins and miter limits, dash patterns that march, tapered width profiles and trim paths.
- **Boolean operations**: Union, intersection, difference and xor of overlapping shapes, live results that follow their operands, fill rules and tolerance.
- **Clipping**: Shapes, groups and text clipped to moving vector masks, inverted or with the even-odd rule, and fills that rise to a level from every side.
- **Path modifiers**: Zig zag, round corners, pucker and bloat, twist, wiggle and offset modifiers, animated, stacked and applied to text and trims.

Style

Colors, gradients, blend modes, themes and backgrounds.

- **Colors**: CSS color strings, Color constructors, named constants, the theme's color roles and the Tailwind palette.

- **Brushes**: Solid, linear, radial and sweep brushes, their extend modes, gradient text and strokes, and gradients that animate.
- **Color maps**: Built-in continuous and categorical color maps, sampling, custom, reversed and translucent maps, and data colored by value.
- **Blend modes**: Every blend mode of a drawable over the same backdrop, and blended groups, member overrides and text.
- **Themes**: The palette of every built-in theme and its aliases, side by side on the same small composition.
- **Custom themes**: A Theme derived from a built-in one with its own colors, fonts, text roles, style classes, spacing tokens and validation.
- **Backgrounds**: The scene background, solid and gradient segment backgrounds, dot grids, mesh gradients, auroras and WGSL shader backgrounds.

Text

Typography, code blocks, Typst layouts and animated text.

- **Typography**: Text roles, style keywords, spacing, styled parts, inline markup and math, paragraph alignment and wrapping, anchors and selections.
- **Code blocks**: Code panels in several languages, with highlighting off, a .tmTheme, theme color options, another monospaced font and different sizes.
- **Text writing**: write() by grapheme, word, line and part, in every order and stagger, and the chalk brush with its seed and roughness.
- **Text reveals**: Masked slide, fade, scale and blur reveals by unit, blur-in and tracking, selection reveals, and the matching conceal and unwrite exits.
- **Text animator**: The text range animator: every selector shape, unit order, animated channel, unit type and sweep range.
- **Typewriter**: Typewriter speed, jitter, cursors, blinking and kept cursor, then backspace and retype edits of a terminal command.
- **Scramble**: Scramble decoding with every charset, reveal delay, speed and seed, and scramble_to into new text.
- **Text on path**: Text along closed and open paths with align, orient, reverse and offset, and path_offset animations that spin and slide it.
- **Typst layouts**: Typst documents as drawables: tables with merged cells and computed fills, grids and columns, aligned math, page widths and a .typ file.
- **Typst packages**: Typst Universe packages in a scene: zebraw code listings and simple-plot function plots, styled with the theme's colors.

Math

Equations, step-by-step derivations, annotations and matrices.

- **Equations**: Typst math notation, display and inline equations, quantities with units, sizes and aligned multi-line equations.
- **Equation parts**: Named and nested equation parts, selections by name or fragment, per-term style and writing an equation part by part.
- **Equation annotations**: Emphasis, braces, leader-line notes and hand-drawn marks on single terms of an equation.

- **Derivations**: Step-by-step derivations: structural transforms between equations, cancelled, revealed, copied and moved terms, and a worked solution.
- **Matrices**: Matrix delimiters, row and column labels, spacing, cell content and entry and label styles.
- **Matrix selections**: Matrix row, column, block, diagonal and predicate selections, traversal orders and cell animations.
- **Matrix edits**: Inserting, removing, swapping and reordering matrix rows and columns, setting entries, and morphing one matrix into another.
- **Matrix operations**: Exact matrix sums, products, inverses, determinants and LU, QR and eigen decompositions computed with SymPy.

Layout

Boxes, rows, grids, zones and animated reflow.

- **Boxes**: The box model: padding in every CSS shorthand, backgrounds, borders, radii, shadows, clipping and size rules.
- **Box styles**: Reusable styles with BoxStyle, .but() and classes, typography inherited by box text, text_box and components.
- **Flex**: Rows and columns with every justify and align value, gaps, wrapping, grow, shrink, basis, margins and absolute children.
- **Grids**: Grid tracks in px, %, fr, auto and scene units, gaps, spans, explicit cells, column flow and alignment in cells.
- **Stacks**: Stacked layers placed by the nine anchors and offsets, every fit mode, and draw order with z_index.
- **Zones**: Zones templates, nesting and inset, place() with anchor, padding, offset and fit, arrange(), fill() and animated moves between zones.
- **Positioning**: Free objects placed with to_edge, to_corner, next_to, align_to, move_to and linear layout constraints with priorities.
- **Reflow**: Animated structure changes: add, remove, replace, set, item, move_child, swap, reverse, detach, moves between boxes and reflow after a scale.
- **Box reveal**: Revealing a box piece by piece with reveal, cascade and stagger, and changing pieces with find, find_all and each.
- **Scatter**: Seeded scatter of formulas around a character's parts, with seeds, gaps, avoided zones and object regions.

Animation

Easing, keyframes, timing, motion paths and procedural motion.

- **Easings**: Every Easing preset, curve family, overshoot mode, spring and shaping factory, racing side by side.
- **Entrances exits**: Every built-in entrance and exit: fades from each direction, stroke drawing, growth from a center, point or edge, and the exits that hide a drawable.
- **Properties**: Every property target of drawable.animate: position, transforms, style, polygon points and custom animations computed by your own function.
- **Keyframes**: Keyframes on every channel, an easing per segment between stops, and linear versus Catmull-Rom paths through the stops.

- **Timing**: Repeats and loops, parallel, sequence and stagger, reshaping a composition's time, and staggered waves and distributed values from every origin.
- **Choreography**: Labels and relative inserts drawn as a timeline from Composition.schedule(), and animations cued on beats and bars of a tempo under a launched sweep.
- **Motion paths**: Travelling along circles, waves and arrows with move_along and its options, and straight moves bent into arcs with path_arc.
- **Secondary motion**: Follow-through with settle, trailing followers, squash and stretch, and exact light physics with throw and inertia.
- **Procedural motion**: Oscillators on every channel and waveform, wiggle, the other Updater presets, seeded randomness and smooth noise.

Transforms

Morphing one object into another, matching parts and pivots.

- **Transform to**: Morphs between shapes and their mirror images, open strokes, solid and gradient fills, transform_to against replacement and fade transforms, fading while morphing, and text to text, equation and shape.
- **Matching**: Matching transforms that pair pieces by shape, place and color, glyph matching between words and equations, and group and SVG morphs that pair parts by name or id.
- **Magic move**: magic_move by name, by SVG id and by a key function, fading or cutting unmatched members, and a leaderboard reordered and carried across a cut with Transition.magic_move.
- **Pivots**: Rotations, scales and skews about a drawable's center, a scene point or a moved pivot, hinged groups, orbits and SVG parts, and shapes that aim at a moving target with look_at.

Emphasis

Drawing attention: indications, highlights and hand-drawn notes.

- **Indications**: Indicate, wiggle, blink, passing flashes, broadcast ripples, flashing frames and underlines, and the emphasis of one word inside a text.
- **Highlights**: Live surrounding frames that follow and retarget, color-cycling boundaries, highlighter marker bands and spotlights that dim everything else.
- **Hand drawn**: Hand-drawn underlines, boxes, circles, strikes and brackets with every roughness, pass count, seed and style, marks that follow moving targets, and chalk shapes.

Effects

Per-object glow, blur, shadow, echo, mattes, shaders and particles.

- **Glow blur shadow**: Glow, blur and drop shadow: their parameters, the drawables they apply to, and how they animate.
- **Echo**: Trails of delayed copies with their count, delay and decay, what they replay, held trails and onion skins.
- **Mattes**: Track mattes in alpha and luma modes, inverted, animated and removed, and metaballs whose drops melt together.
- **Glass**: Frosted and liquid glass over what is drawn behind, every refraction, bevel, twist and dispersion setting, and backdrop blur.

- **Object shaders**: WGSL passes on single drawables with uniforms, every PostProcess preset as an effect, chains and margins.
- **Audio shaders**: Per-object shader effects driven by a music track's bands, level, pulses and spectrogram.
- **Falloff**: Falloffs that give each member of a group its own scale, turn, opacity, position or color: index, distance, noise, sources and their combinations.
- **Repeaters**: Copies stepped by a transform with repeat, copies placed by a grid, circle, path or random distribution, and animated copy counts.
- **Particles**: Particle emitter shapes, motion, looks, bursts and confetti, exact at any seek.
- **Plexus**: Live lines between drifting points, by range, nearest neighbors or sequence, with and without distance fading.

Post-processing

Full-frame passes: color grading, bloom, motion blur and finishes.

- **Color grading**: Exposure, contrast, saturation and temperature grades and a 3D lookup table, applied to the whole frame.
- **Finishes**: Vignette, grain, chromatic aberration, halftone, dither, CRT, pixelate and glitch passes, each ramped in through a Parameter, with labels kept out of the post-process.
- **Bloom**: Bloom threshold, intensity and radius on neon outlines, and a glow that pulses with a Parameter.
- **Motion blur**: Film-shutter motion blur that smears dots by speed, spinning, scaling and orbiting shapes, with one dot kept sharp, and segments with a shutter of their own.
- **Post chains**: Chained passes in both orders, the scene post-process with segment overrides, and a custom WGSL pass whose uniform is a number, a Parameter or noise.

Transitions

Transitions between segments and content reused across them.

- **Cuts and fades**: Cuts, cross-fades, fade-through and zoom-through, with flash and light-leak overlays and a transition sound.
- **Reveals**: Vector reveals: wipes, clock wipes, iris shapes, blinds, push and slide, with their directions and a spring easing.
- **Morph transitions**: Drawables that travel across the cut with Transition.morph pairs declared through scene.link, and Transition.magic_move by name.
- **Shader blends**: Shader transition presets with their settings, a luma reveal from a grayscale image, and a custom WGSL swirl with uniforms.
- **Segment reuse**: Content across segments: reuse through a cross-fade, a persistent badge, release, and a transition added later with scene.link.

Camera

Pan, zoom and rotation, camera paths, insets and parallax.

- **Camera moves**: Pan, zoom and rotate the camera, frame objects with margins, and follow a moving target.
- **Camera states**: Reusable camera poses: state_2d, capture, to, and named save and restore.

- **Camera shots**: Drive the camera along a path, whip pan with motion blur, and shake it with the trauma model.
- **Camera bindings**: Bind the camera's center, zoom and rotation to drawables and reactive values, blended by influence and switched on and off.
- **Camera insets**: Insets that enlarge a detail in a screen joined to it: placement, shapes, view moves, following and fixed screens.
- **Camera views**: Shapes that show what a second camera sees: frames, fit modes, lenses and X-ray view layers.
- **Parallax**: Parallax layers at several depths under pan, zoom and rotation, and depth of field with a rack focus.

3D

Primitives, extrusion, axonometric drawings and 3D axes.

- **Primitives 3d**: Cubes, spheres, cylinders, cones and planes as lit meshes, with their segments, rings and caps.
- **Materials 3d**: Matte, metal and emissive materials, roughness, metallic and emission settings, and animated materials and fills.
- **Transforms 3d**: Turning, moving and scaling drawables in depth with the _3d methods, about their center or a pivot.
- **Extrusion**: Text, equations, SVGs, shapes with holes and groups extruded into lit solids, with depth, bevel, tolerance and materials.
- **Lines 3d**: Polylines through 3D points with solid colors and color maps, and paths traced behind moving objects that fade, cap or thin.
- **Axes 3d**: 3D axes with their grid planes, box sizes, linear, log, symlog and power scales, and layers that animate apart.
- **Plots 3d**: Surfaces, parametric curves and vector fields in a 3D space, static and driven by parameters.
- **Axonometric**: A 3D model drawn as flat shapes from isometric, dimetric, plan, front and side views, turned and morphed between views.
- **Camera 3d**: The 3D camera: look_at, orbit, dolly and dolly zoom, perspective and orthographic lenses, saved poses, state_3d and bind_3d.

Visualization

Axes, plots, charts, vector fields and live readouts.

- **Axes**: Linear, log, symlog, power, category and time axes, tick steps, number formats, styles, titles and crossings.
- **Coordinate spaces**: Cartesian, polar and complex planes, number lines, grids and numbers, coordinates that place objects and animated views.
- **Plots**: Function, parametric and implicit curves, log axes, Riemann sums and other calculus constructions, and curves that follow parameters.
- **Chart marks**: Every chart mark: points, lines, steps, areas, bars, histograms, box plots, violins, error bars, heatmaps, and pies, donuts and roses.
- **Chart encodings**: Color, size, opacity and label channels, scales, categorical palettes, legends and color bars, and explicit axes.

- **Chart transitions**: Charts that morph between specs by key or index, change mark family or data, grow from 2D into 3D, and pies that share out their slices anew.
- **Bar race**: Ranked bars that grow, overtake each other and leave the top, with value formats, colors and a rolling ticker.
- **Vector fields**: Vector fields as arrows and streamlines with their options, flowing highlights, particles, advection and color maps.
- **Readouts**: Rolling counters in every format and mode, live readouts of parameters, functions and time, progress rings and countdowns.

Reactivity

Parameters, computed values, updaters and objects that follow each other.

- **Parameters**: Parameters animated, cut and keyframed, a Variable that shows itself, computed values and chains, every setter that follows a source, and what ends a binding.
- **Updaters**: Per-frame Python updaters, stateless and simulated, sampled series on every channel, stacked and removed presets, and arcs redrawn from a Parameter.
- **Connectors**: Reference points, followers, coordinate and rotation bindings, look_at, and lines, labels and frames that keep up with moving objects.
- **Curve trackers**: A point, tangent, normal and osculating circle riding polylines, Bézier curves, arcs and polygons, and traced paths that fade, cap or thin their samples.
- **Circle to sine**: One angle turns a point on a circle while traced paths draw its sine and cosine waves, with live readouts.

Mechanics

Mechanisms, transmissions, supports, forces and simulations.

- **Supports**: Every support kind, the directions they face, their sizes and their styleable parts.
- **Dimensions**: Linear dimensions with symbols and readouts, the side they sit on, their units, formats, style and parts.
- **Angle dimensions**: Angles between rays with symbols and readouts, in every sweep, with their arrowheads, extensions and labels.
- **Forces**: Force vectors with readouts, forces from a magnitude or from components, moments, coordinate frames and a free body diagram.
- **Mechanisms**: Bars and joints between moving points, a slider-crank and a four-bar linkage with a traced coupler point.
- **Springs**: Helical springs between moving endpoints, with their coils, amplitude, crossings and straight ends, in chains and on levers.
- **Transmissions**: Gears and their options, gear trains, racks and pinions, cams with followers and contact on curves.
- **Simulations**: A damped pendulum, a double pendulum and a mass on a spring, stepped at a fixed rate and exact at any seek.

Audio

Sound effects, music-reactive animation and narration.

- **Music**: Audio clips played at the cursor with their length, volume and fades, and animations cut on the beats and bars of a track's tempo.

- **Sound effects**: Sounds anchored to an animation with volume, delay and offset, sounds placed at a second, and sounds played by transitions and staggers.
- **Signals**: A track's loudness, bands, hits and beats as signals, and the sizes, opacities, updaters, falloffs and readouts they drive.
- **Visualizers**: Equalizer shapes, layouts and fills, and the spectrum, waveform and samples of a track they draw.
- **Narration**: Voiceover blocks timed by their takes, markers in the text, a narration script and a live take recorded while presenting.

Media

Images, SVG, video and Lottie animations.

- **Images**: Raster images fitted to a frame in every mode, cropped by pixels or fractions, panned and zoomed by animating the crop, and sampled at every quality.
- **Svg**: SVG files imported with their gradients, clip paths and filters, drawn in with write, create and fade_in, and parts addressed by id, restyled and animated.
- **Video**: MP4 clips synchronized with the timeline: fitted and cropped, trimmed, looped, sped up and slowed down, and cut into segments.
- **Lottie**: Lottie JSON clips trimmed, looped and sped up, drawn in before they play, and a dotLottie state machine driven by inputs, events and themes.

Slides

Presentations: slides, editorial components, sections and tables.

- **Slides**: A short deck of segments with slide templates, steps that stop, speaker notes, transitions and a looping stop.
- **Sections**: Agendas in every layout and marker, progress rails in rows and columns with captions, and sections of slides that move them.
- **Editorial titles**: Title cards, section headers, banners and lower thirds with their alignments, widths, variants and appearances.
- **Editorial components**: Badges and chips in every appearance and variant, cards, bullet lists and callouts that point at objects.
- **Tables**: Slide tables with their widths, row heights and alignment, and data tables built from layout rows whose cells, rows and columns animate.
- **Branding**: A deck theme derived from presentation, with a logo, footer and slide number placed on every slide but the cover.

Live audience

Polls, quizzes and live zones the audience joins from their phones.

- **Lobby**: The room the audience joins before the first question: its QR code and room code, slots that fill as people join, names and join times.
- **Polls**: A poll for the audience's phones and its live data: bars in every direction and scale, votes, shares, icons and colors, and rehearsal votes.
- **Quizzes**: Quizzes with a right answer, a countdown and a reveal, rehearsed answers, scoring, timing and questions loaded from Markdown with images.

- **Leaderboards**: Players ranked by points with their names, bars, points, correct answers, streaks and a podium.
- **Teams**: The audience split into teams whose points add up, with team scores, averages, bars, leaders and a tug of war.
- **Characters**: The characters the audience builds on their phones: every body, eyes, mouth, color and extra, expressions and named characters.
- **Live zones**: Live zones that pose the audience's characters every frame: position, rotation, scale, squash, lean, looks, expressions and names.
- **Zone behaviors**: The motion helpers of gaanim.live, from lerp and easings to springs, wobbles, impacts and ballistic arcs, and state a zone keeps per player.
- **Zone games**: Characters that move with the game: grouped by the answer each player chose, racing by score, and standing on a podium by rank.

Gallery

Short scenes in the brand's colors, from a logo built of primitives to Fourier epicycles, each one a classic of animation galleries made with the whole API.

- **Logo**: The logo built from primitives: a square that turns into a circle as it moves, and a wordmark cut from ellipses and bars.
- **Measured segment**: A segment between two points whose run, rise and length are measured live while the points move.
- **Vector arrow**: A vector on a plane whose components, angle and coordinates follow it as it sweeps around, leaving a fading trail.
- **Boolean shapes**: Two ellipses and their union, intersection, exclusion and difference, which reshape live as one ellipse turns and slides.
- **Point on shapes**: Points that travel along a circle, a square and a star with a trail of echoes, and an arm that turns about its end.
- **Moving around**: A square that slides, recolors, shrinks and turns, one call at a time and then all at once, squashing as it speeds.
- **Moving angle**: An angle that opens and closes while its arc, its value in degrees and a fading trail follow the moving ray.
- **Sliding ladder**: A ladder whose feet slide along a wall and the floor, its midpoint tracing a quarter circle and its past positions outlining an astroid.
- **Group to destination**: A cluster of points that travels so its gold leader lands on each target, the others trailing it on delayed followers.
- **Moving frame box**: The product rule written term by term, with a frame and a band that glide from one term to the next.
- **Point with trace**: A pen on the end of two turning arms that traces a rose, the second arm turning four times faster the other way.
- **Sine cosine plot**: The sine and the cosine over two turns, with a scanning line that reads both curves until it stops at two pi.
- **Gradient descent**: Gradient descent from two starts on the same curve: one ball settles in a shallow valley, the other finds the deepest one.
- **Riemann area**: Riemann sums that refine from four rectangles to sixty-four under a curve, their total closing in on the area.
- **Hyperbola rectangle**: A rectangle from the origin to a point that slides along the hyperbola y = 6/x, changing shape while its area stays six.

- **Heating curve**: The heating curve of water, traced as a thermometer fills, through melting and boiling.
- **Following camera**: A camera that zooms in on a point riding a sine wave, follows it along two turns and pulls back out.
- **Zoomed inset**: A Koch snowflake explored through a round magnifier that follows a point along its edge and zooms deeper.
- **Surface plot**: A Gaussian hill on 3D axes that the camera circles while the hill narrows into a peak and spreads back out.
- **Light source**: A gold point light that sweeps over three solids like a sun, lighting each from the side it passes.
- **Camera orbit**: Lit solids on 3D axes seen by a camera that circles them, tilts up and down, and keeps a readout fixed on the screen.
- **Warped grid**: A sentence and an equation that write themselves and transform, then a grid bent by a nonlinear function.
- **Fourier epicycles**: Five turning circles stacked on each other, the odd harmonics of a square wave, whose tip draws the wave as they spin.

Showcase

Complete pieces that combine many topics, each with its own look.

- **Dashboard**: An analytics dashboard built only from boxes, revealed piece by piece and updated live.
- **Lower third**: A broadcast lower third, a live badge and a news ticker over drifting footage.
- **Mobile app**: A phone chat app built from boxes: a clipped device, bubbles that wrap and a composer that types.
- **Quiz night**: A live trivia played in teams: a room the audience smashes into, polls and quizzes from Markdown, reveals with confetti, a tug of war, a score race and a podium.

PARTE VI

Referencia

Firmas, parámetros y contratos de la superficie pública

CAPÍTULO

34. Referencia de la API

Firmas, parámetros y comportamiento de la superficie pública de Gaanim

Cada página describe un namespace de la escena (`scene.geometry`, `scene.text`, `scene.viz`…) o una clase central. Las explicaciones están en español; los nombres de clases, métodos, parámetros y valores literales se conservan en inglés porque forman parte de la API ejecutable.

34.1.1. Cómo leer una ficha

Cada ficha documenta un símbolo:

- **Insignia**: `FÁBRICA` (crea un objeto en la escena), `MÉTODO`, `PROPIEDAD`, `CLASE` o `FUNCIÓN`.
- **Firma**: tipos, valores predeterminados y retorno, tomados del stub de tipos de Gaanim salvo que la ficha los escriba.
- **Parámetros**: los que necesitan explicación, con su tipo y su valor predeterminado.
- **Ejemplo**: código que se ejecuta con el runtime real al construir esta documentación. Si muestra una vista previa, se generó con ese mismo código.

Los ejemplos con `scene.render()` son escenas completas que puedes guardar como `main.py` y abrir con `gaanim`:

```
from gaanim import BLUE, Scene

scene = Scene(frame=(16, 9), background="#0f172a")
node = scene.geometry.circle(0.75).fill(BLUE).move_to(0, 0)
scene.play([node.animate.create().duration(1.0)])
scene.render()
```

Los demás son fragmentos: muestran solo las líneas relevantes y dan por hecho una escena como la anterior.

¿Buscas un método concreto? Pulsa `Ctrl K` o `/` y escribe su nombre o la tarea, en español o en inglés.

34.1.2. Por dónde empezar

Si es tu primera vez, lee Escena y Drawable: todo lo demás se construye sobre ellos. Para ver la API en escenas completas, consulta los ejemplos.

CAPÍTULO

35. Escena

Scene, el lienzo lógico, la línea de tiempo, los segmentos, la cámara y la salida

`Scene` es el punto de entrada de toda animación: crea los objetos, programa la línea de tiempo, organiza los segmentos y entrega el resultado al ejecutable de Gaanim. Cada script crea una escena, la llena y termina con `scene.render()`.

```
from gaanim import BLUE, Scene

scene = Scene(frame=(16, 9), background="#0f172a")
dot = scene.geometry.circle(0.6).fill(BLUE)
scene.play([dot.animate.create().duration(0.8)])
scene.wait(0.5)
scene.render()
```

Esta página documenta la escena, su lienzo (`scene.canvas`), su cámara (`scene.camera`) y las secciones. Los métodos comunes de los objetos están en Drawable, las animaciones en Animaciones y los temas en Temas y colores. Para ejecutar, validar y exportar un script, consulta la línea de comandos.

35.1.1. Constructor

35.1.2. Scene

CLASE `Scene(*, frame: tuple[float, float] = (16.0, 9.0), background: BackgroundLike | None = None, margin: float | None = None, theme: ThemeName | Theme | None = "technical", post: PostProcess | None = None)`

Los píxeles de salida no se eligen aquí sino al previsualizar o exportar, así que la composición es la misma a cualquier resolución. Un marco no finito o no positivo, WGSL inválido o un tema desconocido lanzan `ValueError`; un `theme` de otro tipo lanza `TypeError`. La antigua forma `Scene(width, height)` en píxeles ya no se acepta.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`frame`	`tuple[float, float]`	`(16.0, 9.0)`	Ancho y alto del marco lógico, centrado en el origen. Toda la geometría, los márgenes, los tamaños de texto y los trazos usan esta unidad.
`background`	`BackgroundLike | None`	`None`	Color, `Brush` o `Background`. Tiene prioridad sobre el fondo del tema.

`margin`	`float | None`	`None`	Margen uniforme en unidades lógicas; reduce el área segura.
`theme`	`ThemeName | Theme | None`	`"technical"`	Nombre o alias de un tema incluido, o un `Theme`. `None` crea la escena sin tema.
`post`	`PostProcess | None`	`None`	Postprocesado WGSL aplicado a todos los segmentos que no lo sustituyan.

Devuelve: `Scene` — Una escena vacía, lista para crear objetos.

Ejemplo:

```
from gaanim import Scene

scene = Scene(frame=(16, 9), background="#0f172a", margin=0.5, theme="presentation")
print(scene.canvas.frame_width, scene.canvas.safe_width)  # 16.0 15.0

16.0 15.0
```

Sin `theme`, la escena usa el tema `technical`: fondo gris casi negro (`#121212`), texto y ejes claros y formas rellenas con el color de acento. Un `background` explícito gana al fondo del tema, y los colores de cada objeto ganan a los del tema. `theme=None` deja un lienzo sin tema: fondo blanco (si no pasas `background`) y objetos sin estilo también blancos, así que tendrás que darles color. Consulta `Canvas.set_theme`.

35.1.3. Capacidades

Las fábricas viven en handles que pertenecen a la escena. Un objeto de una escena no puede usarse en otra: pasarlo lanza `ValueError`.

35.1.4. Scene.geometry / text / layout / media / viz / slides / mechanics / sections / assets

PROPIEDAD `scene.geometry -> Geometry · scene.text -> Typography · scene.layout -> LayoutBuilder · scene.media -> MediaLibrary · scene.viz -> Visualization · scene.slides -> SlideKit · scene.mechanics -> Mechanics · scene.sections -> SceneSections · scene.assets -> AssetManager`

Cada propiedad agrupa las fábricas de un área.

Devuelve: `handle` — Un handle de solo lectura ligado a esta escena.

Ejemplo:

Propiedad	Contenido
`scene.geometry`	Primitivas, trayectorias, flechas, geometría reactiva y 3D. Ver Geometría.
`scene.text`	Texto, ecuaciones, Typst y código. Ver Texto.

`scene.layout`	Cajas, grids, estilos y zonas. Ver Layout.
`scene.media`	Imágenes, SVG, vídeo, audio y Lottie. Ver Medios y Audio.
`scene.viz`	Ejes, funciones, gráficas, parámetros y matrices. Ver Visualización.
`scene.slides`	Tarjetas, viñetas, tablas e identidad de presentación. Ver Diapositivas.
`scene.mechanics`	Muelles, cotas, fuerzas y apoyos. Ver Mecánica.
`scene.sections`	Agenda y barra de progreso de secciones. Ver Secciones.
`scene.assets`	Carpeta de recursos, precarga y recarga. Ver Recursos.

```
shape = scene.geometry.circle(1)
title = scene.text("Resultado", role="title")
page = scene.layout.column([title, shape])
value = scene.viz.parameter(0.0)
```

35.1.5. Scene.canvas

PROPIEDAD `canvas: Canvas`

Marco, área segura, márgenes, fondo, tema, fuentes y postprocesado. Los miembros de marco y área segura están abajo; los de estilo, en Temas y colores.

Devuelve: `Canvas` — El lienzo lógico de la escena.

35.1.6. Scene.camera

PROPIEDAD `camera: Camera`

Consulta Cámara.

Devuelve: `Camera` — La cámara que se ve en la previsualización y en la exportación.

35.1.7. Scene.time

PROPIEDAD `time: TimeInput`

Es la misma fuente que `scene.viz.time`. Pásala a `computed` o a un setter absoluto; sigue los seeks exactos y la exportación.

Devuelve: `TimeInput` — Los segundos absolutos de la línea de tiempo como entrada reactiva.

Ejemplo:

```
dot.move_to(computed(lambda t: -4 + t, inputs=[scene.time]), 0)
scene.wait(3)
```

Las narraciones (`Scene.voiceover`, `Scene.narration_script` y `Scene.live_take`) están en Audio.

35.1.8. Marco lógico y área segura

El marco es el rectángulo que se ve, en unidades lógicas y centrado en el origen: con el predeterminado de 16 × 9, `x` va de `-8` a `8` e `y` de `-4.5` a `4.5`. El área segura es el marco menos los márgenes; la usan el layout (`within="safe"`) y las colocaciones en bordes y esquinas.

35.1.9. Canvas.frame_width

PROPIEDAD `frame_width: float`

Devuelve: `float` — Ancho del marco lógico; `16.0` en una escena predeterminada.

35.1.10. Canvas.frame_height

PROPIEDAD `frame_height: float`

Devuelve: `float` — Alto del marco lógico; `9.0` en una escena predeterminada.

35.1.11. Canvas.aspect_ratio

PROPIEDAD `aspect_ratio: float`

Devuelve: `float` — Ancho dividido entre alto del marco.

35.1.12. Canvas.safe_width

PROPIEDAD `safe_width: float`

Devuelve: `float` — Ancho del marco menos los márgenes izquierdo y derecho.

35.1.13. Canvas.safe_height

PROPIEDAD `safe_height: float`

Devuelve: `float` — Alto del marco menos los márgenes superior e inferior.

35.1.14. Canvas.set_margin

MÉTODO `set_margin(margin: float) -> None`

Equivale a `Scene(margin=...)` después de crear la escena.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`margin`	`float`	obligatorio	Margen igual en los cuatro lados, en unidades lógicas.

Devuelve: `None` — Reemplaza el área segura actual.

Ejemplo:

```
scene.canvas.set_margin(0.4)
print(scene.canvas.safe_width, scene.canvas.safe_height)  # 15.2 8.2
```

```
15.2 8.2
```

35.1.15. Canvas.set_safe_area

MÉTODO `set_safe_area(*, top: float = 0.0, right: float = 0.0, bottom: float = 0.0, left: float = 0.0) -> None`

Útil cuando una marca o una plataforma reserva franjas propias, como la interfaz de un vídeo vertical.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`top`	`float`	`0.0`	Margen superior en unidades lógicas.
`right`	`float`	`0.0`	Margen derecho.
`bottom`	`float`	`0.0`	Margen inferior.
`left`	`float`	`0.0`	Margen izquierdo.

Devuelve: `None` — Reemplaza el área segura actual.

Ejemplo:

```
scene.canvas.set_safe_area(top=0.5, bottom=1.0)
print(scene.canvas.safe_height)  # 7.5

7.5
```

35.1.16. Canvas.set_preset

MÉTODO `set_preset(name: Literal["widescreen", "vertical", "square"]) -> None`

`"widescreen"` da un marco 16 × 9 con área segura 15 × 8, `"vertical"` un marco 9 × 16 con área segura 8 × 13.5 y `"square"` un marco 10 × 10 con área segura 9 × 9.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`name`	`"widescreen" | "vertical" | "square"`	obligatorio	Composición estándar.

Devuelve: `None` — Reemplaza el marco lógico y el área segura.

Ejemplo:

```
scene.canvas.set_preset("vertical")
page = scene.layout.column([scene.text("Vídeo vertical", role="title")], within="safe")
```

35.1.17. Línea de tiempo

Cada llamada a `play` o `wait` avanza el cursor de autoría. Las animaciones de una misma lista empiezan juntas; llamadas sucesivas van una detrás de otra. `scene.launch(...)` (o `play(..., advance=False)`) empieza en el cursor sin moverlo, para una animación larga que sigue mientras programas lo demás.

35.1.18. Scene.play

MÉTODO `play(items: Playable | Sequence[Playable], *, duration: Optional[float] = None, easing: Optional[Easing] = None, advance: bool = True) -> None`

Programa el bloque de forma atómica: un `Anim` ya usado, de otra escena, repetido o que escribe un canal ocupado en el mismo tramo lanza `ValueError` sin aplicar ningún cambio. `duration` y `easing` son valores por defecto; los que fija cada `Anim` o cada grupo tienen prioridad.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`items`	`Playable | Sequence[Playable]`	obligatorio	Un `Anim`, audio, vídeo, Lottie o `Composition`, o una lista que se reproduce en paralelo.
`duration`	`float | None`	`None`	Duración para las animaciones que no fijan la suya.
`easing`	`Easing | None`	`None`	Easing para las animaciones que no fijan el suyo.
`advance`	`bool`	`True`	Con `False` el cursor no se mueve: el bloque empieza y sigue sonando bajo lo que programes después. Es lo mismo que `scene.launch`.

Devuelve: `None` — Avanza el cursor hasta el final del bloque, salvo con `advance=False`.

Ejemplo:

```
from gaanim import BLUE, GOLD, Easing, Scene

scene = Scene(frame=(16, 9), theme="technical")
circle = scene.geometry.circle(0.8).fill(BLUE).move_to(-1.6, 0)
rect = scene.geometry.rect(1.8, 1.0).fill(GOLD).move_to(1.6, 0)
scene.play([circle.animate.create(), rect.animate.grow_from_center()], duration=0.8)
scene.play(circle.animate.shift_by(1.2, 0), easing=Easing.SNAPPY)
scene.render()
```

35.1.19. Scene.launch

MÉTODO `launch(items: Playable | Sequence[Playable], *, duration: Optional[float] = None, easing: Optional[Easing] = None) -> None`

Empieza el bloque en el cursor y deja el cursor donde estaba: lo que programes después (esperas, otros `play`, cortes, segmentos) ocurre mientras el bloque sigue. Sirve para un giro que dura varios cortes o una música de fondo. Animar el mismo canal del mismo objeto antes de que termine lanza `ValueError`. El vídeo acaba en el cursor, así que lo que siga corriendo ahí se corta; `gaanim check` lo avisa. Los objetos de un segmento salen de pantalla al terminar este, salvo que `scene.persist` los mantenga.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`items`	`Playable | Sequence[Playable]`	obligatorio	Lo mismo que acepta `play`.
`duration / easing`	`float | Easing | None`	`None`	Valores por defecto, como en `play`.

Devuelve: `None` — No mueve el cursor.

Ejemplo:

```
import math
rueda = scene.geometry.regular_polygon(6, 1.5).fill("#38bdf8")
puntos = [scene.geometry.dot(0.2).move_to(x, -3) for x in (-4, -2, 0, 2, 4)]
scene.launch(rueda.animate.rotate_by(math.tau).duration(5.0))
for punto in puntos:
    scene.play(punto.animate.fade_in().duration(1.0))
```

35.1.20. Scene.tempo

MÉTODO `tempo(bpm: float, offset: float = 0.0, *, beats_per_bar: int = 4) -> None`

Rejilla de tempo para cortar a tiempo con la música: `scene.beats(n)` convierte pulsos en segundos y `scene.wait_until(beat=n)` o `scene.wait_until(bar=n)` llevan el cursor a un pulso o al inicio de un compás. Si el cursor ya pasó ese pulso, `wait_until` lanza `ValueError` y dice cuánto se alargó el plano anterior, en vez de desfasarse en silencio. La barra de reproducción del editor dibuja una línea en cada compás. Valores no válidos lanzan `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`bpm`	`float`	obligatorio	Pulsos por minuto; positivo.
`offset`	`float`	`0.0`	Segundo de la línea de tiempo donde cae el pulso 0, el primer tiempo de la música.
`beats_per_bar`	`int`	`4`	Pulsos por compás.

Devuelve: `None` — Fija el tempo de la escena.

Ejemplo:

```
scene.tempo(128, offset=0.0)
logo = scene.geometry.star(5, 1.2, 0.5).fill("#fbbf24")
scene.play(logo.animate.grow_from_center().duration(scene.beats(2)))
scene.wait_until(bar=1)
scene.play(logo.animate.rotate_by(1.0).duration(scene.beats(4)))
scene.wait_until(bar=2)
```

35.1.21. Scene.wait

MÉTODO `wait(seconds: float) -> None`

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`seconds`	`float`	obligatorio	Tiempo que se mantiene la imagen, en segundos.

Devuelve: `None` — Avanza el cursor.

Ejemplo:

```
scene.wait(1.0)
```

35.1.22. Scene.fade_out_all

MÉTODO `fade_out_all(seconds: float) -> None`

Desvanece a la vez, con easing suave, todos los objetos del segmento activo. Es la forma rápida de cerrar una escena.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`seconds`	`float`	obligatorio	Duración del fundido.

Devuelve: `None` — Avanza el cursor `seconds`.

Ejemplo:

```
from gaanim import Scene

scene = Scene(frame=(16, 9), theme="technical")
title = scene.text("Fin", role="title")
scene.play(title.animate.write())
scene.fade_out_all(0.6)
scene.render()
```

35.1.23. Scene.cursor

PROPIEDAD `cursor: float`

Cuenta a través de todos los segmentos y es de solo lectura.

Devuelve: `float` — Posición del cursor de autoría, en segundos absolutos de la línea de tiempo.

Ejemplo:

```
scene.play([title.animate.write().duration(0.8)])
reveal_time = scene.cursor  # 0.8
```

35.1.24. Scene.marker

MÉTODO `marker(name: str) -> None`

Da nombre al instante actual de la línea de tiempo global. Es solo un metadato: no pausa la reproducción. El editor lo dibuja como una banderita violeta sobre la barra de tiempo (al pasar el cursor muestra el nombre y un clic salta a él), y `gaanim export --from` / `--to` lo aceptan en lugar de segundos. Un nombre vacío, repetido o que se lee como número lanza `ValueError`. Para instantes dentro de una `Composition`, usa `label`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`name`	`str`	obligatorio	Nombre único del instante; se recortan los espacios.

Devuelve: `None` — No añade duración ni mueve el cursor.

Ejemplo:

```
scene.play(intro)
scene.marker("climax")
scene.play(outro)
scene.marker("fin")
```

```
gaanim export escena.py --output climax.mp4 --from climax --to fin
```

35.1.25. Scene.markers

PROPIEDAD `markers: list[SceneMarker]`

Cada `SceneMarker` tiene `name`, `time` (segundos absolutos) y `segment` (el segmento activo al crearlo).

Devuelve: `list[SceneMarker]` — Los marcadores creados hasta ahora, en orden temporal.

Ejemplo:

```python
scene.wait(1.5)
scene.marker("climax")
print([(m.name, m.time) for m in scene.markers])  # [('climax', 1.5)]

[('climax', 1.5)]
```

35.1.26. Segmentos

Un segmento es la unidad estructural de la escena: agrupa objetos, define la transición de entrada, las notas para Presenter View, la plantilla y un fondo propio. Los límites entre segmentos son continuos; `stop()` marca dónde la reproducción interactiva espera al presentador.

35.1.27. Scene.segment

MÉTODO `segment(name: str, transition: Transition | None = None, *, notes: str | None = None, template: Callable[..., Layout] | None = None, background: BackgroundLike | None = None, post: PostProcess | Literal[False] | None = None, target: float | None = None) -> Segment`

Crea el segmento y lo activa. La primera llamada sustituye al segmento implícito inicial, que aún no tiene contenido. Un nombre vacío o repetido, una transición en el primer segmento, un `target` que no sea un número positivo de segundos o un guion ilegible lanzan `ValueError`; un `post` de otro tipo lanza `TypeError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`name`	`str`	obligatorio	Nombre no vacío, único en la escena sin distinguir mayúsculas.
`transition`	`Transition | None`	`None`	Transición de entrada desde el segmento anterior. Ver Transiciones.
`notes`	`str | None`	`None`	Notas del orador que muestra Presenter View; una línea `[stop]` marca dónde avanzas. La sección `## nombre` del guion, si existe, las sustituye (ver El guion de la charla).
`template`	`Callable[..., Layout] | None`	`None`	Plantilla de Python que se rellena después con `Segment.bind`.
`background`	`BackgroundLike | None`	`None`	Fondo solo mientras el segmento está activo; `None` usa el de la escena.

`post`	`PostProcess | False | None`	`None`	Postprocesado del segmento: otro `PostProcess`, `False` para ninguno o `None` para heredar `scene.canvas.post`.
`target`	`float | None`	`None`	Segundos que prevés dedicarle al presentar. Presenter View compara con él el tiempo que pasas en la diapositiva (ver El tiempo de la charla). La de la charla entera se fija con `scene.slides.target`.

Devuelve: `Segment` — Handle que aceptan `link` y `bind`.

Ejemplo:

```
intro = scene.segment("Introducción", notes="Presenta el objetivo.", template=title_slide)
intro.bind(title=scene.text("Una idea clara", role="title"))
scene.wait(0.5)
details = scene.segment(
    "Detalles",
    Transition.cross_fade(0.4),
    background=Brush.linear(["#172554", "#0f172a"], start=(-8, 0), end=(8, 0)),
)
scene.wait(0.5)
```

35.1.28. Segment.bind

MÉTODO `bind(**slots: Any) -> Box`

Rellena la plantilla del segmento. Un hueco obligatorio que falta o uno que no existe lanzan `TypeError`; un segmento sin plantilla lanza `ValueError`. Las plantillas incluidas (`title_slide`, `lecture`, `comparison`…) se importan desde `gaanim`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`slots`	`**Any`	`{}`	Un argumento con nombre por hueco de la plantilla.

Devuelve: `Layout` — El layout raíz del segmento.

Ejemplo:

```
results = scene.segment("Resultados", template=comparison, notes="Compara los dos modelos.")
results.bind(title=scene.text("Resultados", role="title"), left=baseline, right=proposed)
scene.stop("comparación")
```

35.1.29. Segment.motion_blur

MÉTODO `segment.motion_blur(shutter_angle=180.0, samples=8, *, phase=None) -> Segment`

Desenfoque de movimiento de este segmento en las exportaciones y las capturas, en lugar del de la escena. Los segmentos sin uno propio usan el de la escena, así que cada segmento puede comparar un ángulo de obturador o una `phase`. Un cuadro justo en un corte pertenece al segmento que entra. Un valor fuera de rango lanza `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`shutter_angle, samples, phase`	`float | None, int, float | None`	`180.0, 8, None`	Como en `scene.canvas.mo` `None` deja nítido el segmento.

Devuelve: `Segment` — El mismo segmento.

Ejemplo:

```
scene.canvas.motion_blur(180)
scene.segment("normal")
scene.wait(1)
scene.segment("amplio").motion_blur(360, samples=16)
scene.wait(1)
scene.segment("nítido").motion_blur(None)
scene.wait(1)
```

35.1.30. Scene.link

MÉTODO `link(from_: Segment, to: Segment, transition: Transition) -> None`

Declara la transición entre dos segmentos ya creados. Hace falta para `Transition.morph`, cuyos pares incluyen objetos del segmento de destino. Un segmento de otra escena, o un `to` que no va después de `from_`, lanzan un error.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`from_`	`Segment`	obligatorio	Segmento de origen.
`to`	`Segment`	obligatorio	Segmento de destino, posterior a `from_`.
`transition`	`Transition`	obligatorio	Transición que sustituye a la de entrada de `to`.

Devuelve: `None` — No mueve el cursor.

Ejemplo:

```
overview = scene.segment("Resumen")
card = scene.geometry.rect(3, 2).fill(BLUE).move_to(-4, 0)
scene.wait(0.5)
detail = scene.segment("Detalle")
panel = scene.geometry.rect(12, 6).fill(BLUE)
scene.wait(0.5)
scene.link(overview, detail, Transition.morph(0.8, pairs=[(card, panel)]))
```

35.1.31. Scene.stop

MÉTODO `stop(name: Optional[str] = None, *, loop: Optional[Playable | Sequence[Playable]] = None, until: Optional[Condition] = None) -> None`

Pausa la reproducción interactiva cuando el cursor llega a este instante. En el límite de un segmento, el segmento saliente sigue visible hasta que se avanza, así que no hace falta un `wait()` final. Si la diapositiva todavía está entrando con su transición, la pausa espera a que termine y avanza el cursor hasta ahí, para que la presentación nunca descanse en una diapositiva a medio entrar. La exportación, las capturas y los seeks ignoran las pausas. Después de un `live_take()` grabado, la pausa espera tanto como la pausa real del orador (ver Audio). Con `loop`, la animación se coloca justo después de la pausa y, en lugar de congelar la imagen, se repite mientras el orador habla: el siguiente paso sale del bucle y sigue desde su final, y retroceder lo salta. Al exportar se reproduce una vez. Para que la repetición no salte, haz que el bucle termine como empieza. Un nombre vacío, una segunda pausa en el mismo instante del segmento o un bucle sin duración lanzan `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`name`	`str | None`	`None`	Etiqueta de la pausa en Presenter View.
`loop`	`Anim | list | None`	`None`	Animación ambiental, con la misma forma que `scene.play`, que se repite mientras la presentación descansa en la pausa.
`until`	`Condition | None`	`None`	Condición del público con la que la pausa avanza sola al presentar. Ver Avanzar por sí sola.

Devuelve: `None` — Sin `loop`, no añade duración ni cambia la imagen, salvo lo que falte de la transición de entrada; con `loop`, avanza además el cursor lo que dura el bucle.

Ejemplo:

```
scene.play([result.animate.write().duration(0.6)])
scene.stop("resultado")
scene.wait(0.5)
# Las flechas siguen moviéndose mientras se explica la pausa.
```

```
scene.stop("dos-placas", loop=sequence(
    arrow.animate.shift_by(0.5, 0).duration(0.75),
    arrow.animate.shift_by(-0.5, 0).duration(0.75),
))
```

35.1.32. Scene.rehearsal

MÉTODO `rehearsal(players: int | Sequence[str] = 12, *, seed: int = 0, arrive: Optional[float] = None, skill: float | Sequence[float] = 0.6, speed: float = 0.5) -> None`

Describe el público inventado que juega la escena fuera de una presentación en vivo. En la previsualización, la exportación y las capturas sus jugadores entran a la sala, responden cada encuesta y cuestionario, el reloj baja y la clasificación suma sus puntos; las zonas vivas también lo usan. Las respuestas de una pregunta llegan entre que se abre y la pausa donde la presentación las esperaría. Los mismos argumentos dan siempre el mismo ensayo. Sin llamarlo ensayan 12 jugadores. Sin jugadores o con más de 200, un apodo vacío o repetido, `skill` o `speed` fuera de 0–1 o un `arrive` negativo lanzan `ValueError`. Ver Ensayo.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`players`	`int | Sequence[str]`	`12`	Cuántos jugadores (Ana, Beto, Caro…) o sus apodos, en orden de llegada; de 1 a 200.
`seed`	`int`	`0`	Elige otro grupo: otros tiempos, respuestas y personajes.
`arrive`	`float | None`	`None`	Segundos en que entran todos; por defecto, entre `scene.audience()` y la primera pausa de la sala.
`skill`	`float | Sequence[float]`	`0.6`	Qué parte de las preguntas aciertan en promedio, de 0 a 1; con equipos, puede ser un valor por equipo.
`speed`	`float`	`0.5`	Qué tan pronto responden, de 0 (al final) a 1 (al instante).

Devuelve: `None` —

Ejemplo:

```
scene.rehearsal(24, seed=3, skill=0.7)
audience = scene.audience()
```

35.1.33. Scene.question

MÉTODO `question(question: Question) -> Poll`

Abre una pregunta escrita fuera de Python, con su tiempo, puntos, imagen, modo y ensayo. Ver Preguntas en un archivo.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`question`	`Question`	obligatorio	Una pregunta de `gaanim.load_questions`, que lee un Markdown o un CSV.

Devuelve: `Poll` — Un cuestionario si la pregunta tiene respuestas correctas; si no, una encuesta.

Ejemplo:

```
>>>from gaanim import *
>>>scene = Scene(frame=(16, 9))
for q in load_questions("preguntas.md"):
    poll = scene.question(q)
```

35.1.34. Scene.roster

MÉTODO `roster(ask: str, *, required: bool = True) -> None`

Pide a cada jugador un dato más al unirse, como su código de alumno o su nombre completo, para que el profesor sepa quién es quién. El dato nunca llega a la pantalla: solo aparece en los resultados que guarda la presentación y en `gaanim relay results`. Quien ya se había unido sigue jugando y lo da la próxima vez que se una. Un texto vacío o de más de 40 caracteres, o un segundo `ask` distinto en la misma escena, lanzan `ValueError`. Ver Guardar los resultados.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`ask`	`str`	obligatorio	Lo que el teléfono pide además del apodo, como `"Código de alumno"`; de 1 a 40 caracteres.
`required`	`bool`	`True`	Si nadie puede unirse sin responderlo.

Devuelve: `None` —

Ejemplo:

```
scene.roster("Código de alumno")
audience = scene.audience()
```

35.1.35. Scene.teams

MÉTODO `teams(names: Sequence[str], *, choose: bool = False, colors: Optional[Sequence[str]] = None) -> Teams`

Hace que el juego se juegue por equipos. Los puntos de cada jugador suman para su equipo. `Teams` da `names`, `colors`, `score(i)`, `players(i)` y `average(i)` (`Parameter`), `leader()` (el índice del equipo que va ganando) y `bar(i, ...)`. En las zonas vivas cada jugador sabe `p.team`, `p.team_index`, `p.team_count`, `p.team_score` y `p.team_rank`. El ensayo reparte a sus jugadores igual. Menos de 2 o más de 6 equipos, un nombre vacío, largo o repetido, colores que no sean un `#rrggbb` por equipo, u otros equipos distintos en la misma escena lanzan `ValueError`. Ver Equipos.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`names`	`Sequence[str]`	obligatorio	Los equipos, de 2 a 6, con nombres de hasta 20 caracteres.
`choose`	`bool`	`False`	Si el teléfono pregunta a qué equipo unirse; si no, el relay pone a cada jugador en el que tiene menos.
`colors`	`Sequence[str] | None`	`None`	Un `#rrggbb` por equipo, como lo muestran los teléfonos; por defecto rosa, azul, naranja, verde, violeta y turquesa.

Devuelve: `Teams` — Los datos de los equipos; no dibuja nada.

Ejemplo:

```
teams = scene.teams(["Rojo", "Azul"])
for team, x in [(0, -4), (1, 4)]:
    scene.viz.readout(teams.score(team), format=".0f").move_to(x, 0)
```

35.1.36. Scene.poll

MÉTODO `poll(question: str, options: Sequence[str], *, multiple: bool = False, image: Optional[str | os.PathLike[str]] = None, rehearse: Optional[Sequence[float]] = None) -> Poll`

Abre una encuesta en el cursor. Recibe votos mientras una presentación está entre este punto y `poll.close()`, o el final del segmento. La escena decide cómo mostrarla con los datos de `Poll`: `qr(tamaño)` y `bar(respuesta, ...)` devuelven drawables; `votes(i)`, `share(i)` y `total()` devuelven `Parameter` que siguen los votos en vivo; `code` y `url` son el código de la sesión y la dirección de votación. Al presentar un paquete `.gaanim` solo las barras siguen los votos en vivo; lo que pase por un `computed` muestra lo que se grabó. Fuera de una presentación vota el ensayo (`scene.rehearsal`). Ver Encuestas a la audiencia. Una pregunta o respuesta vacía, una respuesta repetida, menos de 2 o más de 6 respuestas, o pesos de `rehearse` que no sean uno por respuesta, negativos o todos cero lanzan `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción

`question`	`str`	obligatorio	Pregunta que responde la audiencia.
`options`	`Sequence[str]`	obligatorio	Entre 2 y 6 respuestas distintas.
`multiple`	`bool`	`False`	Deja elegir varias respuestas; `total()` y `share(i)` cuentan a los teléfonos que respondieron.
`image`	`str | PathLike | None`	`None`	Una imagen, relativa al script, que los teléfonos muestran sobre la pregunta.
`rehearse`	`Sequence[float] | None`	`None`	Un peso por respuesta para el voto del ensayo (`[1, 3]` hace la segunda tres veces más popular); sin él vota al azar.

Devuelve: `Poll` — Los datos de la encuesta; no dibuja nada.

Ejemplo:

```
poll = scene.poll("¿Qué crece más rápido?", ["x²", "2ˣ"], rehearse=[1, 2])
card = scene.geometry.rounded_rect(3.4, 3.4, 0.2).fill(WHITE).no_stroke().move_to(-4, 0)
qr = poll.qr(3.0).move_to(-4, 0)
bar = poll.bar(1, length=6).fill(GOLD).no_stroke().move_to(2, 0)
votes = scene.viz.readout(poll.votes(1), format=".0f").move_to(5.6, 0)
scene.stop()
```

35.1.37. Scene.quiz

MÉTODO `quiz(question: str, options: Sequence[str], correct: int, *, time: int = 20, points: int = 1000, image: Optional[str | os.PathLike[str]] = None, rehearse: Optional[float | Sequence[float]] = None) -> Poll`

Abre un cuestionario al estilo Kahoot: una encuesta con respuesta correcta. Los teléfonos se unen al juego con un apodo, responden una sola vez antes de que se acabe el tiempo y una respuesta correcta gana `points × (1 − tiempo / time / 2)`. `quiz.reveal()` marca dónde la presentación revela la respuesta en los teléfonos, `quiz.remaining()` es la cuenta atrás en segundos y `scene.leaderboard` da la clasificación. Una parte de `rehearse` fuera de 0–1 lanza `ValueError`. Ver Modo competencia.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`question`	`str`	obligatorio	Pregunta del juego.
`options`	`Sequence[str]`	obligatorio	Entre 2 y 6 respuestas distintas.
`correct`	`int | Sequence[int]`	obligatorio	Índice de la respuesta correcta (0 para la primera), o una lista

			para selección múltiple: acierta quien marca todas y ninguna otra.
`time`	`int`	`20`	Segundos para responder, de 5 a 300, medidos por el reloj del relay.
`points`	`int`	`1000`	Puntos máximos de una respuesta correcta, de 100 a 10000.
`image`	`str | PathLike | None`	`None`	Una imagen que los teléfonos muestran sobre la pregunta.
`rehearse`	`float | Sequence[float] | None`	`None`	La parte del ensayo que acierta, de 0 a 1 (`0.3` para una pregunta difícil), o un peso por respuesta; sin él responden según su `skill`.

Devuelve: `Poll` — Los datos del cuestionario, como `scene.poll`.

Ejemplo:

```
quiz = scene.quiz("¿Derivada de x²?", ["x", "2x", "x²/2"], correct=1, time=20, rehearse=0.6)
clock = scene.viz.readout(quiz.remaining(), format=".0f").move_to(-5, 3)
scene.wait(1)
scene.stop()
quiz.reveal()
scene.wait(1)
scene.stop()
```

35.1.38. Scene.leaderboard

MÉTODO `leaderboard() -> Leaderboard`

La clasificación del juego: los jugadores de los cuestionarios de la escena, de mejor a peor. Cada método recibe un puesto (0 para el primero): `name(i, size=, weight=, font=, align=)` devuelve el apodo como texto en vivo, `points(i)` y `players()` devuelven `Parameter` y `bar(i, ...)` una barra relativa al primero. Con ellos diseñas una lista, un podio o lo que quieras. Todo sigue a los resultados en vivo también al presentar un `.gaanim`, que guarda los glifos de los apodos; fuera de una presentación ordena a los jugadores del ensayo por los puntos de sus respuestas.

Devuelve: `Leaderboard` — Los datos de la clasificación; no dibuja nada.

Ejemplo:

```
board = scene.leaderboard()
for rank in range(2):
    board.name(rank, size=0.5).move_to(-3, 1 - rank)
    scene.viz.readout(board.points(rank), format=".0f").move_to(3, 1 - rank)
```

35.1.39. Scene.live_zone

MÉTODO `live_zone(audience: Audience, behavior: Callable[[Player], Pose], *, bounds: tuple[float, float, float, float] = (-8.0, -4.5, 8.0, 4.5), size: float = 1.2, squash: float = 0.04, max_stretch: float = 1.4, lean: float = 0.05, max_lean: float = 0.35, follow: float = 1.0, look: float = 0.4, names: bool = False, name_size: float = 0.24, name_color: str = "#ffffff", name_gap: float = 0.08, name_weight: Optional[int] = 700, state: Optional[Mapping[str, float]] = None, update: Optional[Callable[[Player], State]] = None) -> LiveZone`

Abre una zona viva en el cursor, hasta `zone.close()` o el final del segmento: al presentar, cada jugador llega como su personaje y el comportamiento lo coloca en cada cuadro; fuera de una presentación juega el ensayo. El comportamiento y la actualización se compilan, así que un `.gaanim` los ejecuta sin Python. Algo que el compilador no admite lanza `gaanim.live.BehaviorError` señalando la línea; unos límites vacíos, un tamaño no positivo, una expresión desconocida, `state` sin `update` (o al revés) o más de 8 números lanzan `ValueError`. Ver Zonas vivas.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`audience`	`Audience`	obligatorio	El público de `scene.audience()`.
`behavior`	`Callable[[Player], Pose]`	obligatorio	Una función de Python que recibe al jugador `p` y devuelve `pose(x, y, ...)`; se compila al llamar.
`bounds`	`tuple[float, float, float, float]`	`(-8, -4.5, 8, 4.5)`	`(x0, y0, x1, y1)`: los personajes se recortan a esta caja.
`size`	`float`	`1.2`	Alto de un personaje a escala 1.
`squash`	`float`	`0.04`	Cuánto se estira en la dirección en que se mueve, por unidad de

			velocidad (hasta `max_stretch`, 1,4).
`lean`	`float`	`0.05`	Cuánto se inclina al moverse de lado (hasta `max_lean`, 0,35 radianes).
`follow`	`float`	`1.0`	Cuánto se balancean orejas, sombreros y brotes detrás del movimiento.
`look`	`float`	`0.4`	Cuánto miran los ojos hacia donde va.
`names`	`bool`	`False`	Dibuja el apodo bajo los pies (`name_size`, `name_color`, `name_gap`, `name_weight`).
`state`	`Mapping[str, float] | None`	`None`	Números que la zona guarda por jugador (hasta 8) y dónde empieza cada uno.
`update`	`Callable[[Player], State] | None`	`None`	Calcula cada `STEP` (1/60 s) los siguientes números de `state` y devuelve `state(...)`.

Devuelve: `LiveZone` — La zona; `close()` la cierra en el cursor.

Ejemplo:

```
from gaanim.live import pose
audience = scene.audience()

def en_fila(p):
    return pose(-6 + p.index * 1.2, -3, express="happy")

zone = scene.live_zone(audience, en_fila, names=True)
```

35.1.40. Scene.character

MÉTODO `character(avatar: Optional[Sequence[int]] = None, *, name: str = "", size: float = 2.0) -> Character`

Un personaje como los que crea el público en su teléfono, y que se mueve igual que allí. `character.express("happy")` hace una expresión desde el cursor; con `loop=True` la repite hasta la siguiente, y `express()` vuelve a su cara. Expresiones: `happy`, `sad`, `hurt`, `winner` y `surprised`. Como todo depende del tiempo de la línea de tiempo, saltar y exportar dan el mismo resultado.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`avatar`	`Sequence[int] | None`	`None`	Las partes: cuerpo, color, ojos, boca y extra. Sin ellas se deducen de `name`.
`name`	`str`	`""`	El apodo, que también marca el ritmo con que parpadea.
`size`	`float`	`2.0`	Alto en unidades, con sombreros y orejas.

Devuelve: `Character` — Un dibujable que respira, parpadea y hace expresiones.

Ejemplo:

```
hero = scene.character([1, 0, 4, 2, 1], name="Ana").move_to(-3, 0)
scene.wait(1)
hero.express("winner", loop=True)
scene.wait(2)
```

35.1.41. Scene.audience

MÉTODO `audience() -> Audience`

El público del juego: los jugadores en el orden en que entraron, cada uno en un puesto (0 para el primero). `name(i, size=, weight=, font=, align=)` devuelve el apodo como texto en vivo; `count()`, `joined(i)` (1 si el puesto está ocupado, 0 si no) y `age(i)` (segundos desde que entró, hasta 60) devuelven `Parameter`; `qr(size)`, `url` y `code` sirven para invitar. Con ellos diseñas una sala de espera, una arena o lo que quieras. Una escena que lo usa pide el apodo en cuanto el teléfono abre la página. Fuera de una presentación entran los jugadores del ensayo, uno tras otro. Ver Sala de espera.

Devuelve: `Audience` — Los datos del público; no dibuja nada.

Ejemplo:

```
audience = scene.audience()
for slot in range(6):
    audience.name(slot, size=0.4).move_to((slot - 2.5) * 2, 0)
```

35.1.42. Scene.stops

PROPIEDAD `stops: list[SceneStop]`

Cada `SceneStop` tiene `name` (o `None`), `time` en segundos absolutos y `segment`. Léela al final del script, antes de `render()`, para pedir una captura por pausa; `gaanim --diff --capture-stops` hace lo mismo sin cambiar el script.

Devuelve: `list[SceneStop]` — Las pausas creadas hasta ahora, en orden temporal.

Ejemplo:

```
if snapshots := os.environ.get("GAANIM_SNAPSHOTS"):
    scene.snapshots(snapshots, [stop.time for stop in scene.stops])
```

Al empezar un segmento, los objetos de los anteriores dejan de verse. Estos tres métodos cambian a qué segmento pertenece un objeto, en el cursor actual y sin alterar su estado visual. Un objeto de otra escena lanza `ValueError`.

35.1.43. Scene.reuse

MÉTODO `reuse(object: Drawable, *others: Drawable) -> None`

Adopta objetos en el segmento activo. Un objeto visible en el segmento anterior queda quieto durante la transición automática y luego pasa a ser contenido del segmento activo. Llamarlo después de `play()` o `wait()` actúa en ese instante.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`object`	`Drawable`	obligatorio	Primer objeto.
`others`	`Drawable`	`()`	Más objetos de la misma escena.

Devuelve: `None`

35.1.44. Scene.z_layers

MÉTODO `z_layers(*names: str, default: Optional[str] = None) -> None`

Nombra las capas de dibujo de la escena. Todo objeto de una capa (`Drawable.z_layer`) se dibuja encima de todos los de las capas de detrás, sea cual sea su `z_index`, que solo ordena dentro de cada capa. Sirve para que lo que va encima (ventanas, lupas, etiquetas, un cursor) no dependa de un `z_index` mayor que cualquiera del resto del diagrama. Una escena nombra sus capas una vez; repetir las mismas no hace nada. Sin nombres, con un nombre vacío o repetido, con un `default` que no está entre ellos o con capas distintas a las ya nombradas lanza `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`names`	`str`	obligatorio	Nombres de las capas, de atrás hacia adelante.

`default`	`str | None`	`None`	Capa de los objetos sin `z_layer`; sin él, la primera.

Devuelve: `None`

Ejemplo:

```
scene.z_layers("fondo", "contenido", "overlay", default="contenido")
rejilla = scene.geometry.rect(14, 8).fill(GRAY).z_layer("fondo")
cursor = scene.geometry.dot(0.1).fill(GOLD).z_layer("overlay")
```

35.1.45. Scene.persist

MÉTODO `persist(object: Drawable, *others: Drawable) -> None`

Mantiene objetos visibles y animables en los segmentos siguientes, desde el cursor actual. Las transiciones automáticas (`cross_fade`, `slide`…) no los afectan. Con el mismo `z_index`, un objeto persistente se dibuja encima del contenido de los segmentos, durante las transiciones y fuera de ellas; con un `z_index` menor queda debajo (una rejilla persistente con `z_index(-1)`), salvo en una transición por shader, que lo compone encima de la mezcla. Un objeto invisible sigue invisible hasta que una animación de entrada cambia su opacidad. Un objeto que no estaba en pantalla antes del segmento actual y se persiste al empezarlo entra con ese segmento: su transición de entrada lo revela como al resto del segmento, y pasa a ser global cuando termina. Para dejarlo fuera de esa transición, créalo y persístelo antes de `scene.segment(...)`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`object`	`Drawable`	obligatorio	Primer objeto.
`others`	`Drawable`	`()`	Más objetos de la misma escena.

Devuelve: `None`

35.1.46. Scene.release

MÉTODO `release(object: Drawable, *others: Drawable) -> None`

Termina la persistencia y devuelve los objetos al segmento activo. Al inicio de un segmento, el objeto queda quieto durante la transición de entrada y después pasa a ser local. Nunca oculta ni elimina el objeto: la siguiente transición lo trata como contenido saliente normal.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`object`	`Drawable`	obligatorio	Primer objeto.
`others`	`Drawable`	`()`	Más objetos de la misma escena.

Devuelve: `None`

Ejemplo:

```
from gaanim import BLUE, GOLD, Scene, Transition

scene = Scene(frame=(16, 9), background="#0f172a")
title = scene.text("Contexto compartido", role="title").fill(GOLD).move_to(0, 0.875)
scene.play([title.animate.write().duration(0.5)])

scene.segment("contenido", Transition.cross_fade(0.35))
scene.reuse(title)
dot = scene.geometry.dot(0.225).fill(BLUE).move_to(0, -0.375)
scene.play([dot.animate.grow_from_center().duration(0.4)])
scene.persist(title)

scene.segment("cierre", Transition.slide(0.35, "left"))
scene.release(title)
scene.wait(0.5)
scene.render()
```

La identidad de una presentación (logo, pie, número de diapositiva) se configura una vez con `scene.slides.brand(...)`; consulta Presentaciones.

35.1.47. Secciones

`Section` agrupa pasos de contenido que se construyen con funciones de Python. Cada paso abre su propio segmento; la función recibe la escena y crea contenido y pausas sin volver a llamar a `segment`. Estas clases se importan desde `gaanim`.

35.1.48. Section

CLASE `Section(key: str, steps: Sequence[SectionStep], *, title: str | None = None)`

Una clave, un título o una lista de pasos vacíos lanzan `ValueError`; un paso de otro tipo, `TypeError`. `section.build(scene, *, on_enter=None)` abre cada segmento, llama a `on_enter(scene, progress)` y ejecuta la función del paso; devuelve los `Segment` en orden y no inserta pausas. Reutiliza la misma instancia para repetir la sección: el progreso vuelve a empezar y los nombres de segmento incluyen clave, visita, número y nombre del paso. Las excepciones se propagan sin deshacer lo ya creado.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`key`	`str`	obligatorio	Identidad estable de la sección; única entre las secciones de una escena.
`steps`	`Sequence[SectionStep]`	obligatorio	Pasos en orden; se copian al crear la sección.
`title`	`str | None`	`None`	Nombre que muestran la agenda y la barra de progreso; sin él se usa `key`.

Devuelve: `Section` — Tiene `key`, `title` y `steps`.

Ejemplo:

```
from gaanim import Scene, Section, SectionStep

scene = Scene(frame=(16, 9), theme="technical")

def contexto(scene):
    title = scene.text("Contexto", size=0.5)
    scene.play(title.animate.write())
    scene.stop()

def propuesta(scene):
    title = scene.text("Propuesta", size=0.5)
    scene.play(title.animate.write())
    scene.stop()

section = Section("introduccion", [
    SectionStep(name="Contexto", build=contexto, notes="Presenta el problema."),
    SectionStep(name="Propuesta", build=propuesta),
])

def al_entrar(scene, progress):
    print(progress.key, progress.index, progress.total, progress.fraction)

segments = section.build(scene, on_enter=al_entrar)
scene.render()

introduccion 1 2 0.5
introduccion 2 2 1.0
```

35.1.49. SectionStep

CLASE `SectionStep(*, name: str, build: Callable[[Scene], None], transition: Transition | None = None, notes: str | None = None, template: Callable[..., Layout] | None = None, background: BackgroundLike | None = None)`

Un nombre vacío lanza `ValueError`; una función o plantilla que no se puede llamar, `TypeError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción

`name`	`str`	obligatorio	Nombre del paso.
`build`	`Callable[[Scene], None]`	obligatorio	Función que crea el contenido del paso.
`transition, notes, template, background`		`None`	Igual que en `Scene.segment`.

Devuelve: `SectionStep` — Valor inmutable.

35.1.50. SectionProgress

CLASE `SectionProgress(key, step, index, total, visit, segment)`

`key` y `step` identifican la sección y el paso; `index` (desde 1) y `total` cuentan los pasos; `visit` (desde 1) cuenta las repeticiones de la sección; `segment` es el `Segment` recién abierto, útil para `bind`. `fraction` vale `index / total`. Las pausas dentro de un paso no avanzan el progreso.

Devuelve: `SectionProgress` — Contexto que recibe `on_enter`.

35.1.51. Scene.sections

PROPIEDAD `sections: SceneSections`

Construyen las dos piezas de navegación de una charla sobre una base neutra de colores del tema. Cada parte es un drawable normal que puedes reestilizar, animar u ocultar. Las secciones pueden ser instancias de `Section`, claves o pares `(clave, título)`.

Devuelve: `SceneSections` — Fábricas `agenda(...)` y `progress_rail(...)`.

Ejemplo:

```
agenda = scene.sections.agenda(sections, pitch=0.66)
agenda.root.move_to(-2, 1.4, Anchor.TOP_LEFT)
rail = scene.sections.progress_rail(sections, segmented=True, captions=True)
rail.root.move_to(0, -4.3).hud()
scene.persist(rail.root)

for section in sections:
    scene.play([agenda.animate.focus(section), rail.animate.enter(section)])
    section.build(scene, on_enter=lambda scene, p: scene.play(rail.animate.to(p)))
```

Agenda. `scene.sections.agenda(sections, current=None, *, direction="column", pitch=0.5, styles=None, item=None, marker=None, marker_gap=0.3)` devuelve un `Agenda`. Cada

entrada está en uno de tres estados, `done`, `current` o `upcoming`, y se coloca por su punto izquierdo--central a `pitch` unidades de la anterior, en columna o en fila. Cada entrada guarda un drawable por estado y solo muestra el suyo, así que un estado puede cambiar peso, color o contenido: `focus` hace un fundido cruzado entre ellos. Por defecto `done` usa `foreground`, `current` usa `accent` con peso 700 y `upcoming` usa `muted`; `styles={"current": TextStyle(...)}` sustituye cualquiera de ellos e `item=lambda scene, entry, state: ...` sustituye el texto por cualquier drawable. `marker=lambda scene: ...` añade un indicador a `marker_gap` de la entrada actual que se desliza con `focus`. Para anotar esas funciones, `NavigationEntry` y `NavigationState` se importan desde `gaanim`.

- `agenda.item(key)` es el grupo de una entrada, `agenda.items("done")` las entradas en un estado y `agenda.variant(key, "current")` el drawable de ese estado.
- `agenda.focus(key)` y `agenda.advance(steps)` cambian el estado de inmediato; `agenda.animate.focus(key)` y `agenda.animate.advance()` devuelven composiciones para `scene.play`, con `easing=` opcional. Los destinos aceptan una clave, un índice desde 0, un `Section` o un `SectionProgress`.

Barra de progreso. `scene.sections.progress_rail(sections=None, *, length=12.0, thickness=0.08, orientation="horizontal", segmented=False, ...)` devuelve un `ProgressRail`. Una barra continua es una sola pista con una marca en cada límite de sección; `segmented=True` da a cada sección su propia pista, separadas por `gap`. `rail.animate.to(progress)` llena las secciones anteriores a un `SectionProgress` y su parte de pasos; un número en `[0, 1]` fija la fracción de toda la barra. `rail.animate.enter(section)` marca una sección como actual sin llenar nada, como en una diapositiva divisoria. `captions=True` nombra cada sección sobre su tramo y la colorea por estado. Las partes son `track`, `fills`, `marks`, `captions` y `label`, y los argumentos `track=`, `mark=` y `caption=` sustituyen sus formas. Las barras se llenan de izquierda a derecha, o hacia arriba con `orientation="vertical"`.

35.1.52. Salida

35.1.53. Scene.render

MÉTODO `render() -> None`

Llámalo una vez, al final del script. Qué se hace con la escena (previsualizarla, validarla, exportarla o capturarla) lo decide el comando que ejecuta el script. Fuera de la aplicación de Gaanim, por ejemplo con `python main.py`, lanza `RuntimeError`.

Devuelve: `None` — Entrega la línea de tiempo al ejecutable de Gaanim.

Ejemplo:

```python
from gaanim import Scene

scene = Scene(frame=(16, 9))
scene.wait(0.5)
scene.render()
```

35.1.54. Scene.snapshots

MÉTODO `snapshots(directory: str, times: Sequence[float]) -> int`

Pide al ejecutable que capture seeks exactos para la comparación visual. Sin `gaanim --diff`, o con otra ruta, lanza `RuntimeError`. Consulta Capturas y comparación visual.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`directory`	`str`	obligatorio	La ruta que `gaanim --diff` pasa en `GAANIM_SNAPSHOTS`.
`times`	`Sequence[float]`	obligatorio	Instantes absolutos que se capturan.

Devuelve: `int` — Número de fotogramas capturados.

Ejemplo:

```
import os

if snapshots := os.environ.get("GAANIM_SNAPSHOTS"):
    scene.snapshots(snapshots, [0.0, 1.0])
```

35.1.55. Scene.thumbnail

MÉTODO `thumbnail(time: Optional[float] = None) -> None`

Elige el fotograma que un archivo `.gaanim` guarda como portada, la imagen que el explorador de archivos muestra tras `gaanim register`, y la imagen del proyecto en el Inicio. Sin llamarlo, la portada es el fotograma de la primera pausa o, sin pausas, el del primer segmento que más muestra. Un `time` negativo o no finito lanza `ValueError`. Consulta Compartir sin Python.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`time`	`float | None`	`None`	Instante de la línea de tiempo, en segundos; `None` toma el cursor.

Devuelve: `None` — Solo marca el instante.

Ejemplo:

```
titulo = scene.text("Resultados").scale_by(2)
scene.play(titulo.animate.write())
scene.thumbnail()
```

35.1.56. Variación reproducible

35.1.57. Scene.random

MÉTODO `random(seed: int = 0) -> Random`

La misma semilla da los mismos valores en cualquier plataforma, así que la escena sale igual en la previsualización y en la exportación.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`seed`	`int`	`0`	Semilla del generador.

Devuelve: `Random` — Un generador con semilla.

Ejemplo:

```
rng = scene.random(seed=42)
stars = [scene.geometry.dot(0.04).move_to(rng.uniform(-7, 7), rng.uniform(-4, 4)) for _ in
range(60)]
```

35.1.58. Random.uniform

MÉTODO `uniform(low: float = 0.0, high: float = 1.0) -> float`

`high < low` lanza `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`low`	`float`	`0.0`	Límite inferior, incluido.
`high`	`float`	`1.0`	Límite superior, excluido.

Devuelve: `float` — Un número en `[low, high)`.

35.1.59. Random.gauss

MÉTODO `gauss(mean: float = 0.0, std: float = 1.0) -> float`

Una desviación negativa lanza `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`mean`	`float`	`0.0`	Media.
`std`	`float`	`1.0`	Desviación típica.

Devuelve: `float` — Un valor de una distribución normal.

35.1.60. Random.integer

MÉTODO `integer(low: int, high: int) -> int`

Un rango vacío lanza `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción

`low`	`int`	obligatorio	Límite inferior, incluido.
`high`	`int`	obligatorio	Límite superior, excluido.

Devuelve: `int` — Un entero en `[low, high)`.

35.1.61. Random.choice

MÉTODO `choice(items: Sequence[Any]) -> Any`

Una secuencia vacía lanza `IndexError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`items`	`Sequence[Any]`	obligatorio	Elementos entre los que elegir.

Devuelve: `Any` — Uno de los elementos.

35.1.62. Random.shuffle

MÉTODO `shuffle(items: list[Any]) -> None`

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`items`	`list[Any]`	obligatorio	Lista que se baraja.

Devuelve: `None` — Modifica la lista.

Ejemplo:

```
rng = scene.random(seed=7)
colors = [BLUE, GOLD, CYAN]
rng.shuffle(colors)
size = rng.gauss(1.0, 0.1)
count = rng.integer(3, 8)
accent = rng.choice(colors)
```

35.1.63. Random.seed

PROPIEDAD `seed: int`

Devuelve: `int` — La semilla con la que se creó el generador.

35.1.64. Scene.noise

MÉTODO `noise(*, frequency: float = 1.0, amplitude: float = 1.0, octaves: int = 1, seed: int = 0, center: float = 0.0) -> Computed`

Ruido simplex fractal con semilla, evaluado de forma nativa en cada fotograma sin llamar a Python; la reproducción, los seeks y la exportación coinciden. Valores inválidos lanzan `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`frequency`	`float`	`1.0`	Rapidez con que cambia el valor.
`amplitude`	`float`	`1.0`	El valor queda en `[-amplitude, amplitude]` alrededor de `center`.
`octaves`	`int`	`1`	Capas de detalle fino, de 1 a 8.
`seed`	`int`	`0`	Semilla del ruido.
`center`	`float`	`0.0`	Valor central.

Devuelve: `Computed` — Un escalar reactivo que depende del tiempo de la línea de tiempo.

Ejemplo:

```
drift = scene.noise(frequency=0.6, amplitude=0.3, octaves=3, seed=5)
title.rotate_to(computed(lambda v: 0.1 * v, inputs=[drift]))
scene.wait(2)
```

35.1.65. Cámara

`scene.camera` controla qué parte del mundo se ve. Sus métodos directos aplican un corte en el cursor, sin avanzar el tiempo, y devuelven la cámara para encadenar. Los mismos métodos bajo `scene.camera.animate` devuelven un `Anim` que se programa con `scene.play` junto a cualquier otra animación y admite `.duration()`, `.delay()` y `.easing()`.

```
from gaanim import BLUE, GOLD, Scene

scene = Scene(frame=(16, 9), background="#0f172a")
circle = scene.geometry.circle(0.8).fill(BLUE).move_to(-4, 1)
square = scene.geometry.square(1.2).fill(GOLD).move_to(4, -1)
scene.play([scene.camera.animate.frame_to(circle, margin=0.6).duration(0.8)])
scene.play([scene.camera.animate.frame_to(square, margin=0.6).duration(0.8)])
scene.play([scene.camera.animate.reset().duration(0.6)])
scene.render()
```

Los destinos de posición (`pan_to`, `follow`, `look_at`, `bind_*`) aceptan un `Drawable`, un `AnchorPoint`, un `PointRef` o una tupla 2D o 3D. El zoom y la rotación aceptan también `Parameter`, `Variable` y `Computed`. Las APIs de cámara rechazan valores no finitos, zooms, campos de visión y planos de recorte inválidos, poses de `look_at` degeneradas e influencias fuera de `[0, 1]` con `ValueError`, sin recortar valores en silencio.

35.1.65.1. Poses guardadas

35.1.66. Camera.state_2d

MÉTODO `state_2d(center: tuple[float, float] = (0.0, 0.0), zoom: float = 1.0, rotation: float = 0.0) -> CameraState`

Valida la pose sin avanzar la línea de tiempo. Un `CameraState` pertenece a la escena que lo creó y no depende del tamaño del lienzo ni de la ventana.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`center`	`tuple[float, float]`	`(0.0, 0.0)`	Punto que queda en el centro de la vista.
`zoom`	`float`	`1.0`	Zoom ortográfico; más de 1 acerca.
`rotation`	`float`	`0.0`	Giro de la vista en radianes.

Devuelve: `CameraState` — Una pose ortográfica reutilizable.

Ejemplo:

```
detail = scene.camera.state_2d(center=(-1.6, 0.4), zoom=1.5)
scene.play([scene.camera.animate.to(detail).duration(0.8)])
```

35.1.67. Camera.capture

MÉTODO `capture() -> CameraState`

No tiene duración. Registra la pose escrita en el script antes de los bindings, los efectos temporales, el shake y la vista de inspección del editor, así que restaurarla da el mismo resultado al previsualizar, buscar y exportar.

Devuelve: `CameraState` — La pose que la cámara tiene en el cursor.

35.1.68. Camera.save

MÉTODO `save(name: str) -> CameraState`

Captura la pose actual y la guarda con un nombre para `restore`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`name`	`str`	obligatorio	Nombre no vacío; reemplaza una pose guardada con el mismo nombre.

Devuelve: `CameraState` — La pose capturada.

35.1.69. Camera.to

MÉTODO `to(state: CameraState) -> Camera`

Salta a la pose en el cursor. Una pose de otra escena lanza `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`state`	`CameraState`	obligatorio	Pose de esta escena.

Devuelve: `Camera` — La misma cámara.

35.1.70. Camera.restore

MÉTODO `restore(name: str) -> Camera`

Salta a una pose guardada. Un nombre desconocido lanza `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`name`	`str`	obligatorio	Nombre usado en `save`.

Devuelve: `Camera` — La misma cámara.

Ejemplo:

```
scene.camera.save("general")
scene.camera.zoom_to(2.0)
scene.wait(0.5)
scene.camera.restore("general")
```

35.1.70.1. Encuadre

35.1.71. Camera.pan_to

MÉTODO `pan_to(x: float, y: float) -> Camera | pan_to(target: Endpoint) -> Camera`

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`x, y`	`float`	obligatorio	Punto que queda en el centro de la vista.
`target`	`Endpoint`	obligatorio	Objeto, punto de anclaje o tupla que se centra.

Devuelve: `Camera` — La misma cámara.

35.1.72. Camera.zoom_to

MÉTODO `zoom_to(zoom: ScalarSource) -> Camera`

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`zoom`	`ScalarSource`	obligatorio	Zoom ortográfico; más de 1 acerca y menos de 1 aleja. Debe ser positivo.

Devuelve: `Camera` — La misma cámara.

35.1.73. Camera.frame_to

MÉTODO `frame_to(targets: Drawable | Sequence[Drawable], margin: float | tuple[float, float] | tuple[float, float, float, float] | None = None, *, dynamic: bool = False) -> Camera`

Centra y ajusta el zoom para que los objetos quepan en la vista.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`targets`	`Drawable | Sequence[Drawable]`	obligatorio	Objetos que deben caber en la vista.
`margin`	`float | tuple`	`None`	Margen alrededor en unidades de escena: un valor, `(vertical, horizontal)` o cuatro

			lados en el orden de CSS. Sin él, media unidad por lado.
`dynamic`	`bool`	`False`	Recalcula la unión en cada fotograma, después de los updaters y del layout.

Devuelve: `Camera` — La misma cámara.

35.1.74. Camera.rotate_to

MÉTODO `rotate_to(angle: ScalarSource) -> Camera`

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`angle`	`ScalarSource`	obligatorio	Giro de la vista en radianes.

Devuelve: `Camera` — La misma cámara.

35.1.75. Camera.orthographic

MÉTODO `orthographic(zoom: float = 1.0) -> Camera`

Selecciona la proyección ortográfica, la de una escena 2D.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`zoom`	`float`	`1.0`	Zoom positivo.

Devuelve: `Camera` — La misma cámara.

35.1.76. Camera.reset

MÉTODO `reset() -> Camera`

Restaura la pose 2D predeterminada, el vector vertical, el objetivo y la proyección ortográfica.

Devuelve: `Camera` — La misma cámara.

Ejemplo:

```
scene.camera.pan_to(circle).zoom_to(1.5).rotate_to(0.1)
scene.wait(0.5)
scene.camera.frame_to(circle, margin=0.4)
scene.wait(0.5)
scene.camera.reset()
```

35.1.77. Camera.animate

PROPIEDAD `animate: CameraAnimation`

Cada método devuelve un `Anim` nuevo; construirlo no cambia nada hasta pasarlo a `scene.play`. Un movimiento dura un segundo salvo que `.duration()` fije otra; la `duration` de `scene.play` sustituye ese valor por defecto, como en las demás animaciones. `shake` (medio segundo la sacudida senoidal) y `whip_pan` (0,4 s) tienen sus propios valores por defecto, que `scene.play` también sustituye; una sacudida con trauma dura hasta apagarse.

Devuelve: `CameraAnimation` — El vocabulario animado de la cámara.

35.1.77.1. Movimientos animados

35.1.78. CameraAnimation.to

MÉTODO `to(state: CameraState) -> Anim`

Entre una pose ortográfica y una en perspectiva, el movimiento va en perspectiva desde o hacia la vista equivalente a la ortográfica, y solo es ortográfico en su extremo ortográfico: el contenido 3D queda siempre delante del ojo. Una pose de otra escena lanza `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`state`	`CameraState`	obligatorio	Pose de esta escena.

Devuelve: `Anim` — Movimiento hasta la pose.

35.1.79. CameraAnimation.restore

MÉTODO `restore(name: str) -> Anim`

Un nombre desconocido lanza `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`name`	`str`	obligatorio	Nombre usado en `Camera.save`.

Devuelve: `Anim` — Movimiento hasta la pose guardada.

Ejemplo:

```
scene.camera.save("general")
scene.play([scene.camera.animate.zoom_to(2.0).duration(0.6)])
scene.play([scene.camera.animate.restore("general").duration(0.6)])
```

35.1.80. CameraAnimation.pan_to

MÉTODO `pan_to(x: float, y: float) -> Anim | pan_to(target: Endpoint) -> Anim`

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`x, y`	`float`	obligatorio	Punto que termina en el centro de la vista.
`target`	`Endpoint`	obligatorio	Objeto, punto de anclaje o tupla que se centra.

Devuelve: `Anim` — Desplazamiento de la vista.

Ejemplo:

```
scene.play([scene.camera.animate.pan_to(-1.6, 0.4).duration(0.8)])
```

35.1.81. CameraAnimation.zoom_to

MÉTODO `zoom_to(zoom: ScalarSource, *, interpolation: Literal["exponential", "linear"] = "exponential") -> Anim`

Con `"exponential"` el zoom sigue `z0 * (z1 / z0) ** p`, donde `p` es el progreso con easing: el área visible cambia en la misma proporción en cada fotograma y un zoom de 8× se lee a velocidad constante. `"linear"` usa `z0 + (z1 - z0) * p`. Un zoom constante no positivo o una interpolación desconocida lanzan `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`zoom`	`ScalarSource`	obligatorio	Zoom final; más de 1 acerca.
`interpolation`	`"exponential" | "linear"`	`"exponential"`	Cómo se interpola el zoom.

Devuelve: `Anim` — Zoom animado.

Ejemplo:

```
scene.play([scene.camera.animate.zoom_to(8.0).duration(1.5)])
scene.play([scene.camera.animate.zoom_to(1.0, interpolation="linear").duration(0.8)])
```

35.1.82. CameraAnimation.frame_to

MÉTODO `frame_to(targets: Drawable | Sequence[Drawable], margin: float | tuple[float, float] | tuple[float, float, float, float] | None = None, *, dynamic: bool = False, interpolation: Literal["exponential", "linear"] = "exponential") -> Anim`

Parámetros:

Nombre	Tipo	Predeterminado	Descripción

`targets`	`Drawable | Sequence[Drawable]`	obligatorio	Objetos que deben caber en la vista.
`margin`	`float | tuple`	`None`	Igual que en `Camera.frame_to`.
`dynamic`	`bool`	`False`	Sigue a los objetos si se mueven durante el movimiento, y al terminar la cámara conserva el último encuadre.
`interpolation`	`"exponential" | "linear"`	`"exponential"`	Con `"exponential"` el zoom es exponencial y el desplazamiento acompaña al cambio de ancho visible, así que la vista escala alrededor de un punto fijo y el contenido viaja en línea recta. `"linear"` interpola posición y zoom por separado.

Devuelve: `Anim` — Encuadre animado.

Ejemplo:

```
scene.play([scene.camera.animate.frame_to([circle, label], margin=(0.32, 0.48),
dynamic=True).duration(0.9)])
```

35.1.83. CameraAnimation.rotate_to

MÉTODO `rotate_to(angle: ScalarSource) -> Anim`

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`angle`	`ScalarSource`	obligatorio	Giro final de la vista en radianes.

Devuelve: `Anim` — Giro animado.

Ejemplo:

```
scene.play([scene.camera.animate.rotate_to(0.15).duration(0.5)])
```

35.1.84. CameraAnimation.follow

MÉTODO `follow(target: Endpoint, *, offset: tuple[float, float] = (0.0, 0.0), offset_space: Literal["world", "local"] = "world", lag: float = 0.0) -> Anim`

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`target`	`Endpoint`	obligatorio	Objeto o punto que se sigue.
`offset`	`tuple[float, float]`	`(0.0, 0.0)`	Desplazamiento de la vista respecto del objetivo.
`offset_space`	`"world" | "local"`	`"world"`	`"local"` gira el desplazamiento con el objetivo.
`lag`	`float`	`0.0`	Retraso de seguimiento en segundos, determinista.

Devuelve: `Anim` — Seguimiento durante la duración del `Anim`. Al terminar, la cámara se queda en la pose que alcanzó, calculada con la posición del objetivo al final del `play`, y el siguiente movimiento parte de ahí.

Ejemplo:

```
scene.play([
    circle.animate.move_to(4, 0).duration(2.0),
    scene.camera.animate.follow(circle.anchor_point(Anchor.TOP), offset=(0, 0.24),
lag=0.2).duration(2.0),
])
```

35.1.85. CameraAnimation.shake

MÉTODO `shake(amplitude: Optional[float] = None, frequency: Optional[float] = None, *, trauma: Optional[float] = None, decay: Optional[float] = None, rotation: Optional[float] = None, seed: Optional[int] = None) -> Anim`

Sigue el modelo de trauma: el desplazamiento es proporcional a `trauma ** 2`, así que un golpe leve apenas se nota y uno fuerte sacude mucho. Traslación y giro salen de ruido coherente con semilla. Dura `trauma / decay` segundos (un segundo con `decay=0`); un `.duration()` más corto también vuelve suavemente al reposo, y el easing no cambia el decaimiento. Se suma después de follow, encuadre y bindings, y es función pura del tiempo. Pasar solo `amplitude` (sin `trauma`, `decay`, `rotation` ni `seed`) mantiene la sacudida sinusoidal anterior: `amplitude` es el pico, `frequency` cuenta oscilaciones por clip (8 por defecto) y dura 0.5 s. Valores negativos o `trauma` mayor que 1 lanzan `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`amplitude`	`float | None`	`0.4`	Desplazamiento máximo en unidades de escena con trauma 1.
`frequency`	`float | None`	`12`	Frecuencia del ruido en Hz.
`trauma`	`float | None`	`0.8`	Intensidad inicial, en `[0, 1]`.
`decay`	`float | None`	`1.5`	Trauma que se pierde por segundo.
`rotation`	`float | None`	`0.02`	Giro máximo en radianes con trauma 1.
`seed`	`int | None`	`0`	Semilla del ruido.

Devuelve: `Anim` — Sacudida que termina en reposo.

Ejemplo:

```
scene.play([scene.camera.animate.shake(trauma=0.8, decay=1.5, frequency=12, rotation=0.02,
seed=0)])
scene.play([scene.camera.animate.shake(trauma=0.4, seed=3)])  # un golpe más leve
```

35.1.86. CameraAnimation.follow_path

MÉTODO `follow_path(route: Drawable, *, orient: bool = False, rotate_offset: float = 0.0, start: float = 0.0, end: float = 1.0) -> Anim`

Lee la trayectoria en coordenadas de escena cuando se reproduce la animación, como `Anim.move_along`. El giro de `orient` se calcula en cada fotograma, así que nunca salta. Easing predeterminado: suave. Una ruta de otra escena, `start` o `end` fuera de `[0, 1]` o iguales, o un `rotate_offset` no finito lanzan `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`route`	`Drawable`	obligatorio	Drawable cuya trayectoria recorre la cámara.
`orient`	`bool`	`False`	Gira la vista para que el sentido de avance apunte a la derecha de la pantalla.
`rotate_offset`	`float`	`0.0`	Giro extra en radianes; `-math.pi / 2` apunta el avance hacia arriba.
`start / end`	`float`	`0.0 / 1.0`	Tramo recorrido, como fracciones de longitud de arco; `start > end` lo recorre al revés.

Devuelve: `Anim` — Recorrido de la cámara.

Ejemplo:

```
route = scene.geometry.path([(-6, -2), (-2, 2), (2, -2), (6, 2)])
scene.play([scene.camera.animate.follow_path(route, orient=True).duration(4)])
```

35.1.87. CameraAnimation.whip_pan

MÉTODO `whip_pan(to: Endpoint | float, y: Optional[float] = None, *, blur: bool = True) -> Anim`

Paneo rápido de 0.4 s salvo `.duration`, con easing exponencial de entrada y salida: casi todo el recorrido ocurre en un instante a mitad de camino. Con `blur`, los fotogramas exportados y los snapshots del barrido reciben un desenfoque de movimiento de 360° y 24 subfotogramas aunque la escena no tenga uno propio; la vista previa sigue nítida, como con `canvas.motion_blur`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`to / y`	`Endpoint | float`	obligatorio	Destino, como en `pan_to`.
`blur`	`bool`	`True`	Desenfoque de movimiento en los fotogramas del barrido.

Devuelve: `Anim` — Barrido rápido.

Ejemplo:

```
section_b = scene.text("B", role="title").move_to(16, 0)
scene.play([scene.camera.animate.whip_pan(section_b)])
scene.play([scene.camera.animate.whip_pan(0, 0, blur=False).duration(0.3)])
```

35.1.88. CameraAnimation.orthographic

MÉTODO `orthographic(zoom: float = 1.0) -> Anim`

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`zoom`	`float`	`1.0`	Zoom positivo.

Devuelve: `Anim` — Paso animado a la proyección ortográfica. Desde una cámara en perspectiva, el ojo se aleja por su eje de visión mientras el campo de visión se cierra: el objetivo sigue centrado, la profundidad se aplana y la proyección pasa a ortográfica al terminar, con la misma pose. Desde una cámara ortográfica, anima el zoom.

Ejemplo:

```
cube = scene.geometry.cube(2.0)
scene.camera.look_at(eye=(7, 5, 10), target=(0, 0, 0)).perspective(0.7)
scene.play([scene.camera.animate.orthographic(1.5)], duration=1.0)
```

35.1.89. CameraAnimation.reset

MÉTODO `reset() -> Anim`

Devuelve: `Anim` — Vuelta animada a la pose y proyección predeterminadas. Desde una cámara en perspectiva, el movimiento sigue en perspectiva hacia la vista equivalente a la 2D y pasa a ortográfica al terminar.

Ejemplo:

```
scene.play([scene.camera.animate.zoom_to(2.0).duration(0.5)])
scene.play([scene.camera.animate.reset().duration(0.5)])
```

35.1.89.1. Capas de parallax

Una capa de parallax agrupa drawables que están más lejos o más cerca de la cámara que el plano de la escena. Con la cámara ortográfica, sus miembros se desplazan `1 / depth` del paneo de cámara: el fondo lejano se mueve despacio y el primer plano pasa rápido, sin 3D real. No confundir con `Scene.z_layers` (orden de dibujo con nombre), `Drawable.view_layer` (filtro de vistas de cámara) ni con `.layer(nombre)` de gráficos y ejes (una parte del objeto).

35.1.90. Scene.layer

MÉTODO `layer(depth: float) -> ParallaxLayer`

La capa ve una cámara centrada en `posición / depth` y con zoom `zoom ** (1 / depth)`; la rotación afecta a todas por igual. `depth=1` se comporta como el resto de la escena, `depth=3` se mueve a un tercio de la velocidad de la cámara, `depth=0.6` pasa más rápido y `math.inf` no se desplaza ni hace zoom, como `Drawable.hud` salvo por la rotación. Sacudida, seguimiento y bindings también mueven las capas; un miembro convertido en HUD con `Drawable.hud` sigue fijo en pantalla. Las capas lejanas se dibujan detrás de las cercanas: su `z_index` por defecto es `round(-100 * ln(depth))`, limitado a ±900, y cada miembro suma el suyo. Con cámara en perspectiva las capas no se mueven, porque la profundidad real ya produce parallax. Una profundidad no positiva lanza `ValueError`.

Parámetros:

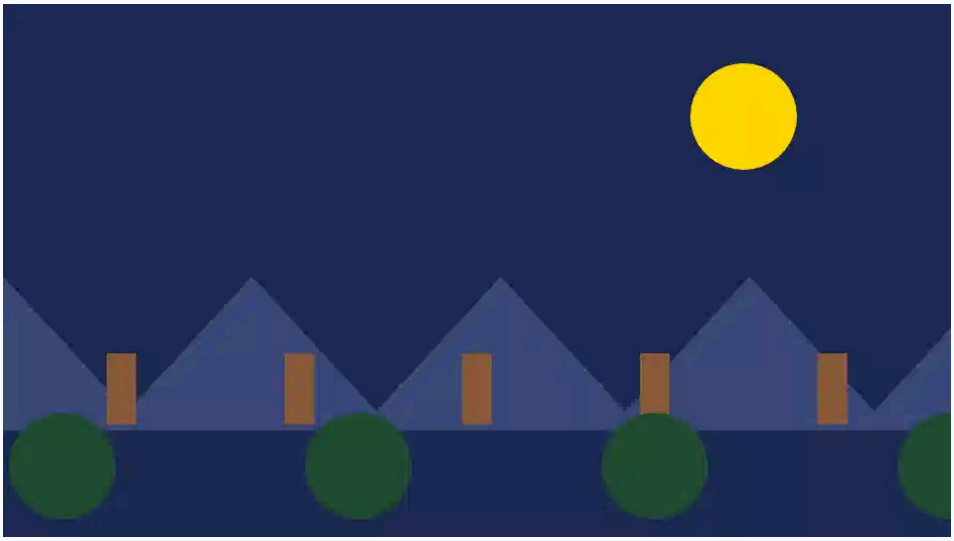
Nombre	Tipo	Predeterminado	Descripción
`depth`	`float`	obligatorio	Distancia a la cámara en unidades de la distancia del plano de la escena; positiva, `math.inf` permitido.

Devuelve: `ParallaxLayer` — Capa vacía; es un `Drawable` cuyos miembros se añaden con `add`.

Ejemplo:

```
import math
from gaanim import *

scene = Scene(frame=(16, 9), background="#1d2b53")
sky = scene.layer(depth=math.inf).add(scene.geometry.circle(0.9).fill(GOLD).move_to(4.5, 2.6))
far = scene.layer(depth=3.0)
for i in range(6):
    far.add(scene.geometry.polygon([(-2.4, 0), (0, 2.6), (2.4, 0)]).fill("#3b4a7a").move_to(-8 + i
* 4.2, -1.4))
for i in range(8):
    scene.geometry.rect(0.5, 1.2).fill("#8a5a3b").move_to(-6 + i * 3, -2.0)
near = scene.layer(depth=0.6)
for i in range(5):
```

```
    near.add(scene.geometry.circle(0.9).fill("#1f4d33").move_to(-7 + i * 5, -3.3))
scene.play([scene.camera.animate.pan_to(6, 0).duration(3)])
scene.render()
```



35.1.91. ParallaxLayer.add

MÉTODO `add(drawable: Drawable, *others: Drawable) -> Self`

Mueve los drawables a la capa conservando su posición actual; añadirlo a otra capa lo mueve allí. Mover, desvanecer o animar la capa actúa sobre todos sus miembros. Un drawable de otra escena, la propia capa u otra capa lanzan `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`drawable`	`Drawable`	obligatorio	Primer drawable que entra en la capa.
`*others`	`Drawable`	obligatorio	Más drawables.

Devuelve: `Self` — La misma capa, para encadenar.

Ejemplo:

```
far = scene.layer(depth=3.0)
far.add(scene.geometry.circle(1.0).move_to(4, 2), scene.geometry.square(1.0).move_to(-4, 2))
print(far.depth)  # 3.0
```

```
3.0
```

35.1.91.1. Profundidad de campo

Las capas de parallax pueden desenfocarse según su distancia al foco, como con una lente real: cada capa se desenfoca `aperture * |1 - focus / depth|` unidades de escena, como máximo `max_blur`. Las capas a la profundidad `focus` quedan nítidas y el desenfoque crece al alejarse de ella hacia delante o hacia atrás; una capa en `math.inf` se desenfoca `aperture`. Los drawables fuera de capas, y los HUD dentro de ellas, nunca se desenfocan. Cada capa se desenfoca como una sola imagen, con

un efecto de shader en su raíz que reemplaza cualquier `shader_effect` puesto en la propia capa; una capa desenfocada se dibuja fuera de pantalla y cuesta más.

35.1.92. Camera.depth_of_field

MÉTODO `depth_of_field(focus: Optional[float] = None, *, aperture: Optional[float] = None, max_blur: Optional[float] = None) -> Camera`

Fija la profundidad de campo en el cursor. Los valores omitidos se conservan; al principio el foco es 1, la apertura 0 (sin desenfoque) y `max_blur` 0.5. Un foco no finito o no positivo, o una `aperture` o `max_blur` negativos o no finitos lanzan `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`focus`	`float | None`	`None`	Profundidad nítida, finita y positiva; `1` es el plano de la escena.
`aperture`	`float | None`	`None`	Escala del desenfoque, en unidades de escena; `0` lo apaga.
`max_blur`	`float | None`	`None`	Desenfoque máximo, en unidades de escena.

Devuelve: `Camera` — La cámara.

Ejemplo:

```
import math
from gaanim import *

scene = Scene(frame=(16, 9), background="#1d2b53")
far = scene.layer(depth=3.0)
for i in range(5):
    far.add(scene.geometry.polygon([(-2.4, 0), (0, 2.6), (2.4, 0)]).fill("#3b4a7a").move_to(-8 + i
* 4.2, -1.2))
for i in range(6):
    scene.geometry.rect(0.5, 1.2).fill("#8a5a3b").move_to(-6.5 + i * 2.6, -2.0)
near = scene.layer(depth=0.5)
for i in range(4):
    near.add(scene.geometry.circle(0.9).fill("#1f4d33").move_to(-6 + i * 4.4, -3.4))
scene.camera.depth_of_field(focus=1.0, aperture=0.12)
scene.wait(0.5)
scene.play([scene.camera.animate.focus_to(3.0).duration(1.5)])
scene.render()
```

35.1.93. CameraAnimation.depth_of_field / focus_to

MÉTODO

`depth_of_field` anima los mismos valores que `Camera.depth_of_field`; `focus_to(focus)` es el cambio de foco (rack focus). El foco se mueve en profundidad inversa, así que pasa de forma pareja por capas cercanas y lejanas; la apertura y `max_blur` cambian linealmente. Easing predeterminado: suave.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`focus`	`float`	obligatorio	Nueva profundidad nítida.

Devuelve: `Anim` — Cambio animado de la profundidad de campo.

Ejemplo:

```
scene.camera.depth_of_field(focus=0.6, aperture=0.12)
scene.play([scene.camera.animate.focus_to(3.0).duration(1.5)])
scene.play([scene.camera.animate.depth_of_field(aperture=0.0).duration(0.5)])
```

35.1.93.1. Bindings persistentes

Un binding es una restricción de cámara que no se dibuja: desde que se crea (salvo con `enabled=False`) escribe los canales que declara en cada fotograma, también a través de segmentos. `influence` es un escalar en `[0, 1]` que mezcla el binding con la pose anterior, y los bindings posteriores se componen sobre los anteriores. `follow` y el encuadre dinámico actúan después, y el shake se suma al final.

35.1.94. Camera.bind_2d

MÉTODO `bind_2d(*, center: Optional[Endpoint] = None, zoom: Optional[ScalarSource] = None, rotation: Optional[ScalarSource] = None,`

`influence: Optional[ScalarSource] = None, enabled: bool = True) -> CameraConstraint`

Necesita `center`, `zoom` o `rotation`, y selecciona la proyección ortográfica.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`center`	`Endpoint | None`	`None`	Punto que sigue el centro de la vista.
`zoom`	`ScalarSource | None`	`None`	Zoom reactivo.
`rotation`	`ScalarSource | None`	`None`	Giro reactivo en radianes.
`influence`	`ScalarSource | None`	`None`	Peso en `[0, 1]`; sin él, 1.
`enabled`	`bool`	`True`	Activo desde el cursor actual.

Devuelve: `CameraConstraint` — El binding, con `enable()` y `disable()`.

Ejemplo:

```
import math

theta = scene.viz.parameter(0.0)
focus = scene.geometry.point_ref(
    computed(lambda t: t * 2.25, inputs=[theta]),
    computed(lambda t: math.sin(t * 2), inputs=[theta]),
)
rig = scene.camera.bind_2d(center=focus, zoom=computed(lambda t: 1 + t * 0.3, inputs=[theta]))
scene.play([theta.animate.set(1.0).duration(2.0)])
rig.disable()
```

35.1.95. CameraConstraint.enable

MÉTODO `enable() -> None`

Devuelve: `None` — Activa el binding en el cursor actual.

35.1.96. CameraConstraint.disable

MÉTODO `disable() -> None`

Los cambios quedan registrados en la línea de tiempo: un seek anterior vuelve a verlo activo.

Devuelve: `None` — Desactiva el binding en el cursor actual.

35.1.96.1. Vistas de cámara

Una vista de cámara muestra dentro de una figura lo que ve una segunda cámara. `Drawable.camera_view` (en Drawable) convierte cualquier figura cerrada en pantalla; `scene.camera.inset` monta de una vez la pantalla, el marco y los conectores. Ambos devuelven un `CameraView`.

35.1.97. Camera.inset

MÉTODO `inset(target: Endpoint, *, zoom: float | Parameter = 2.0, at: Anchor | tuple[float, float] = Anchor.TOP_RIGHT, size: float | tuple[float, float] | None = None, aspect: Optional[float] = None, shape: Literal["rect", "rounded", "circle"] = "rounded", follow: bool = False, connectors: bool = True, color: Optional[ColorLike] = None, fixed: bool = False, background: Paint | Literal["canvas"] | None = "canvas", exclude: Sequence[Drawable] = (), layers: Sequence[str] = ()) -> CameraView`

Muestra un detalle ampliado en una pantalla aparte. El marco sigue el zoom, así que siempre recuadra lo que la pantalla muestra en reposo. Lanza `ValueError` con un zoom o tamaño no positivo, un `shape` o `at` desconocido, o un `target` reactivo sin `follow`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`target`	`Endpoint`	obligatorio	Qué ampliar: un punto, un objeto o un punto de anclaje. Con `follow=True`, cualquier extremo reactivo.
`zoom`	`float | Parameter`	`2.0`	Aumento de la pantalla en reposo.
`at`	`Anchor | tuple[float, float]`	`Anchor.TOP_RIGHT`	Esquina o borde del fotograma donde va la pantalla

			(`Anchor.CENTER` la centra), o punto donde se centra.
`size`	`float | tuple[float, float] | None`	`None`	Ancho de la pantalla, o diámetro si es un círculo; por defecto, el 30 % del ancho de la escena (20 % para un círculo). `size=(ancho, alto)` fija ambos lados de un rectángulo.
`aspect`	`float | None`	`None`	Proporción ancho / alto de un

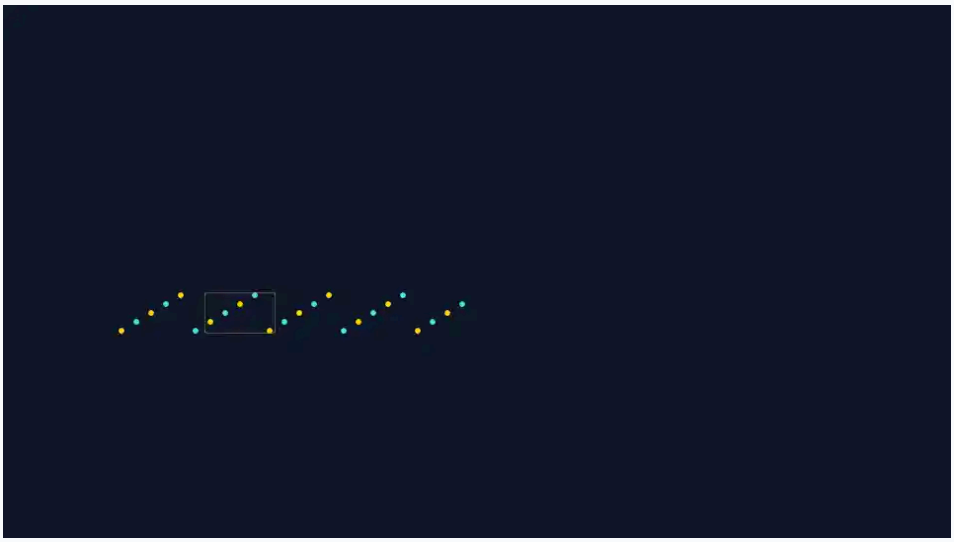
			rectángulo y su marco; por defecto, la del fotograma. Úsala para encuadrar entero un panel alto o una barra de herramientas. Un círculo no admite `aspect`.
`shape`	`str`	`"rounded"`	`"rect"`, `"rounded"` o `"circle"`.
`follow`	`bool`	`False`	El marco sigue a `target` mientras se mueve.
`connectors`	`bool`	`True`	Dos líneas que unen las esquinas enfrentadas del

			marco y la pantalla.
`color`	`Color | None`	`None`	Trazo de la pantalla, el marco y los conectores; por defecto, el acento del tema.
`fixed`	`bool`	`False`	Fija la pantalla a la imagen como un HUD, para que no se mueva con `scene.camera`.
`background / exclude / layers`		obligatorio	Como en `Drawable.camera` Los conectores nunca aparecen en la vista.

Devuelve: `CameraView` — La vista, con sus `connectors`.

Ejemplo:

```
from gaanim import Anchor, CYAN, GOLD, Scene
scene = Scene(frame=(16, 9), background="#0f172a")
for i in range(24):
    scene.geometry.dot(0.05).fill(CYAN if i % 2 else GOLD).move_to(-6 + 0.25 * i, -1 + 0.15 * (i %
5))
view = scene.camera.inset((-4, -0.7), zoom=4, at=Anchor.TOP_RIGHT)
scene.play([view.animate.pop_out().duration(0.8)])
scene.play([view.animate.pan_to(-1.5, -0.7).duration(1.2), view.animate.zoom_to(8).duration(1.2)])
scene.render()
```



35.1.98. CameraView.screen

PROPIEDAD `screen: Drawable`

Devuelve: `Drawable` — La figura que muestra la vista.

35.1.99. CameraView.frame

PROPIEDAD `frame: Drawable`

Devuelve: `Drawable` — El objeto que hace de cámara; se anima como cualquier otro.

35.1.100. CameraView.connectors

PROPIEDAD `connectors: list[Drawable]`

Devuelve: `list[Drawable]` — Las líneas de un inset; vacía en otras vistas.

35.1.101. CameraView.zoom

PROPIEDAD `zoom: Parameter | Computed`

Una vista creada con un número devuelve un `Computed`; con un `Parameter` o `Computed`, lo devuelve tal cual. Si el tamaño del marco fija el zoom, lanza `ValueError`.

Devuelve: `Parameter | Computed` — El zoom como valor reactivo, para lecturas y vínculos.

Ejemplo:

```
label = scene.viz.readout(view.zoom, format=".1f", prefix="x")
```

35.1.102. CameraView.pan_to

MÉTODO `pan_to(x: float, y: float) -> CameraView`

Apunta la cámara en el cursor actual: mueve el centro del marco.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`x, y / target`	`float / Endpoint`	obligatorio	Coordenadas, un objeto o un punto de anclaje.

Devuelve: `CameraView` — La misma vista.

35.1.103. CameraView.zoom_to

MÉTODO `zoom_to(zoom: float) -> CameraView`

Fija el zoom en el cursor actual. Si el tamaño del marco fija el zoom, escala el marco para alcanzarlo, medido con su `fit` sobre ambos tamaños en ese punto. Un zoom no positivo, o una vista que sigue un `Computed`, lanza `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`zoom`	`float`	obligatorio	Aumento, mayor que cero.

Devuelve: `CameraView` — La misma vista.

35.1.104. CameraView.rotate_to

MÉTODO `rotate_to(radians: ScalarSource) -> CameraView`

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`radians`	`float`	obligatorio	Giro de la cámara; la vista gira en sentido contrario.

Devuelve: `CameraView` — La misma vista.

35.1.105. CameraView.follow

MÉTODO `follow(target: Endpoint, *, offset: tuple[float, float] = (0.0, 0.0)) -> CameraView`

Mantiene la cámara sobre `target` desde el cursor; `pan_to` deja de tener efecto después, y los movimientos anteriores se siguen viendo.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`target`	`Endpoint`	obligatorio	Qué seguir.
`offset`	`tuple[float, float]`	`(0, 0)`	Desplazamiento en unidades de escena.

Devuelve: `CameraView` — La misma vista.

35.1.106. CameraView.animate

PROPIEDAD `animate: CameraViewAnimation`

Devuelve: `CameraViewAnimation` — Proxy de animación; sus resultados se pasan a `scene.play`.

35.1.107. CameraViewAnimation.zoom_to

MÉTODO `zoom_to(zoom: float) -> Anim`

El zoom propio de la vista cambia a velocidad percibida constante; un `Parameter` se anima de forma lineal y una vista con zoom de marco escala el marco.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`zoom`	`float`	obligatorio	Aumento final.

Devuelve: `Anim` — La animación.

35.1.108. CameraViewAnimation.pan_to / rotate_to

MÉTODO

Versiones animadas de `CameraView.pan_to` y `CameraView.rotate_to`.

Devuelve: `Anim` — La animación.

35.1.109. CameraViewAnimation.pop_out

MÉTODO `pop_out() -> Anim`

La pantalla sale de la región que ve su cámara y crece hasta su sitio. Encogida sobre esa región muestra la escena a tamaño real, así que la vista brota de la escena sin saltos. Si es la primera

animación de la pantalla, hace de entrada: hasta que empieza, la pantalla (y en un inset, el marco y los conectores) está oculta. Tras `pop_in`, la devuelve a donde estaba y los vuelve a mostrar.

Devuelve: `Anim` — La animación.

35.1.110. CameraViewAnimation.pop_in

MÉTODO `pop_in() -> Anim`

Encoge la pantalla de vuelta a la región que ve su cámara y, al llegar, la oculta junto con el marco y los conectores de un inset, así que una capa de vista deja de verse sobre la escena. Es la salida simétrica de `pop_out`; el siguiente `pop_out` los vuelve a mostrar. En una vista de `camera_view`, el marco es tuyo y conserva su visibilidad.

Devuelve: `Anim` — La animación.

35.1.110.1. Cámara 3D

EXPERIMENTAL

La API 3D es experimental: todavía tiene errores conocidos y puede cambiar entre versiones. Úsala para explorar, pero no la tomes como estable.

La cámara 3D usa coordenadas de mundo `(x, y, z)` y ángulos en radianes. Las primitivas 3D están en Geometría. Para una introducción, consulta la guía Cámara y 3D.

35.1.111. Camera.state_3d

MÉTODO `state_3d(eye: tuple[float, float, float], target: tuple[float, float, float], up: tuple[float, float, float] = (0.0, 1.0, 0.0), fov_y: float = 0.7853981633974483, near: float = 0.1, far: float = 1000.0) -> CameraState`

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`eye`	`tuple[float, float, float]`	obligatorio	Posición de la cámara.
`target`	`tuple[float, float, float]`	obligatorio	Punto al que mira.
`up`	`tuple[float, float, float]`	`(0.0, 1.0, 0.0)`	Dirección vertical del mundo.
`fov_y`	`float`	`0.785…`	Campo de visión vertical en radianes (π/4).
`near, far`	`float`	`0.1, 1000.0`	Planos de recorte, con `0 < near < far`.

Devuelve: `CameraState` — Una pose en perspectiva reutilizable.

35.1.112. Camera.perspective

MÉTODO `perspective(fov_y: float, near: float = 0.1, far: float = 1000.0) -> Camera`

Cambia a proyección en perspectiva en el cursor.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`fov_y`	`float`	obligatorio	Campo de visión vertical en radianes, en `(0, pi)`.
`near`	`float`	`0.1`	Plano de recorte cercano, positivo.
`far`	`float`	`1000.0`	Plano de recorte lejano, mayor que `near`.

Devuelve: `Camera` — La misma cámara.

Ejemplo:

```
scene.camera.perspective(fov_y=0.785, near=0.1, far=1000.0)
```

35.1.113. Camera.look_at

MÉTODO `look_at(eye: Endpoint, target: Endpoint, up: Optional[tuple[float, float, float]] = None) -> Camera`

Coloca la cámara en `eye` mirando a `target`. Los extremos se resuelven después del layout reactivo; `eye` y `target` deben ser distintos y `up` no puede ser nulo ni paralelo a la dirección de la mirada.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`eye`	`Endpoint`	obligatorio	Posición de la cámara en el mundo.
`target`	`Endpoint`	obligatorio	Punto al que mira.
`up`	`tuple[float, float, float] | None`	`None`	Dirección vertical; `(0, 1, 0)` si se omite.

Devuelve: `Camera` — La misma cámara.

Ejemplo:

```
scene.camera.look_at(eye=(7, 5, 6), target=(0, 0, 0))
```

35.1.114. Camera.bind_3d

MÉTODO `bind_3d(*, eye: Optional[Endpoint] = None, target: Optional[Endpoint] = None, fov_y: Optional[ScalarSource] = None, up: tuple[float, float, float] = (0.0, 1.0, 0.0), influence: Optional[ScalarSource] = None, enabled: bool = True) -> CameraConstraint`

Necesita `eye`, `target` o `fov_y`, y selecciona la proyección en perspectiva.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`eye`	`Endpoint | None`	`None`	Posición reactiva de la cámara.
`target`	`Endpoint | None`	`None`	Punto reactivo al que mira.
`fov_y`	`ScalarSource | None`	`None`	Campo de visión reactivo en radianes.
`up`	`tuple[float, float, float]`	`(0.0, 1.0, 0.0)`	Dirección vertical.
`influence`	`ScalarSource | None`	`None`	Peso en `[0, 1]`.
`enabled`	`bool`	`True`	Activo desde el cursor actual.

Devuelve: `CameraConstraint` — El binding.

Ejemplo:

```
rig = scene.camera.bind_3d(eye=(6, 4, 8), target=(0, 0, 0), fov_y=0.8, influence=0.75)
scene.wait(1)
rig.disable()
```

35.1.115. CameraAnimation.look_at

MÉTODO `look_at(eye: Endpoint, target: Endpoint, up: Optional[tuple[float, float, float]] = None) -> Anim`

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`eye`	`Endpoint`	obligatorio	Posición final de la cámara.
`target`	`Endpoint`	obligatorio	Punto al que mira al final.
`up`	`tuple[float, float, float] | None`	`None`	Dirección vertical; `(0, 1, 0)` si se omite.

Devuelve: `Anim` — Movimiento animado de la cámara.

35.1.116. CameraAnimation.orbit

MÉTODO `orbit(delta_yaw: float, delta_pitch: float) -> Anim`

Usa incrementos pequeños para un giro suave.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`delta_yaw`	`float`	obligatorio	Giro horizontal alrededor del objetivo, en radianes.
`delta_pitch`	`float`	obligatorio	Giro vertical, en radianes.

Devuelve: `Anim` — Órbita alrededor del objetivo actual de `look_at`.

Ejemplo:

```
scene.play([scene.camera.animate.orbit(delta_yaw=0.5, delta_pitch=0.1).duration(1.0)])
```

35.1.117. CameraAnimation.perspective

MÉTODO `perspective(fov_y: float, near: float = 0.1, far: float = 1000.0) -> Anim`

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`fov_y`	`float`	obligatorio	Campo de visión vertical final, en radianes.
`near`	`float`	`0.1`	Plano cercano.
`far`	`float`	`1000.0`	Plano lejano.

Devuelve: `Anim` — Paso animado a la perspectiva.

35.1.118. CameraAnimation.dolly

MÉTODO `dolly(factor: float) -> Anim`

Un factor menor que 1 acerca la cámara y uno mayor la aleja.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`factor`	`float`	obligatorio	Multiplicador positivo de la distancia al objetivo.

Devuelve: `Anim` — Acercamiento o alejamiento.

Ejemplo:

```
scene.play([scene.camera.animate.dolly(factor=0.85).duration(0.6)])
```

35.1.119. CameraAnimation.dolly_zoom

MÉTODO `dolly_zoom(factor: float) -> Anim`

La cámara avanza o retrocede por su línea de visión hasta `factor` veces su distancia al objetivo de `look_at` mientras el campo de visión cambia para que el plano del objetivo conserve su tamaño en pantalla: el fondo se estira (`factor` mayor que 1) o se precipita. La distancia cambia exponencialmente. Solo el contenido 3D lo muestra, porque la capa 2D no se mueve con una cámara en perspectiva. Sin `perspective` o con la cámara sobre su objetivo avisa y no hace nada. Un `factor` no finito o no positivo lanza `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`factor`	`float`	obligatorio	Multiplicador positivo de la distancia al objetivo.

Devuelve: `Anim` — Efecto vértigo.

Ejemplo:

```
scene.camera.perspective(fov_y=0.6)
scene.camera.look_at(eye=(0, 1, 8), target=(0, 0, 0))
scene.play([scene.camera.animate.dolly_zoom(1.8).duration(2.0)])
```

35.1.119.1. Cámara de inspección del editor

El editor tiene una cámara de inspección separada de `scene.camera`, que nunca cambia las capturas, Presenter View ni la exportación. Empieza desactivada; cada vez que se activa con `I` (o con el botón **Interactive** de la barra de overlays, `O`) parte de una copia de la cámara de la escena en ese instante.

- `Num0`: alterna entre **Free 3D** y **Camera View** (la cámara de la escena).
- En 3D, arrastrar con el botón derecho: orbitar; con el central o Mayús + izquierdo: desplazar. En 2D, arrastrar con cualquier botón desplaza la vista y la escena sigue al cursor; un clic sin arrastrar selecciona el objeto.
- Rueda: hacia arriba acerca y hacia abajo aleja. En 2D acerca hacia el punto bajo el cursor, que se queda quieto.
- `W`, `A`, `S`, `D`: desplazan la cámara. Las flechas siguen navegando entre paradas y escenas.
- Mientras la inspección está activa, un clic en la vista previa no avanza a la siguiente parada; `Enter` y las flechas sí. Arrastrar o usar la rueda sobre los paneles del editor no mueve la cámara.
- `F`: encuadra la selección, o toda la escena si no hay nada seleccionado.
- `R`: restablece y encuadra; `I`: activa o desactiva la inspección.

El marco de salida visible conserva el marco lógico y su proporción, y la selección con el ratón se limita a ese marco. Mientras la escena tiene contenido 3D, la barra de tiempo desactiva el ajuste magnético.

35.1.119.2. Overlays del editor

`O` muestra una barra con guías que se dibujan solo en el editor, nunca en las capturas ni en la exportación. `O` o `Esc` la ocultan.

- **Bounds** (`B`): el marco de salida y su tamaño en unidades lógicas.
- **Coordinates** (`C`): los ejes con sus marcas y el punto bajo el cursor en unidades de la escena. Mantén Mayús para ajustarlo a la grilla y pulsa `Ctrl+C` (`Cmd+C` en macOS) para copiarlo como tupla, lista para pegar en `move_to((x, y))`. La etiqueta se oculta sobre los paneles del editor.
- **Grid** (`G`): líneas cada 1, 2 o 5 unidades (o sus potencias de diez), según el zoom. Durante la inspección cubren toda la vista, no solo el marco. En 3D es una grilla sobre el plano XZ alrededor del punto que mira la cámara, con el eje X en rojo y el Z en azul.

- **Guides** (`M`): márgenes seguros de acción (90 %) y de títulos (80 %), y los tercios del marco.
- **Layout** (`K`): inspecciona las cajas como las herramientas de desarrollo de un navegador. Cada caja lleva un contorno discontinuo y las zonas del segmento actual un contorno turquesa con su nombre. Al pasar el cursor por una caja se colorean su padding (verde), las celdas de sus hijos (azul), sus márgenes (naranja) y los gaps entre ellos (morado), con una etiqueta de su tipo, tamaño, padding y gap en px de diseño. Sigue la caja en cada momento de la línea de tiempo, también mientras se reorganiza.
- El objeto seleccionado se enmarca con su centro y su tamaño en unidades, y el **Inspector** muestra sus propiedades (ver más abajo).
- Con la inspección activa, la barra muestra el zoom de la vista (o **Free 3D**) en azul cuando ya no coincide con la cámara de la escena, y un botón para restablecerla (`R`). El botón de captura guarda el fotograma actual como PNG, sin overlays (`Ctrl+Mayús+S`). El icono de teclado lista todos los atajos.

El editor recuerda qué overlays dejaste activados entre sesiones; el modo en sí siempre empieza oculto.

35.1.119.3. Inspector, consola y animaciones

Con los overlays visibles, un clic en la vista previa selecciona un objeto y el **Inspector** (botón de la barra) muestra:

- su tipo con el vocabulario del script (`circle`, `rounded_rect`, `text`…) y el nombre que le diste con `named`;
- la línea del script que lo creó, con **Open** para ir a ella en tu editor de código y **Copy** para llevarte `archivo:línea`. Si lo creó una función tuya, debajo aparecen las líneas que la llamaron (**called from main.py · line 11**), cada una también se abre con un clic;
- su posición, rotación (en radianes, como `rotate_to`, y en grados), escala, tamaño y centro en unidades de la escena; su relleno y su trazo con el color en hexadecimal, y su opacidad;
- su `z_index`, su capa, su profundidad de parallax o si es HUD, si es visible, el texto que dibuja y cuántos hijos tiene.

Sin nada seleccionado, el inspector muestra la cámara de la escena: su posición, rotación, zoom (o campo de visión, recorte y objetivo en perspectiva), el encuadre y, si está activa, la profundidad de campo. Los valores son los del instante de la línea de tiempo, así que cambian al reproducir o recorrer la escena. Un clic sobre una letra selecciona el texto entero y, a igual `z_index`, gana la caja más pequeña bajo el cursor: la etiqueta sobre su tarjeta, el miembro dentro de su grupo. **Inside …** sube al grupo que lo contiene. Las líneas siguen siendo correctas tras cada recarga, aunque la recarga reutilice segmentos ya compilados.

La **Console** (`J`, o el botón de la barra) reúne lo que antes solo llegaba a la terminal: los avisos de Gaanim y de Python (`warnings.warn`), lo que escribe `print`, los errores del script con su traceback y las líneas de estado de cada recarga. Cada mensaje ocupa una fila y enlaza con la línea del script que lo produjo; un clic en la fila muestra debajo el mensaje completo y, si lo tiene, su traceback. Filtra por errores, avisos y mensajes, busca por texto o archivo, copia lo listado como en la terminal y limpia la lista. Por defecto muestra solo la última ejecución del script; **History** conserva las anteriores. Mientras está oculta, una insignia arriba a la izquierda cuenta los errores y avisos de la última ejecución y la abre con un clic.

Animations (`T`, o el botón de la barra) abre sobre la barra de reproducción una línea de tiempo de la escena: una fila por objeto animado, más la de la cámara, y un bloque por cada `play` que lo anima, coloreado por tipo (movimiento, transformación, aparición, estilo, trazo o cámara). Las animaciones de las letras de un texto aparecen en la fila del texto. Muestra el segmento bajo el cabezal; **Whole scene** muestra toda la escena, y la búsqueda filtra por objeto o por animación. Un clic o un arrastre

en la regla mueve el cabezal. Un clic en un bloque pausa, lleva el cabezal a su inicio y selecciona el objeto para verlo en el inspector; un doble clic abre la línea del `play`. Al pasar el cursor, el bloque muestra su duración, su curva, la línea del `play` y las líneas desde las que se llamó. Un clic en el nombre de la fila selecciona el objeto y un doble clic abre la línea que lo creó. Mientras está oculto, el panel no cuesta nada por fotograma.

El inspector aparece arriba a la derecha y el panel de animaciones sobre la barra de reproducción. Arrastra cualquiera de los dos por su cabecera (la fila con los puntos) para llevarlo a otro sitio de la ventana; se queda ahí al cambiar de selección y nunca sale de la pantalla. Un doble clic en la cabecera lo devuelve a su sitio. Mientras el panel de animaciones está en su sitio, la consola se coloca encima de él.

Open usa el comando de `GAANIM_EDITOR` (por ejemplo `code --goto {file}:{line}`); sin ella, el primero que encuentre en el `PATH` entre VS Code (`code`), Cursor, VSCodium, Zed y Sublime Text, y si no hay ninguno abre el archivo con la aplicación predeterminada, que no salta a la línea. El panel de error del script también ofrece **Open in the editor**.

35.1.120. Recorte y máscaras

Cualquier objeto vectorial puede recortar a otro con una máscara viva que se recalcula en cada fotograma: consulta `Drawable.clip` y `Drawable.no_clip`.

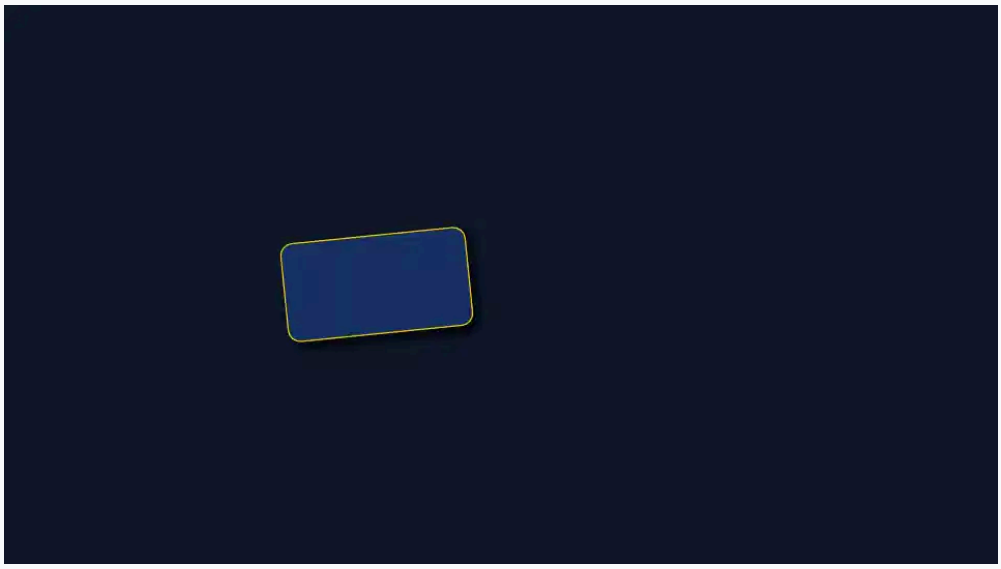
CAPÍTULO

36. Drawable

El handle fluido que devuelven todas las fábricas: estilo, posición, efectos, anclajes y relaciones reactivas

Cada fábrica de la escena (`scene.geometry`, `scene.text`, `scene.media`, `scene.viz`, `scene.slides`, `scene.mechanics`, `scene.layout`) devuelve un `Drawable`: un handle que encadena estilo, posición y efectos. Los tipos especializados (`Text`, `Image`, `Layout`, `Dimension`…) heredan todos estos métodos y conservan su tipo al encadenarlos.

```
from gaanim import BLUE, GOLD, Scene

scene = Scene(frame=(16, 9), background="#0f172a")
card = scene.geometry.rounded_rect(3, 1.6, 0.2).fill(BLUE).stroke(GOLD, 0.04)
card.move_to(-2, 0).rotate_to(0.1).shadow("#00000099")
scene.play([card.animate.move_to(2, 0).rotate_to(-0.1).duration(1.2)])
scene.render()
```



Hay dos formas de cambiar un objeto:

- Los **setters inmediatos** (`fill`, `move_to`, `opacity`…) fijan el estado inicial. Si los llamas después de un `play` o un `wait`, crean un corte reversible en el cursor de la línea de tiempo.
- El proxy `.animate` usa el mismo vocabulario, pero devuelve un `Anim` que interpola hacia el destino dentro de `scene.play(...)`. Consulta Animaciones para las entradas, los énfasis y el control del tiempo.

36.1.1. Estilo

Relleno, trazo, opacidad y orden de dibujo. Los colores aceptan `Color`, cadenas CSS o hex y `Brush` para degradados (ver Colores y temas).

36.1.2. Drawable.fill

MÉTODO `fill(paint: Paint) -> Self`

Pinta el interior de los contornos cerrados. En un grupo o un SVG importado, el color llega a todos los descendientes.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`paint`	`Paint`	obligatorio	`Color`, cadena CSS/hex, tupla RGB(A) o `Brush`.

Ejemplo:

```
disk = scene.geometry.circle(1).fill(BLUE)
badge = scene.geometry.square(1).fill("#f97316").move_to(3, 0)
```

36.1.3. Drawable.no_fill

MÉTODO `no_fill() -> Self`

Quita el relleno. Úsalo para contornos: `circle(1).no_fill().stroke(WHITE, 0.04)`.

36.1.4. Drawable.stroke

MÉTODO `stroke(paint: Paint, width: float, *, align: Optional[Literal["inside", "center", "outside"]] = None, scale_with_object: Optional[bool] = None) -> Self`

En contornos cerrados, incluidos los glifos de un texto, el trazo queda entero dentro por defecto, así `write` dibuja un ancho constante. `"center"` lo reparte a ambos lados y `"outside"` lo dibuja entero por fuera, por ejemplo como halo bajo una etiqueta que tapa líneas. Los caminos abiertos siempre centran su trazo. La alineación es estado de declaración y no se anima; otros valores lanzan `ValueError`. Los guiones conservan su longitud con cualquier alineación, y los remates, uniones y guiones fijados antes con `stroke_style` se conservan: `stroke` solo cambia la pintura y el ancho.

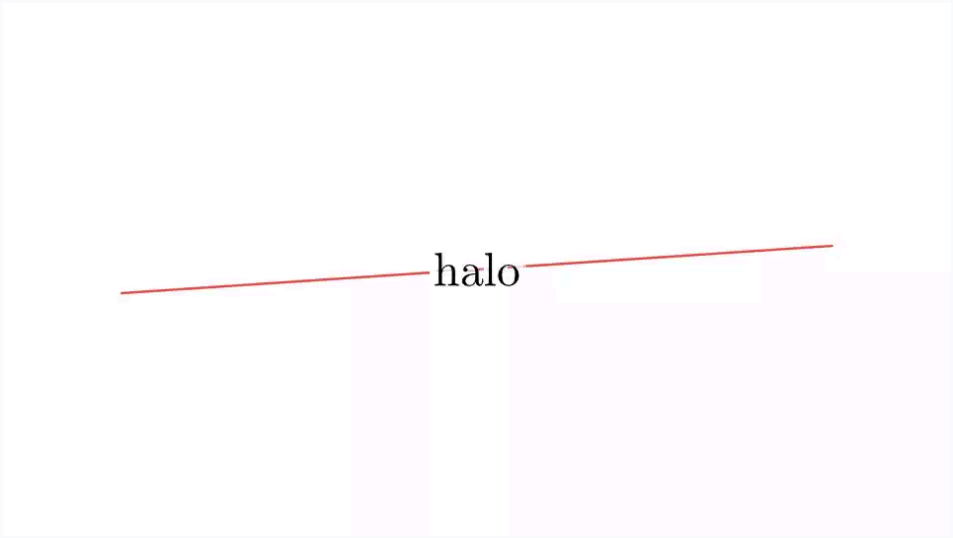
Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`paint`	`Paint`	obligatorio	`Color` o `Brush`.
`width`	`float`	obligatorio	Ancho en unidades lógicas de escena. No sigue la escala de la figura: `scale_to`, `scale_to_3d`, `matrix_to`, las inclinaciones, la escala de sus grupos y sus animaciones cambian la forma, no el pincel, que queda redondo e igual de ancho en todos los lados. También en SVG escalados.

`align`	`str | None`	`None`	`"inside"`, `"center"` u `"outside"` respecto a los contornos cerrados; `None` deja el trazo dentro. En los tres casos el trazo visible mide `width`.
`scale_with_object`	`bool | None`	`None`	`True` hace que el ancho siga la escala de la figura, como `scale_stroke_with_object`.

Ejemplo:

```
from gaanim import Anchor, BLACK, RED, WHITE, Scene
scene = Scene(frame=(16, 9), background=WHITE)
scene.geometry.line(-6, -0.4, 6, 0.4).stroke(RED, 0.04)
scene.text("halo", size=0.8).fill(WHITE).stroke(WHITE, 0.15, align="outside").move_to(0, 0,
Anchor.CENTER)
scene.text("halo", size=0.8).fill(BLACK).move_to(0, 0, Anchor.CENTER)
scene.wait(0.1)
scene.render()
```



36.1.5. Drawable.scale_stroke_with_object

MÉTODO `scale_stroke_with_object(enabled: bool = True) -> Self`

Hace que el ancho del trazo siga la escala que acumula la figura (la suya, la de sus grupos y la de un espacio de coordenadas), como un dibujo que se amplía: al escalar una cara ×1.18, las cejas y la nariz engrosan con ella. Una escala no uniforme ensancha el pincel en su eje. En un grupo o un SVG importado se aplica a todos sus trazos. Es estado de declaración y no se anima.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`enabled`	`bool`	`True`	`False` vuelve al ancho en unidades de escena.

Ejemplo:

```
face = scene.geometry.group([head, brow]).scale_stroke_with_object()
face.scale_to(1.18)
```

36.1.6. Drawable.chalk

MÉTODO `chalk(seed: int = 0, roughness: float = 0.01) -> Self`

Dibuja el relleno y el trazo como tiza en una pizarra: el contorno tiembla un poco y un grano rompe la pintura. En un texto o un grupo se aplica a cada glifo y miembro. Vale para toda la escena, aunque se llame después de un `play`; no se anima. `write(brush="chalk")` hace lo mismo desde la animación. Los paquetes `.gaanim` la graban y reproducen igual.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`seed`	`int`	`0`	Semilla del temblor y del grano: la misma semilla dibuja la misma tiza en cada fotograma y exportación.
`roughness`	`float`	`0.01`	Desplazamiento máximo del contorno, en unidades de escena; `0` deja el contorno limpio y solo aplica el grano.

Ejemplo:

```
box = scene.geometry.rect(3, 1.5).no_fill().stroke(WHITE, 0.05).chalk(seed=2)
scene.play([box.animate.write().duration(1)])
```

36.1.7. Drawable.no_stroke

MÉTODO `no_stroke() -> Self`

Quita el trazo. Las flechas sólidas se ven mejor con `.fill(color).no_stroke()`.

36.1.8. Drawable.stroke_style

MÉTODO `stroke_style(style: StrokeStyle) -> Self`

Aplica la geometría completa del trazo; el `align` del estilo, si lo tiene, coloca el trazo como `stroke(align=...)`. En una llamada directa, la pintura debe ser un color CSS literal o un `Brush`; los nombres de token se resuelven cuando el `StrokeStyle` va dentro de `Theme.styles`. Métricas inválidas lanzan `ValueError`.

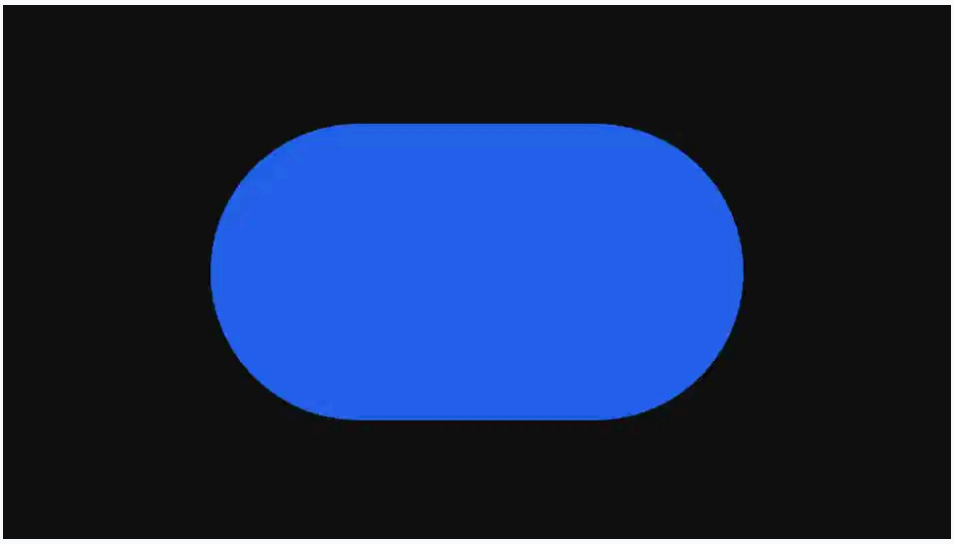
Parámetros:

Nombre	Tipo	Predeterminado	Descripción

`style`	`StrokeStyle`	obligatorio	Pintura, ancho, extremo, unión, límite de inglete, guiones, desfase de guiones y alineación.

Ejemplo:

```
from gaanim import Scene, StrokeStyle
scene = Scene(frame=(16, 9))
guide = scene.geometry.line(-2, 0, 2, 0).stroke_style(
    StrokeStyle("#2563eb", 5, cap="round", dashes=[18, 10])
)
scene.wait(0.1)
scene.render()
```



36.1.9. Drawable.style_class

MÉTODO `style_class(name: str) -> Self`

Añade una clase ordenada que usa `Theme.styles`. Varias llamadas se aplican en cascada, en orden; los estilos del constructor y los setters fluidos siguen teniendo prioridad. En un grupo, la clase llega a todos los miembros visuales.

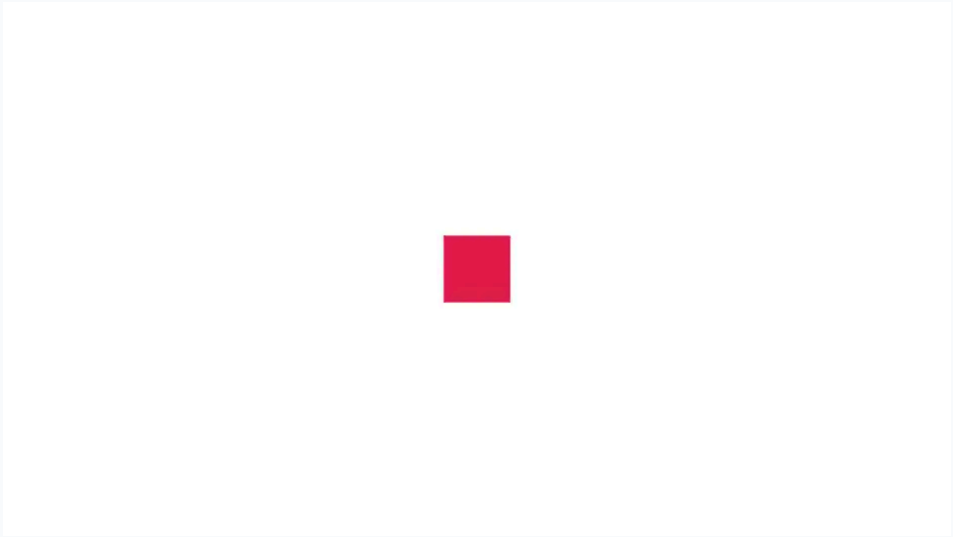
Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`name`	`str`	obligatorio	Clase del tema sin el punto inicial; admite letras, dígitos, _ y -.

Ejemplo:

```
from gaanim import Scene, Style, Theme
theme = Theme("paper", styles={".warning": Style(fill="#e11d48")})
scene = Scene(frame=(16, 9), theme=theme)
warning = scene.geometry.square(1.125).style_class("warning")
```

```
scene.wait(0.1)
scene.render()
```



36.1.10. Drawable.opacity

MÉTODO `opacity(op: ScalarSource) -> Self`

La opacidad se multiplica por la jerarquía: la de un grupo escala la de sus miembros sin cambiar sus valores propios. Un grupo o un SVG translúcido se compone como una sola capa, como `opacity` en CSS: al fundirlo, lo que una pieza tapa sigue tapado (los ojos no asoman bajo los párpados) y las piezas translúcidas no se marcan más. Con una fuente reactiva el canal queda vinculado desde el cursor; un número termina el vínculo con un corte reversible. Mientras esté vinculado, anima la fuente: animar el canal directamente lanza un error.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
op	`float | Parameter | Variable | Computed | TimeInput`	obligatorio	Opacidad de 0 a 1, o una fuente reactiva.

Ejemplo:

```
ghost = scene.geometry.circle(1).fill(WHITE).opacity(0.3)
level = scene.viz.parameter(0.2)
halo = scene.geometry.circle(1.4).fill(GOLD).opacity(level)
scene.play([level.animate.set(1.0).duration(1)])
```

36.1.11. Drawable.z_index

MÉTODO `z_index(z: int) -> Self`

El `z_index` de un grupo o de un texto se suma al de cada descendiente, así que mueve todo el subárbol. Los empates conservan el orden de creación. Con capas (`Scene.z_layers`), ordena solo dentro de la capa del objeto.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`z`	`int`	obligatorio	Capa de dibujo; los valores mayores quedan encima.

Ejemplo:

```
front = scene.geometry.circle(1).fill(GOLD).z_index(5)
back = scene.geometry.rect(3, 1).fill(BLUE)
```

36.1.12. Drawable.z_layer

MÉTODO `z_layer(name: str) -> Self`

Dibuja el objeto en la capa `name`: queda encima de todo lo que está en las capas de detrás, sea cual sea su `z_index`, que solo ordena dentro de la capa. Un grupo o un texto se lleva a sus miembros, salvo a los que nombran su propia capa. El orden de dibujo vale para toda la escena, así que una llamada después de mostrar el objeto también rige desde el principio. Un nombre que la escena no declaró lanza `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`name`	`str`	obligatorio	Una de las capas nombradas con `Scene.z_layers`.

Ejemplo:

```
scene.z_layers("modelo", "overlay")
muro = scene.geometry.rect(4, 3).fill(BLUE).z_index(400)
lupa = scene.geometry.circle(0.8).fill(GOLD).z_layer("overlay")
```

36.1.13. Drawable.covers_on_purpose

MÉTODO `covers_on_purpose() -> Self`

Marca que el objeto, y sus miembros si es un grupo, tapa textos a propósito, como un diálogo dibujado sobre un modelo mientras se explica: el aviso de textos tapados de `gaanim check` deja de contar los textos que tapa. No cambia nada del dibujo y vale para toda la escena.

Ejemplo:

```
dialogo = scene.geometry.rect(4, 2).fill(WHITE).covers_on_purpose()
```

36.1.14. Drawable.named

MÉTODO `named(name: str) -> Self`

Da un nombre al objeto y lo devuelve; `Drawable.name` lo lee (`None` si no tiene). El nombre identifica al mismo objeto en dos estados: `magic_move` y `Transition.magic_move` emparejan los objetos con el mismo nombre. Un nombre en blanco lanza `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`name`	`str`	obligatorio	Nombre no vacío; no hace falta que sea único.

Ejemplo:

```
bar = scene.geometry.rect(3, 0.5).fill(BLUE).named("ana")
print(bar.name)

ana
```

36.1.15. Posición y transformaciones

Setters absolutos (`move_to`, `scale_to`, `rotate_to`, `skew_to`) y relativos (`shift_by`, `scale_by`, `rotate_by`) en unidades de escena y radianes. Sus versiones animadas están en Animaciones.

36.1.16. Drawable.move_to

MÉTODO `move_to(x, y, anchor=None) | move_to(reference) | move_to(point) -> Self`

Sin `anchor`, los objetos se colocan por su centro visual, medido con la geometría tal como se declaró: mover después una parte de un SVG (`svg.part("g").shift_by(...)`) no mueve el SVG entero; las raíces de sistemas de coordenadas colocan su origen matemático, para que las etiquetas no desplacen los ejes. `move_to(reference)` crea una relación de layout diferida centro con centro que no sigue animaciones posteriores: usa `follow` o `attach_to` para eso. Una referencia o un `AnchorPoint` no se combinan con `y` ni con `anchor`. `Text` acepta además `TextAnchor` (ver Texto). Ojo: sin `anchor`, un texto de una línea se coloca por su línea base (`TextAnchor.BASELINE_CENTER`), no por su centro visual, así que un título con `move_to(0, y)` queda medio cuerpo más arriba; usa `move_to(0, y, Anchor.CENTER)` para centrarlo. Las coordenadas aceptan fuentes reactivas, con las mismas reglas que `opacity`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`x / y`	`float | fuente reactiva`	obligatorio	Posición de destino en unidades de escena.

`anchor`	`Anchor | str | None`	`None`	Un `Anchor` o su nombre, como `"top_left"`. Punto propio que se coloca en `(x, y)`; posicional o por nombre.
`reference`	`Drawable`	obligatorio	Alternativa: centra este objeto sobre otro.
`point`	`AnchorPoint`	obligatorio	Alternativa: coloca el centro sobre un anclaje de otro objeto.

Ejemplo:

```
box = scene.geometry.rect(3, 1.5).move_to(-4, 1, Anchor.TOP_LEFT)
label = scene.text("centro").move_to(box)
corner = scene.geometry.dot(0.1).move_to(box.anchor_point(Anchor.BOTTOM_RIGHT))
```

36.1.17. Drawable.shift_by

MÉTODO `shift_by(dx: float, dy: float) -> Self`

Traslada el objeto `(dx, dy)` unidades de escena respecto a su posición actual, como un corte inmediato en el cursor.

Ejemplo:

```
dot = scene.geometry.dot(0.1).move_to(0, 0)
scene.wait(0.5)
dot.shift_by(1, 0.5)
```

36.1.18. Drawable.scale_to

MÉTODO `scale_to(factor: ScalarSource) -> Self`

Fija la escala uniforme absoluta alrededor del pivote (ver `with_pivot`). Acepta fuentes reactivas.

Ejemplo:

```
logo = scene.media.svg("assets/logo.svg").scale_to(2.5)
```

36.1.19. Drawable.scale_by

MÉTODO `scale_by(factor: float) -> Self`

Multiplica la escala actual por `factor` en el cursor. Mayor que 1 agranda y menor que 1 reduce.

36.1.20. Drawable.rotate_to

MÉTODO `rotate_to(radians: ScalarSource) -> Self`

Fija la rotación absoluta en radianes alrededor del pivote (ver `with_pivot`); los valores positivos giran en sentido antihorario. Acepta fuentes reactivas.

Ejemplo:

```
tilted = scene.geometry.square(1).rotate_to(math.pi / 4)
```

36.1.21. Drawable.rotate_by

MÉTODO `rotate_by(radians: float) -> Self`

Suma `radians` a la rotación actual en el cursor.

36.1.22. Drawable.skew_to

MÉTODO `skew_to(x: float, y: float) -> Self`

Fija el sesgo (*skew*) absoluto alrededor del pivote; `(0, 0)` lo quita. Los factores son tangentes sin unidades: con el pivote en la base, `skew_to(0.15, 0)` desplaza la parte superior 0.15 unidades por cada unidad de altura. En un grupo deforma a todos los hijos a la vez, sin trocear la figura. `animate.skew_to(x, y)` lo anima. Lanza `ValueError` con factores no finitos.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`x`	`float`	obligatorio	Corte horizontal: cada punto se desplaza en x `x` veces su altura sobre el pivote.
`y`	`float`	obligatorio	Corte vertical: cada punto se desplaza en y `y` veces su distancia horizontal al pivote.

Ejemplo:

```
from gaanim import BLUE, GOLD, Easing, Scene
scene = Scene(frame=(16, 9), background="#0f172a")
ground = scene.geometry.rect(8, 0.1).fill(GOLD).move_to(0, -2.05)
house = scene.geometry.group([
    scene.geometry.rect(3, 2.5).fill(BLUE).move_to(0, -0.75),
    scene.geometry.polygon([(-1.8, 0.5), (1.8, 0.5), (0, 2)]).fill(BLUE),
]).with_pivot(0, -2)
scene.play(house.animate.skew_to(0.2, 0).duration(0.4).easing(Easing.SMOOTH))
scene.play(house.animate.skew_to(-0.12, 0).duration(0.4).easing(Easing.SMOOTH))
scene.play(house.animate.skew_to(0, 0).duration(0.5).easing(Easing.SMOOTH))
scene.render()
```

36.1.23. Drawable.matrix_to

MÉTODO `matrix_to(matrix: tuple[tuple[float, float], tuple[float, float]]) -> Self`

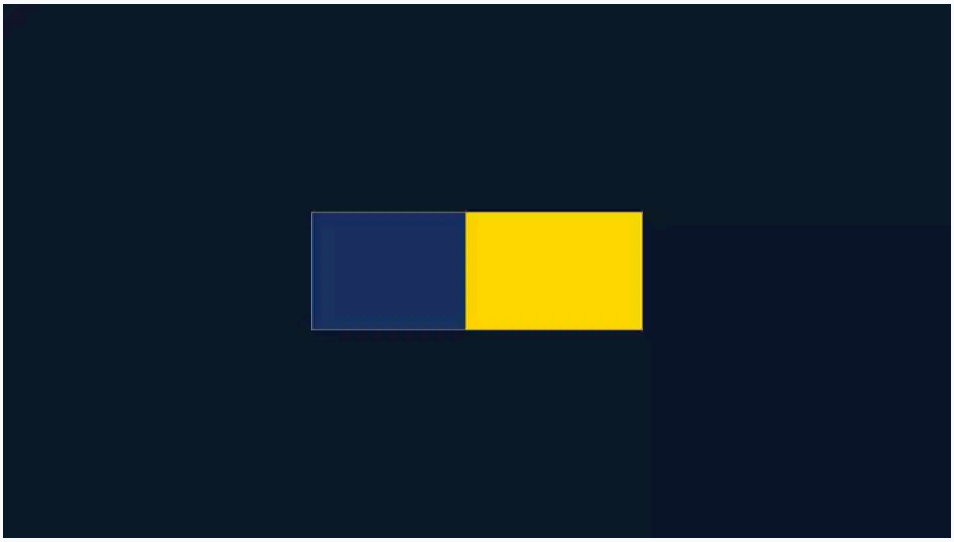
Aplica una transformación lineal 2D cualquiera alrededor del pivote y sustituye el giro, el sesgo y la escala actuales. A diferencia de `skew_to`, lleva los ejes x e y a cualquier dirección, así que una cara dibujada de frente pasa a una proyección isométrica u oblicua sin trocearla. `animate.matrix_to(...)` la anima: se descompone en giro, sesgo y escala, que se interpolan juntos. Lanza `ValueError` con una matriz no finita o no invertible.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`matrix`	`tuple[tuple[float, float], tuple[float, float]]`	obligatorio	Filas `((a, b), (c, d))` de la aplicación `(x, y) → (a x + b y, c x + d y)`.

Ejemplo:

```python
import math
from gaanim import BLUE, GOLD, GRAY, Scene
scene = Scene(frame=(16, 9), background="#0f172a")
c, s = math.cos(math.pi / 6), math.sin(math.pi / 6)
# Las caras giran sobre su centro: el eje x baja o sube 30°, el y sigue vertical.
side = scene.geometry.rect(3, 2).fill(BLUE).stroke(GRAY, 0.03).move_to(-1.3, 0)
front = scene.geometry.rect(3, 2).fill(GOLD).stroke(GRAY, 0.03).move_to(1.3, 0)
scene.play([
    side.animate.matrix_to(((c, 0), (-s, 1))).duration(0.8),
    front.animate.matrix_to(((c, 0), (s, 1))).duration(0.8),
])
scene.render()
```



36.1.24. Drawable.bounds

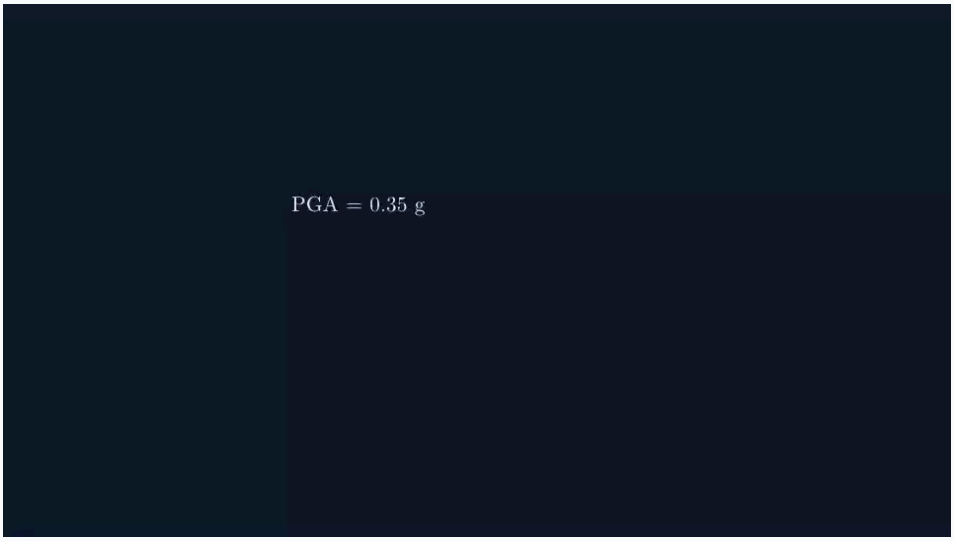
MÉTODO `bounds() -> Bounds`

Mide la caja de cualquier objeto (figuras, texto, matemática, SVG, imágenes, grupos) en el cursor actual, tal como se dibujaría ahí: cuenta la maquetación, las transformaciones, la composición del texto, las animaciones que terminaron hasta `scene.cursor` y los cambios inmediatos hechos en él (como un `move_to` sobre un objeto ya animado), e incluye a los descendientes. La geometría que los objetos reactivos regeneran en cada fotograma se mide como se declaró. Medir un objeto recién creado, antes del siguiente `play` o `wait` y sin nada que actúe sobre él (animaciones, grupos posteriores, layouts), cuesta alrededor de un milisegundo: se compila solo su declaración y la de sus miembros o referencias de `next_to`/`align_to` que cumplan lo mismo. En cualquier otro caso se compila la escena escrita hasta ese punto. Las mediciones sin nada escrito entre ellas comparten una compilación, y también los miembros de un mismo layout recién creado, así que conviene medir antes de animar y medir juntas las cajas que necesites. Una medición posterior reanuda esa compilación desde el segmento donde cambió la escena, como una recarga del editor, y la siguiente ejecución del script reanuda la última; un segmento con funciones de Python (reactivas, updaters) no se reutiliza entre ejecuciones. Lanza `ValueError` si el objeto no tiene geometría en el cursor. Sustituye a `scene.text.measure`.

Devuelve: `Bounds` — `x` e `y` (el centro), `left`, `right`, `bottom`, `top`, `width`, `height` y `center` en unidades de escena.

Ejemplo:

```
from gaanim import GOLD, Scene
scene = Scene(frame=(16, 9), background="#0f172a")
label = scene.text("PGA = 0.35 g", role="label").move_to(-2, 1)
box = label.bounds()
frame = scene.geometry.rounded_rect(box.width + 0.56, box.height + 0.32, 0.14)
frame.no_fill().stroke(GOLD, 0.04).move_to(*box.center)
scene.play([frame.animate.create().duration(0.6)])
scene.render()
```



36.1.25. Drawable.pixel

`MÉTODO` `pixel(px: float, py: float) -> AnchorPoint`

Convierte un píxel de una imagen o un vídeo en un punto de anclaje que sirve en `move_to`, `scene.camera.inset`, `pan_to` o conectores, y que sigue a la imagen si se mueve, escala o gira. Tiene en cuenta el recorte, `width=` y el modo de ajuste; mide sobre el archivo original, no sobre una captura reducida. Lanza `ValueError` en otros objetos o con coordenadas no finitas.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`px, py`	`float`	obligatorio	Píxel del archivo original, contado desde la esquina superior izquierda con y hacia abajo, como en un editor de imágenes.

Devuelve: `AnchorPoint` — El punto de la imagen o el vídeo en ese píxel.

Ejemplo:

```
ui = scene.media.image("assets/cover.png", width=12)
view = scene.camera.inset(ui.pixel(812, 240), zoom=3)
```

36.1.26. Drawable.with_pivot

MÉTODO `with_pivot(x: float, y: float) -> Self`

Fija el pivote de rotación, escala y sesgo en coordenadas de escena. Úsalo para bisagras y brazos que giran alrededor de un extremo. Sin él, el pivote es el centro de la caja en las figuras declaradas con coordenadas de escena (`line`, `polygon`, arcos, flechas y curvas) y el origen propio en las demás, que en `circle`, `rect` o `text` también es su centro. Una vez mostrado el objeto, el pivote cambia en el cursor sin moverlo y vale para las animaciones siguientes. En una parte de un SVG el punto también está en coordenadas de escena, aunque el SVG esté escalado o movido. `pivot(x, y)` es un alias.

Ejemplo:

```
from gaanim import BLUE, GOLD, Scene
from math import pi
scene = Scene(frame=(16, 9), background="#0f172a")
hinge = scene.geometry.dot(0.09).fill(GOLD).move_to(-2.5, 1.25)
arm = scene.geometry.rect(1.125, 0.225).fill(BLUE).move_to(hinge).with_pivot(-2.5, 1.25)
scene.play([arm.animate.rotate_by(pi/2.5).duration(1.0)])
scene.render()
```

36.1.27. Drawable.pivot

MÉTODO `pivot(x: float, y: float) -> Self`

Alias de `with_pivot`.

36.1.28. Drawable.copy

MÉTODO `copy() -> Drawable`

Copia el objeto tal como está declarado: forma, estilo, efectos y posición, con copias de sus miembros y, en un SVG importado, de sus partes con nombre (`copia.part("cabeza")` es la cabeza de la copia). No copia animaciones ni updaters, y los cambios posteriores de uno no

llegan al otro. La copia se dibuja encima de lo declarado antes. Sirve para fantasmas o reflejos de un personaje sin importar el SVG varias veces.

Devuelve: `Drawable` — Un objeto nuevo e independiente; la copia de un `Text` es un `Text`.

Ejemplo:

```
robot = scene.media.svg("assets/robot.svg").scale_to(2)
ghost = robot.copy().opacity(0.3).shift_by(-1.5, 0)
ghost.part("head").fill(GOLD)
```

36.1.29. Posición relativa

Coloca un objeto libre respecto a otro o a los bordes del área segura. Los hijos de un `Layout` no usan estos métodos: su posición la decide el contenedor (ver Layout).

36.1.30. Drawable.next_to

MÉTODO `next_to(reference: Drawable, direction: Direction, spacing: float = 0.24, aligned_edge: Optional[AnchorName | Anchor] = None) -> Self`

Coloca este objeto junto a otro sin superponerlos.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`reference`	`Drawable`	obligatorio	Objeto de referencia.
`direction`	`Direction`	obligatorio	Lado donde se coloca este objeto.
`spacing`	`float`	`0.24`	Separación entre bordes en unidades de escena.
`aligned_edge`	`Anchor | str | None`	`None`	Borde común que se alinea, como `Anchor` o por su nombre; `None` centra en el eje transversal.

Ejemplo:

```
title = scene.text("Resultados", role="title")
rule = scene.geometry.line(length=4).next_to(title, Direction.DOWN, spacing=0.2)
note = scene.text("n = 42", role="caption").next_to(title, Direction.RIGHT, spacing=0.3,
aligned_edge=Anchor.BOTTOM)
```

36.1.31. Drawable.align_to

MÉTODO `align_to(reference: Drawable, target_anchor: AnchorName | Anchor, reference_anchor: Optional[AnchorName | Anchor] = None) -> Self`

Hace coincidir un anclaje propio con un anclaje de otro objeto.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`reference`	`Drawable`	obligatorio	Objeto de referencia.
`target_anchor`	`Anchor | str`	obligatorio	Anclaje propio que se alinea.
`reference_anchor`	`Anchor | str | None`	`None`	Anclaje de la referencia; `None` usa el mismo que `target_anchor`.

Ejemplo:

```
panel = scene.geometry.rect(5, 3).move_to(1, 0)
tag = scene.slides.badge("NUEVO").align_to(panel, Anchor.TOP_LEFT)
```

36.1.32. Drawable.to_edge

MÉTODO `to_edge(direction: Direction, buff: float = 0.24) -> Self`

Lleva el objeto al borde indicado del área segura, separado `buff` unidades.

Ejemplo:

```
title = scene.text("Capítulo 2", role="title").to_edge(Direction.UP, buff=0.4)
```

36.1.33. Drawable.to_corner

MÉTODO `to_corner(corner: AnchorName | Anchor, buff: float = 0.24) -> Self`

Lleva el objeto a una esquina del área segura (`Anchor.TOP_LEFT`, `Anchor.BOTTOM_RIGHT`… o su nombre, como `"bottom_right"`), separado `buff` unidades.

Ejemplo:

```
page = scene.text("3 / 12", role="caption").to_corner(Anchor.BOTTOM_RIGHT, buff=0.3)
```

36.1.34. Anclajes y puertos

Puntos no dibujados que siguen al objeto en cada fotograma. Sirven como extremos de líneas, conectores, cotas y seguidores (ver Geometría).

36.1.35. Drawable.anchor_point

MÉTODO `anchor_point(anchor: Optional[AnchorName | Anchor] = None, *, offset: tuple[float, float] = (0.0, 0.0)) -> AnchorPoint`

El desplazamiento local rota y escala con el objeto y con sus padres. Valores no finitos lanzan `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
anchor	Anchor | str | None	None	Uno de los nueve anclajes de los límites locales, o su nombre ("top_left"); None equivale a Anchor.CENTER.
offset	tuple[float, float]	(0.0, 0.0)	Desplazamiento adicional en el espacio local.

Ejemplo:

```
frame = scene.geometry.rect(2.25, 1.125)
corner = frame.anchor_point(Anchor.TOP_RIGHT, offset=(0.1, 0))
link = scene.geometry.line((-4, 2), corner)
```

36.1.36. Drawable.with_port

MÉTODO `with_port(name: str, anchor: AnchorName | Anchor, *, offset: tuple[float, float] = (0.0, 0.0)) -> Self`

Define un anclaje local con nombre y devuelve el mismo objeto. El puerto sigue los límites, el reflow y las transformaciones de los padres. Los nombres son únicos e inmutables: vacíos, repetidos u offsets no finitos lanzan `ValueError`.

Ejemplo:

```
node = scene.geometry.rect(2, 1).with_port("out", Anchor.RIGHT, offset=(0.1, 0))
```

36.1.37. Drawable.port

MÉTODO `port(name: str) -> AnchorPoint`

Devuelve el `AnchorPoint` reactivo de un puerto definido con `with_port`. Un nombre desconocido lanza `KeyError`.

Ejemplo:

```
target = scene.geometry.circle(0.5).move_to(4, 1)
edge = scene.geometry.connector(node.port("out"), target)
```

36.1.38. Drawable.at_coordinate

MÉTODO `at_coordinate(coordinate: CoordinateRef) -> Drawable`

Coloca el objeto en una coordenada simbólica de un espacio de coordenadas (`plane.coord(x, y)`). La posición sigue al espacio y a sus cambios de vista.

Ejemplo:

```
plane = scene.viz.cartesian_2d(Axis.linear(-4, 4), Axis.linear(-2, 2))
peak = scene.geometry.dot(0.1).fill(GOLD).at_coordinate(plane.coord(1, 1.5))
```

Cada objeto expone también las expresiones lineales `left`, `right`, `top`, `bottom`, `center_x`, `center_y`, `width` y `height` para las restricciones de Layout.

36.1.39. Efectos y recortes

Resplandor, desenfoque, sombra, recorte por máscara y recorte del trazo. Los efectos se aplican a rellenos y trazos, incluidos los degradados.

36.1.40. Drawable.glow

MÉTODO `glow(color: ColorLike, radius: float = 0.16, intensity: float = 1.0) -> Drawable`

Añade un resplandor alrededor del objeto.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`color`	`Color`	obligatorio	Color del resplandor.
`radius`	`float`	`0.16`	Radio en unidades de escena.
`intensity`	`float`	`1.0`	Intensidad relativa.

Ejemplo:

```
from gaanim import BLUE, GOLD, Scene
scene = Scene(frame=(16, 9), background="#0f172a")
obj = scene.geometry.circle(0.56).fill(BLUE).stroke(GOLD, 0.04).move_to(0, 0)
obj.glow(GOLD, radius=0.225)
scene.play([obj.animate.grow_from_center().duration(0.8)])
scene.render()
```

36.1.41. Drawable.blur

MÉTODO `blur(sigma: float = 0.04) -> Drawable`

Desenfoque gaussiano de `sigma` unidades de escena; `blur(0)` lo quita. Después del primer `play` o `wait` es un corte en el cursor, como `fill`. Un `sigma` negativo o no finito lanza `ValueError`.

Ejemplo:

```
blob = scene.geometry.circle(2).fill("#1E3A8A").blur(0.15)
```

36.1.42. Drawable.shader_effect

MÉTODO `shader_effect(effect: Optional[PostProcess | Sequence[PostProcess]], *, margin: float = 0.25) -> Self`

Dibuja el objeto y sus descendientes en una textura propia, le aplica los pases y devuelve el resultado a la escena en su lugar del orden de dibujo. Es un `PostProcess.shader` sobre un solo objeto: en el shader, `gaanim_scene(uv)` lee solo ese objeto (transparente alrededor), `uv` va de 0 a 1 sobre su caja con el margen, `resolution` es el tamaño de la textura en píxeles y `time` es el tiempo de la escena. Los uniforms aceptan `Parameter`, `Computed` y señales de audio. Los presets de `PostProcess` (`grain`, `chromatic_aberration`…) también sirven. Es estado de declaración: vale para toda la línea de tiempo. Si un objeto con efecto contiene otro, se aplica el efecto exterior. Se ve en la vista previa, en la exportación y en los paquetes `.gaanim`, que guardan los uniforms de cada fotograma.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`effect`	`PostProcess | Sequence[PostProcess] | None`	obligatorio	Un pase o una cadena de pases WGSL; `None` quita el efecto.
`margin`	`float`	`0.25`	Unidades de escena que se agregan alrededor del objeto, para efectos que se salen de su contorno

			(un brillo, una onda).

Ejemplo:

```
from gaanim import GOLD, WHITE, PostProcess, Scene
scene = Scene(frame=(16, 9), background="#0b1020")
ripple = PostProcess.shader("""
fn gaanim_post(uv: vec2<f32>, resolution: vec2<f32>, time: f32) -> vec4<f32> {
    let wave = gaanim_uniforms.amount * sin(uv.x * 14.0 + time * 6.0);
    return gaanim_scene(uv + vec2<f32>(0.0, wave));
}
""", uniforms={"amount": 0.12})
title = scene.text("Ondas").fill(WHITE).scale_to(2.0).shader_effect(ripple, margin=0.4)
plain = scene.text("Sin efecto").fill(GOLD).scale_to(1.0).move_to(0, -2.5)
scene.wait(2)
scene.render()
```

36.1.43. Drawable.matte

MÉTODO `matte(source: Optional[Drawable], mode: Literal["alpha", "alpha_inverted", "luma", "luma_inverted"] = "alpha") -> Self`

Muestra el objeto y sus descendientes solo a través de `source`, como un *track matte* de After Effects. Con `"alpha"` se ve donde `source` es opaco (una foto o un degradado dentro de las letras de un título); con `"luma"`, donde `source` es claro (un rectángulo con degradado que avanza hace un revelado suave); los modos invertidos muestran lo contrario. El mate sigue a `source` cuando se mueve, escala, aparece o se escribe, pero `source` deja de dibujarse por sí mismo. Un `source` desenfocado (`blur`, `blur_in`) suaviza el mate, no el objeto: los bordes de sus partes siguen nítidos dentro de la zona suave, así que unas barras enmascaradas por un texto que entra con `blur_in` se ven enteras donde un glifo aún es una mancha. Para que el resultado entre suave, funde `source` con `animate.opacity` o desenfoca el objeto. Es estado de declaración, vale para toda la línea de tiempo y se ve en la vista previa, la exportación y los paquetes `.gaanim`. Un `source` de otra escena, el propio objeto o un modo desconocido lanzan `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`source`	`Drawable | None`	obligatorio	El mate: el objeto cuya forma o brillo deja ver a este; `None` quita el mate.
`mode`	`str`	`"alpha"`	`"alpha"`, `"alpha_inverted"`, `"luma"` o `"luma_inverted"`.

Ejemplo:

```
from gaanim import CORAL, GOLD, WHITE, Scene
scene = Scene(frame=(16, 9), background="#0b1020")
title = scene.text("MATE").fill(WHITE).scale_to(3.0)
stripes = scene.geometry.group([
    scene.geometry.rect(0.5, 3).fill(color).no_stroke().move_to(-3.5 + 0.5 * i, 0)
    for i, color in zip(range(15), [GOLD, CORAL, "#22d3ee"] * 5)
])
stripes.matte(title)
scene.play([title.animate.write().duration(1.5)])
scene.render()
```

36.1.44. Drawable.glass

MÉTODO `glass(blur: float = 0.25, *, saturation: float = 1.4, refraction: float = 0.08, edge: float = 0.3, dispersion: float = 0.0, bevel: float = 0.12, transparency: float = 1.0, twist: float = 0.0) -> Self`

Convierte el objeto en vidrio. Dentro de su contorno se ve el lienzo y todo lo que se dibujó antes, desenfocado y saturado. El contorno es una lente con un borde redondeado de ancho `bevel`: ahí lo que hay detrás se curva, se separa en colores y recibe un brillo. El borde sigue el contorno real de cualquier forma, metaballs incluidos. El objeto se dibuja encima, así que un relleno translúcido tiñe el vidrio y un trazo lo enmarca; el vidrio aparece y se desvanece con él. `liquid_glass()` es el mismo vidrio con valores de *Liquid Glass* (`refraction=1.0`, `dispersion=0.12`, `bevel=0.6`, `blur=0.04`, `edge=0.8`, `saturation=1.4`, `transparency=0.92`, `twist=0.6`): casi sin desenfoque, con el centro plano, un borde ancho que se empina hacia el contorno, curva con fuerza lo que hay detrás y lo desliza a lo largo del contorno, y una línea fina

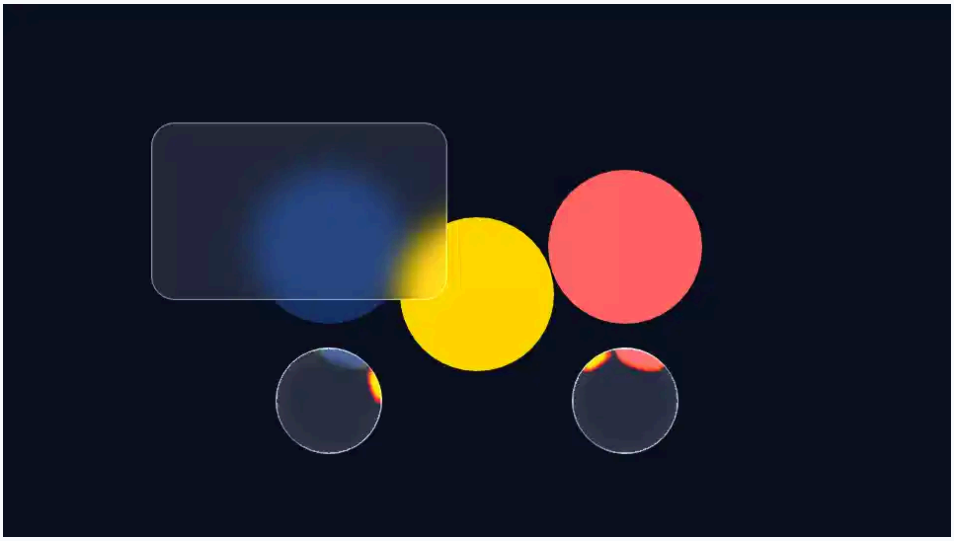
de luz en el contorno. Con un relleno claro (`"#ffffff26"`) se ve como en iOS. `backdrop_blur(radius, saturation=1.0)` es solo el desenfoque, y `no_glass()` quita el efecto. Es estado de declaración y se ve en la vista previa, la exportación y los paquetes `.gaanim`. Valores negativos o no finitos, o `edge`, `dispersion` o `transparency` mayores que 1, lanzan `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`blur`	`float`	`0.25`	Desenfoque de lo que hay detrás, en unidades de escena (la sigma de una gaussiana).
`saturation`	`float`	`1.4`	Saturación de esos colores; 1 los deja igual.
`refraction`	`float`	`0.08`	Cuánto se curva hacia dentro, en el borde, lo que hay justo fuera, en unidades de escena.
`edge`	`float`	`0.3`	Brillo del borde, con la luz arriba a la izquierda, de 0 a 1.
`dispersion`	`float`	`0.0`	Aberración cromática: cuánto se separan el rojo, el verde y el azul en el borde, de 0 a 1.
`bevel`	`float`	`0.12`	Ancho del borde redondeado de la lente, en unidades de escena; hacia dentro el vidrio es plano y no deforma.
`twist`	`float`	`0.0`	Desliza lo que muestra el borde a lo largo del contorno, como fracción de la curvatura, de −2 a 2: positivo gira en sentido horario (hacia la derecha arriba y hacia la izquierda abajo), como en *Liquid Glass*.
`transparency`	`float`	`1.0`	Cuán claro es el vidrio, de 0 a 1: 1 deja ver lo que hay detrás, valores menores lo vuelven lechoso y 0 es blanco opaco.

Ejemplo:

```
from gaanim import BLUE, CORAL, GOLD, Scene
scene = Scene(frame=(16, 9), background="#0b1020")
for i, color in enumerate([BLUE, GOLD, CORAL]):
    scene.geometry.circle(1.3).fill(color).no_stroke().move_to(-2.5 + 2.5 * i, 0.4 * (-1) ** i)
card = scene.geometry.rounded_rect(5, 3, 0.4).fill("#ffffff1a").stroke("#ffffff55",
0.03).move_to(-3, 1)
card.glass(blur=0.25, refraction=0.14)
drops = [scene.geometry.circle(0.9).move_to(x, -2.2) for x in (-2.5, 2.5)]
lens = scene.geometry.metaballs(drops, smoothness=0.9).fill("#ffffff10").liquid_glass()
scene.play([card.animate.move_to(3, 1).duration(2.0), drops[0].animate.move_to(1.0,
-2.2).duration(2.0)])
scene.render()
```



36.1.45. Drawable.shadow

MÉTODO `shadow(color: ColorLike, x: float = 0.08, y: float = -0.08, blur: float = 0.06) -> Drawable`

Sombra proyectada desplazada y desenfocada. En un texto o un grupo, una sombra desenfocada se dibuja una sola vez a partir de la silueta de sus miembros rellenos y debajo de todos, como `drop-shadow` de CSS sobre un grupo: los solapes no se oscurecen y el coste no crece con cada glifo. Los miembros recortados, con mezcla o solo con trazo proyectan la suya.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`color`	`Color`	obligatorio	Color de la sombra; su alfa escala la opacidad.
`x / y`	`float`	`0.08 / -0.08`	Desplazamiento en unidades de escena.
`blur`	`float`	`0.06`	Desenfoque en unidades de escena.

Ejemplo:

```
card = scene.geometry.rounded_rect(4, 2, 0.2).fill("#18202E").shadow("#00000080", x=0.12, y=-0.12, blur=0.1)
```

36.1.46. Drawable.no_effects

MÉTODO `no_effects() -> Drawable`

Quita `glow`, `blur` y `shadow` sin tocar el relleno ni el trazo. Después del primer `play` o `wait` es un corte en el cursor: hasta ahí los efectos se ven.

36.1.47. Drawable.tip

MÉTODO `tip(end: Optional[Literal["arrow", "dot"]] = "arrow", start: Optional[Literal["arrow", "dot"]] = None, *, length: Optional[float] = None, width: Optional[float] = None) -> Drawable`

Pone puntas de flecha o puntos en los extremos de cualquier trazo: polilíneas, curvas, arcos o conectores con `via`. Se rellenan con el color del trazo y siguen a los extremos actuales del camino, así que acompañan a `trim`, `create()` y a los caminos que se regeneran; `animate.grow_arrow()` dibuja el trazo desde la cola con la punta delante. El vértice de la flecha es el extremo del camino y el trazo termina bajo su base. Si el camino es más corto que la punta, esta se encoge con él. `tip(None)` las quita. Un tipo desconocido o un tamaño no positivo lanzan `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`end / start`	`"arrow" | "dot" | None`	`"arrow" / None`	Punta de cada extremo del camino.
`length`	`float | None`	`None`	Largo de la punta de flecha; por defecto, cinco grosores de trazo y al menos 0.15.
`width`	`float | None`	`None`	Ancho de la base de la flecha (0.9 del largo) o diámetro del punto (tres grosores de trazo).

Ejemplo:

```
from gaanim import CYAN, Scene
scene = Scene(frame=(16, 9), background="#0f172a")
route = scene.geometry.polyline([(-5, -1.5), (-2, 1.5), (2, -1.5), (5, 1.5)]).no_fill()
route.stroke(CYAN, 0.06).tip(end="arrow", start="dot")
scene.play([route.animate.grow_arrow().duration(1.5)])
scene.render()
```

36.1.48. Drawable.blend

MÉTODO `blend(mode: BlendModeName = "normal") -> Drawable`

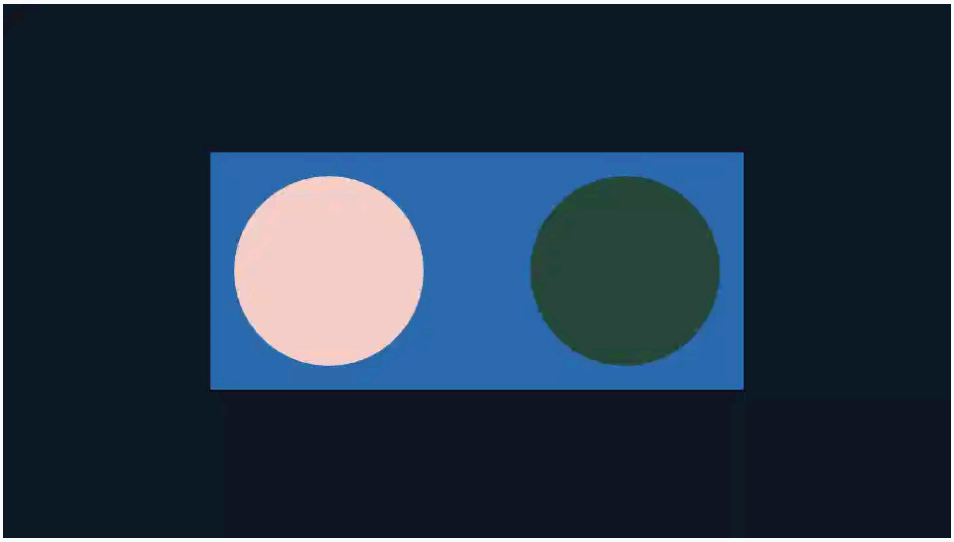
Modo de fusión con lo que hay debajo. `"screen"` y `"add"` aclaran (luces, destellos), `"multiply"` oscurece como tinta (resaltadores) y `"normal"` vuelve a pintar encima, también en un miembro de un grupo con otro modo. Como `fill`, en un grupo o un `Text` llega a todos sus miembros; un miembro al que le cambias el modo después conserva el suyo. El objeto se pinta en su propia capa, así que dentro de un recorte, un grupo con opacidad o una transición solo se funde con el contenido de ese grupo. No se anima. Un modo desconocido lanza `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`mode`	`str`	`"normal"`	`"normal"`, `"multiply"`, `"screen"`, `"overlay"`, `"darken"`, `"lighten"`, `"color_dodge"`, `"color_burn"`, `"hard_light"`, `"soft_light"`, `"difference"`, `"exclusion"`, `"hue"`, `"saturation"`, `"color"`, `"luminosity"` o `"add"`.

Ejemplo:

```
from gaanim import Scene
scene = Scene(frame=(16, 9), background="#101826")
panel = scene.geometry.rect(9, 4).fill("#2B6CB0").move_to(0, 0)
leak = scene.geometry.circle(1.6).fill("#F6AD55").move_to(-2.5, 0).blend("screen")
ink = scene.geometry.circle(1.6).fill("#F6AD55").move_to(2.5, 0).blend("multiply")
scene.play([leak.animate.move_to(-1, 0).duration(1.0), ink.animate.move_to(1, 0).duration(1.0)])
scene.render()
```



36.1.49. Drawable.echo

MÉTODO `echo(count: int = 5, *, delay: float = 0.04, decay: float = 0.6, hold: bool = False, start: Optional[float] = None, end: Optional[float] = None) -> Drawable`

Copias que siguen al objeto en el tiempo, como el efecto Echo de After Effects: la copia `k` lo muestra como estaba hace `k * delay` segundos, con `decay ** k` de su opacidad y debajo de él. Cada copia repite las animaciones del propio objeto (`animate`, `create`, fundidos, color y forma) con ese retraso, así que es exacta en cualquier búsqueda y en todas las exportaciones, SVG incluido. Se ocultan mientras el objeto está oculto y no cruzan un corte de segmento. No retrasan el movimiento de los *updaters*, de las posiciones reactivas ni de un grupo padre que se mueve, ni copian Lottie o vídeo. En un `Text`, cada glifo repite sus propias animaciones. Se declara una vez y vale para toda la línea de tiempo.

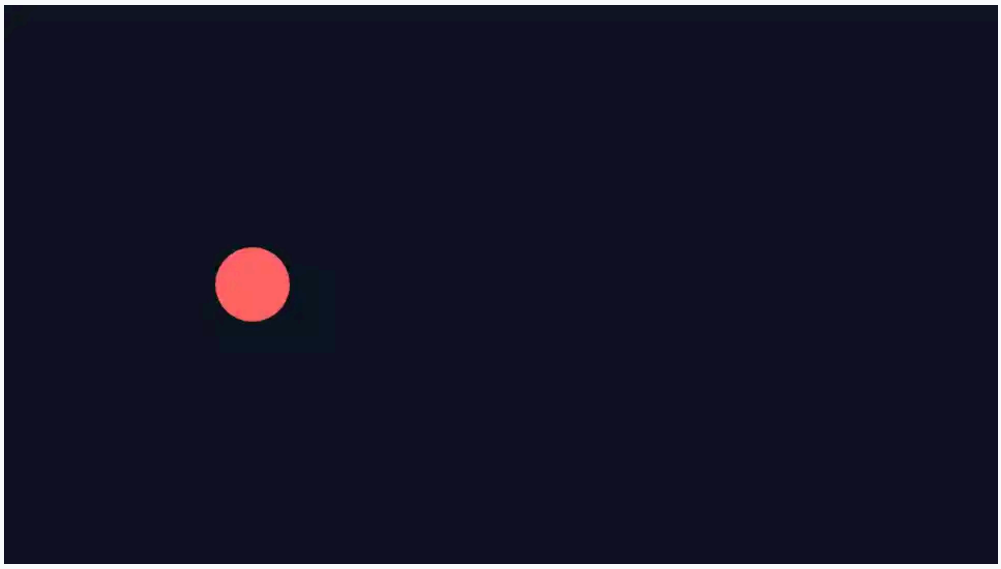
Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`count`	`int`	`5`	Copias, de 1 a 32; `0` quita el eco.
`delay`	`float`	`0.04`	Segundos entre copias; positivo.
`decay`	`float`	`0.6`	Opacidad de cada copia respecto a la anterior, en `(0, 1]`.
`hold`	`bool`	`False`	Con `True` las copias se retrasan a lo largo del movimiento y no del reloj: cuando el objeto se detiene, quedan congeladas donde estaban (papel cebolla) y siguen cuando vuelve a moverse. Sin él, alcanzan al objeto al detenerse y dejan de dibujarse: una copia que coincide con el objeto no se apila sobre él, así que un objeto translúcido en reposo se ve igual que sin eco.
`start / end`	`float | None`	`None`	Segundos de la escena que graban las copias: cada copia muestra el objeto solo como estaba entre `start` y `end`, y se oculta fuera de ese intervalo. Así la estela crece desde `start`, se vacía después de `end` y no aparece en el resto de movimientos. Cualquiera de los dos puede quedar abierto; `end` debe ser mayor que `start`.

Ejemplo:

```
from gaanim import CORAL, GOLD, Scene
scene = Scene(frame=(16, 9), background="#0e1422")
ball = scene.geometry.circle(0.45).fill(CORAL).move_to(-6, 0).echo(6, delay=0.05, decay=0.7)
scene.play([ball.animate.move_to(6, 0).duration(1.2)])
scene.play([ball.animate.move_to(0, 0).fill(GOLD).duration(0.6)])
scene.render()
```

```
from gaanim import CORAL, Scene
scene = Scene(frame=(16, 9), background="#0e1422")
# La estela solo existe durante la sacudida, de 1.0 s a 1.6 s.
head = scene.geometry.circle(0.6).fill(CORAL).move_to(-4, 0).echo(6, delay=0.04, start=1.0, end=1.6)
scene.play([head.animate.move_to(0, 0).duration(1.0)])
scene.play([head.animate.wiggle().duration(0.6)])
scene.play([head.animate.move_to(4, 0).duration(1.0)])
scene.render()
```



36.1.50. Drawable.count

MÉTODO `count(count: float) -> Drawable`

Muestra las primeras `count` copias de un grupo creado con `Geometry.repeat` o `Geometry.duplicate`; las demás siguen en el grupo pero ocultas, de la última hacia atrás. Un recuento fraccionario funde la siguiente copia: `2.5` muestra dos copias y media tercera. Antes del primer `scene.play` fija el recuento inicial; después lo cambia en el cursor. Para animarlo usa

`animate.count`. Un valor fuera de rango, o un objeto que no sea uno de esos grupos, lanza `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`count`	`float`	obligatorio	Copias visibles, de 0 al número de copias del grupo.

Ejemplo:

```
ring = scene.geometry.duplicate(scene.geometry.dot(0.1), Distribution.circle(16, 2.5)).count(4)
```

36.1.51. Drawable.points

MÉTODO `points(points: Sequence[tuple[float, float]]) -> Drawable`

Fija todos los vértices de un polígono o una polilínea. Antes del primer `scene.play` cambia la forma declarada, así que varias figuras pueden nacer iguales y deformarse luego; después la cambia en el cursor. `animate.points` sigue desde ahí. Una polilínea con `closed=True` sigue cerrada. Otro número de puntos, un valor no finito, un objeto que no sea polígono ni polilínea o una polilínea con coordenadas reactivas lanzan `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`points`	`Sequence[tuple[float, float]]`	obligatorio	Un punto por vértice, en las coordenadas en que se declaró la figura.

Ejemplo:

```
tri = scene.geometry.polygon([(0, 0), (1, 0), (0, 1)]).points([(-1, -1), (1, -1), (0, 1)])
scene.play([tri.animate.points([(-2, -1), (2, -1), (0, 2)]).duration(1.0)])
```

36.1.52. Drawable.squash_stretch

MÉTODO `squash_stretch(amount: float = 0.1, max_ratio: float = 1.6) -> Drawable`

Estira el objeto en la dirección de su velocidad y lo aplasta en la perpendicular, conservando el área: el estiramiento es `1 + amount * velocidad`, como mucho `max_ratio`, y el aplastamiento su inverso. La velocidad sale de las animaciones del propio objeto medidas en `t ± 1/60` s, así que un objeto quieto nunca se deforma y las búsquedas son exactas. Los miembros de un grupo se deforman con él. No cuentan el movimiento de los *updaters*, las posiciones reactivas ni un padre que se mueve. Un `amount` negativo o un `max_ratio` menor que 1 lanzan `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`amount`	`float`	`0.1`	Estiramiento por unidad de velocidad (unidades de escena por segundo); `0` quita el efecto.
`max_ratio`	`float`	`1.6`	Estiramiento máximo, al menos 1.

Ejemplo:

```
from gaanim import CORAL, Scene
scene = Scene(frame=(16, 9), background="#0f1729")
ball = scene.geometry.circle(0.5).fill(CORAL).move_to(-6, 0).squash_stretch(0.08, max_ratio=1.8)
scene.play([ball.animate.move_to(6, 0).duration(1.0)])
scene.play([ball.animate.move_to(0, -2.5).duration(0.5)])
scene.render()
```

36.1.53. Drawable.stroke_profile

MÉTODO `stroke_profile(profile: Optional[Sequence[tuple[float, float]]]) -> Drawable`

Trazo de grosor variable, como los perfiles de ancho de Illustrator: el ancho se interpola linealmente entre los puntos, así que `[(0, 0), (0.5, 1), (1, 0)]` se ensancha en el centro y termina en punta. El trazo se dibuja como un contorno relleno: no admite guiones, pero `trim`, `create` y `show_passing_flash` recortan la parte visible y el perfil se ajusta a ella, así que un destello afinado sigue acabando en punta. En un grupo o un `Text` llega a todos sus miembros. Una posición fuera de [0, 1] o un factor negativo lanzan `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`profile`	`Sequence[tuple[float, float]] | None`	obligatorio	Pares `(posición, factor)`: posición en [0, 1] de la longitud visible y

			factor del ancho del trazo; `None` vuelve al trazo normal.

Ejemplo:

```
import math
from gaanim import CORAL, GOLD, TEAL, Scene
scene = Scene(frame=(16, 9), background="#101826")
brush = scene.geometry.arc(-4, 0, 2.0, 0.3, 4.2).no_fill().stroke(GOLD, 0.35).stroke_taper(0.35,
0.5)
wave = scene.geometry.polyline([(x / 10, 1 + 0.6 * math.sin(x / 4)) for x in range(-10, 61)])
wave.stroke(TEAL, 0.3).stroke_profile([(0.0, 0.1), (0.5, 1.0), (1.0, 0.1)])
flash = scene.geometry.polyline([(-1, -2.2), (2, -1.2), (5, -2.6), (7, -1.8)]).stroke(CORAL,
0.25).stroke_taper(0.5, 0.5)
scene.play([brush.animate.create(), wave.animate.create()])
scene.play([flash.animate.show_passing_flash(time_width=0.5)])
scene.render()
```

36.1.54. Drawable.stroke_taper

MÉTODO `stroke_taper(start: float = 0.2, end: float = 0.2) -> Drawable`

Atajo de `stroke_profile` para pinceladas y flechas caligráficas.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`start / end`	`float`	`0.2`	Fracción de la longitud visible en la que el trazo se afina hasta una punta, al principio y al final; `start + end <= 1`.

36.1.55. Drawable.annotate

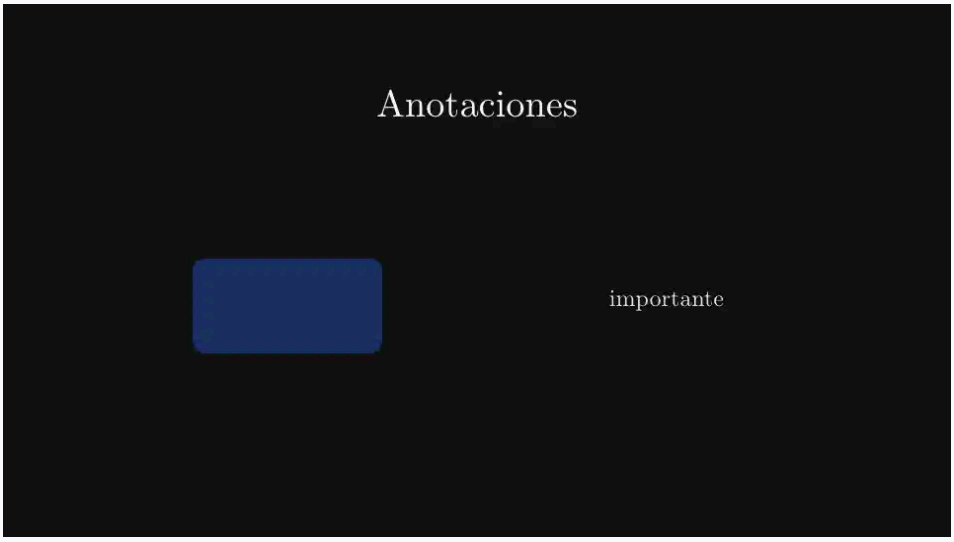
PROPIEDAD `annotate: Annotations`

Subrayados, cajas, círculos, tachados y corchetes con trazo de mano alzada, al estilo de rough-notation. Cada método devuelve un `Drawable` nuevo con trazo: las líneas tiemblan y se arquean como dibujadas a mano, y la forma se dibuja `passes` veces sobre sí misma con otro azar. La marca sigue los límites del objetivo en cada fotograma, conserva su garabato cuando el objetivo se mueve y permanece oculta hasta que un `play` la incluye, así que `marca.animate.create()` la dibuja pasada a pasada. Se puede desvanecer o reestilizar como cualquier drawable, pero se mueve moviendo su objetivo. La misma `seed` dibuja siempre el mismo garabato. También existe en las selecciones de texto (`TextSelection.annotate`); para resaltar con rotulador usa `TextSelection.marker`.

Devuelve: `Annotations` — Constructor de anotaciones alrededor de este drawable.

Ejemplo:

```
from gaanim import *

scene = Scene(frame=(16, 9))
title = scene.text("Anotaciones", role="title").move_to(0, 2.6)
card = scene.geometry.rounded_rect(3.2, 1.6, 0.2).fill(BLUE).move_to(-3.2, -0.6)
word = scene.text("importante").move_to(3.2, -0.6)
ul = title.annotate.underline(color=RED, seed=1)
box = card.annotate.box(color=GOLD, seed=2)
circle = word.annotate.circle(color=RED, seed=3)
scene.play([ul.animate.create(), box.animate.create()], duration=1.2)
scene.play([circle.animate.create()], duration=0.9)
scene.render()
```



36.1.56. Annotations.underline / box / circle / strike_through / crossed_off

MÉTODO

`underline` traza una línea bajo el objetivo (por defecto 0.08 por debajo y 0.04 a cada lado), `box` una caja (0.1 por lado), `circle` una elipse que pasa por el centro de cada lado acolchado (0.25

arriba y abajo, 0.35 a los lados), `strike_through` una línea por el centro (0.06 a cada lado) y `crossed_off` las dos diagonales (0.05 por lado). Un `roughness` o `padding` negativo o no finito, `passes` fuera de 1 a 8 o un `width` no positivo lanzan `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`color`	`ColorLike | None`	`None`	Color del trazo; sin color se usa el trazo del tema.
`width`	`float | None`	`None`	Grosor del trazo; requiere `color`.
`roughness`	`float`	`1.0`	Escala del temblor: `0` dibuja geometría limpia y `1` el aspecto de rough-notation.
`passes`	`int`	`2`	Veces que se dibuja la forma, de 1 a 8.
`seed`	`int`	`0`	Semilla del garabato.
`padding`	`Padding | None`	`None`	Crecimiento de los límites del objetivo: escalar, `(vertical, horizontal)` o `(arriba, derecha, abajo, izquierda)`.

Devuelve: `Drawable` — La marca, oculta hasta que un `play` la incluye.

Ejemplo:

```
price = scene.text("99 €")
scene.play([price.annotate.crossed_off(color=RED, roughness=1.5, passes=1).animate.create()])
```

36.1.57. Annotations.bracket

MÉTODO `bracket(sides: str | Sequence[str] = ("left", "right"), *, color: Optional[ColorLike] = None, width: Optional[float] = None, roughness: float = 1.0, passes: int = 2, seed: int = 0, padding: Optional[Padding] = None) -> Drawable`

Corchetes a mano alzada en los lados elegidos; acepta los demás argumentos de `underline`, con `padding` de 0.05 arriba y abajo y 0.1 a los lados. Un lado desconocido lanza `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`sides`	`str | Sequence[str]`	`("left", "right")`	Lados con corchete: `"left"`, `"right"`, `"top"` o `"bottom"`.

Devuelve: `Drawable` — Los corchetes, ocultos hasta que un `play` los incluye.

Ejemplo:

```
block = scene.text("primer paso\nsegundo paso")
scene.play([block.annotate.bracket("left", color=BLUE).animate.create()])
```

36.1.58. Drawable.modifiers

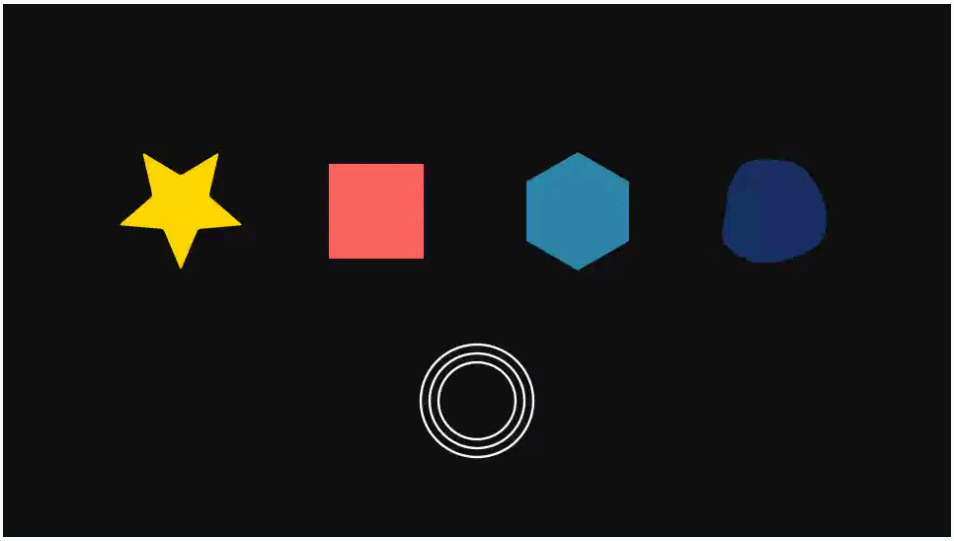
PROPIEDAD `modifiers: PathModifiers`

Modificadores no destructivos, como los operadores de forma de After Effects y Lottie: cada método añade uno a partir del cursor y lo devuelve, y sus números se animan con `modificador.animate`. Se aplican en el orden en que se añadieron, sobre el trazado ya animado (recortes, `write`, morph) y justo antes de dibujarlo, a cada miembro con trazado de un grupo o un `Text`. `bounds()` y el layout siguen midiendo la forma original. Un trazado abierto puede ganar área; las primitivas abiertas (`line`, `arc`, `polyline`, `traced_path`…) nacen sin relleno, así que solo se ve el trazo hasta que le des uno con `fill()`.

Devuelve: `PathModifiers` — La pila de modificadores de trazado del objeto.

Ejemplo:

```
from gaanim import BLUE, CORAL, GOLD, TEAL, WHITE, Scene
scene = Scene(frame=(16, 9))
star = scene.geometry.star(5, 1.1, 0.5).fill(GOLD).move_to(-5.0, 1.0)
zz = star.modifiers.zigzag(size=0.0, ridges=3)
star.modifiers.round_corners(0.04)
square = scene.geometry.square(1.6).fill(CORAL).move_to(-1.7, 1.0)
pb = square.modifiers.pucker_bloat(0.0)
hexagon = scene.geometry.regular_polygon(6, 1.0).fill(TEAL).move_to(1.7, 1.0)
tw = hexagon.modifiers.twist(0.0)
blob = scene.geometry.circle(0.9).fill(BLUE).move_to(5.0, 1.0)
blob.modifiers.wiggle_path(size=0.12, detail=3, frequency=1.5, seed=3)
ring = scene.geometry.circle(0.5).no_fill().stroke(WHITE, 0.04, align="center").move_to(0, -2.2)
rings = ring.modifiers.offset(0.15, join="round", copies=3)
scene.play([zz.animate.size(0.12), pb.animate.amount(-0.5), tw.animate.angle(1.4),
rings.animate.amount(0.3)])
scene.play([pb.animate.amount(0.4).duration(0.8), tw.animate.angle(-1.0).duration(0.8)])
scene.render()
```



36.1.59. PathModifiers.zigzag

MÉTODO `zigzag(size: float = 0.1, ridges: int = 4, *, smooth: bool = False) -> PathModifier`

Los vértices quedan en su sitio, así que con `size = 0` la forma no cambia.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`size`	`float`	`0.1`	Distancia de cada pico a cada lado del trazado, en unidades. Animable.
`ridges`	`int`	`4`	Picos por segmento, de 1 a 256.
`smooth`	`bool`	`False`	Ondas en lugar de esquinas.

Devuelve: `PathModifier` — El modificador, que anima `size`.

36.1.60. PathModifiers.round_corners

MÉTODO `round_corners(radius: float) -> PathModifier`

Redondea las esquinas entre dos tramos rectos; un lado más corto que dos radios limita sus esquinas y las curvas conservan sus vértices.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`radius`	`float`	obligatorio	Radio de cada esquina, `>= 0`. Animable.

Devuelve: `PathModifier` — El modificador, que anima `radius`.

36.1.61. PathModifiers.pucker_bloat

MÉTODO `pucker_bloat(amount: float) -> PathModifier`

Los vértices se acercan al centro del trazado en esa fracción y las curvas entre ellos se alejan, o al revés con valores negativos.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`amount`	`float`	obligatorio	Entre `-1` (pucker, puntas) y `1` (bloat, lóbulos). Animable.

Devuelve: `PathModifier` — El modificador, que anima `amount`.

36.1.62. PathModifiers.twist

MÉTODO `twist(angle: float) -> PathModifier`

El giro se desvanece hasta cero en el vértice más lejano del centro.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`angle`	`float`	obligatorio	Giro en el centro, en radianes. Animable.

Devuelve: `PathModifier` — El modificador, que anima `angle`.

36.1.63. PathModifiers.wiggle_path

MÉTODO `wiggle_path(size: float = 0.1, *, detail: int = 6, frequency: float = 1.0, seed: int = 0) -> PathModifier`

Contorno vivo con ruido coherente con semilla, desde el instante en que se añade; los puntos se unen con curvas suaves. La misma semilla da la misma forma en el mismo tiempo, así que un seek es exacto.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`size`	`float`	`0.1`	Desplazamiento máximo a lo largo de la normal, en unidades. Animable.
`detail`	`int`	`6`	Puntos por segmento, de 1 a 256.
`frequency`	`float`	`1.0`	Cambios por segundo. Animable.
`seed`	`int`	`0`	Semilla del ruido.

Devuelve: `PathModifier` — El modificador, que anima `size` y `frequency`.

36.1.64. PathModifiers.offset

MÉTODO `offset(amount: float, *, join: Literal["miter", "round", "bevel"] = "miter", copies: int = 1) -> PathModifier`

Agranda o encoge las formas cerradas; encoger más allá del radio interior no deja nada. Un trazado abierto se vuelve una banda de `|amount|` a cada lado. Las copias forman un solo trazado, así que un trazo alineado por dentro (el predeterminado) solo muestra la mitad interior del anillo exterior: para anillos usa `align="center"`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`amount`	`float`	obligatorio	Unidades hacia fuera, o hacia dentro si es negativo. Animable.
`join`	`"miter" | "round" | "bevel"`	`"miter"`	Cómo gira el contorno en las esquinas.

`copies`	`int`	`1`	Contornos a `amount`, `2 * amount`, ..., de 1 a 64.

Devuelve: `PathModifier` — El modificador, que anima `amount`.

36.1.65. PathModifier.animate

PROPIEDAD `animate: PathModifierAnimation`

Anima los números del modificador como cualquier otra animación. Un nombre que el modificador no tiene lanza `ValueError`; `PathModifier.names` los enumera y `PathModifier.set(size=0.2)` los cambia de golpe desde el cursor.

Devuelve: `PathModifierAnimation` — Métodos `size`, `radius`, `amount`, `angle` y `frequency`, que devuelven un `Anim`.

36.1.66. Drawable.motion_blur

MÉTODO `motion_blur(enabled: bool = True) -> Drawable`

Si el desenfoque de movimiento de la escena lo difumina. Con `False`, cada subcuadro lo dibuja como está en el tiempo del cuadro, por ejemplo para un título o un HUD que no deben emborronarse mientras todo lo demás se mueve. Las copias de `echo` heredan el ajuste.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`enabled`	`bool`	`True`	`False` mantiene nítido el objeto y sus miembros.

36.1.67. Drawable.post_process

MÉTODO `post_process(enabled: bool = True) -> Drawable`

Si el postprocesado de la escena, o el de su segmento, se aplica al objeto. Con `False` se dibuja encima del cuadro ya procesado, por ejemplo una etiqueta o un HUD que `pixelate`, `glitch` o `halftone` harían ilegibles. Queda encima de todo lo procesado, sea cual sea su `z_index`, y sigue las transiciones de su segmento; en una transición por shader se funde con su progreso. Las copias de `echo` heredan el ajuste.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`enabled`	`bool`	`True`	`False` deja fuera del postprocesado el objeto y sus miembros.

Ejemplo:

```python
from gaanim import CORAL, PostProcess, Scene
scene = Scene(frame=(16, 9), background="#0f172a")
scene.canvas.post = PostProcess.pixelate(24)
scene.geometry.circle(2).fill(CORAL).move_to(-3, 0)
scene.text("pixelado", size=0.9).move_to(-3, -3)
scene.text("legible", size=0.9).move_to(3, -3).post_process(False)
scene.wait(0.5)
scene.render()
```

36.1.68. Drawable.trim

MÉTODO `trim(start: Optional[float] = None, end: Optional[float] = None, offset: Optional[float] = None, mode: Optional[Literal["simultaneous", "sequential"]] = None) -> Self`

Muestra solo una parte del trazo, como los *trim paths* de After Effects. Los valores omitidos conservan el ajuste anterior. Anímalo con `animate.trim(...)`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`start / end`	`float | None`	`None`	Ventana visible del camino como fracción de su longitud; al principio valen 0 y 1.
`offset`	`float | None`	`None`	Desplaza la ventana; da la vuelta al final del camino.
`mode`	`str | None`	`None`	`"simultaneous"` (predeterminado) recorta cada subcamino a la vez; `"sequential"` recorta la longitud total y los subcaminos aparecen uno tras otro.

Ejemplo:

```python
from gaanim import GOLD, Scene
scene = Scene(frame=(16, 9), background="#0f172a")
```

```
ring = scene.geometry.circle(1.5).no_fill().stroke(GOLD, 0.08).trim(end=0.0)
scene.play([ring.animate.trim(end=1.0).duration(1.0)])
scene.play([ring.animate.trim(start=0.7, offset=0.5).duration(0.8)])
scene.render()
```

36.1.69. Drawable.clip

MÉTODO `clip(mask: Drawable, rule: Literal["nonzero", "evenodd"] = "nonzero", invert: bool = False) -> Drawable`

Recorte vectorial vivo: la máscara puede moverse, escalarse o rotar y el recorte se recalcula en cada fotograma. La máscara sigue visible; usa `no_fill().no_stroke()` si solo debe definir la silueta. Una regla desconocida lanza `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`mask`	`Drawable`	obligatorio	Silueta vectorial que recorta este objeto.
`rule`	`str`	`"nonzero"`	Regla de relleno de la máscara: `"nonzero"` o `"evenodd"`.
`invert`	`bool`	`False`	Recorta al exterior de la máscara.

Ejemplo:

```
from gaanim import BLUE, GOLD, Scene
scene = Scene(frame=(16, 9), background="#0f172a")
stripes = scene.geometry.group([
    scene.geometry.rect(0.4, 4).fill(GOLD if i % 2 else BLUE).move_to(-3 + 0.4 * i, 0)
    for i in range(16)
])
lens = scene.geometry.circle(1.2).no_fill().no_stroke().move_to(-2, 0)
stripes.clip(lens)
scene.play([lens.animate.move_to(2, 0).duration(1.2)])
scene.render()
```

36.1.70. Drawable.no_clip

MÉTODO `no_clip() -> Drawable`

Quita el recorte por máscara. En las marcas de datos de un espacio cartesiano, deja que la marca sobresalga del área de trazado.

36.1.71. Drawable.camera_view

MÉTODO `camera_view(frame: Optional[Drawable] = None, *, center: tuple[float, float] | Drawable | AnchorPoint | None = None, zoom: ScalarSource | None = None, fit: Literal["contain", "cover", "stretch"] = "contain", background: Paint | Literal["canvas"] | None = "canvas", exclude: Sequence[Drawable] = (), layers: Sequence[str] = ()) -> CameraView`

Convierte esta figura en una pantalla que muestra lo que ve una segunda cámara, como un *picture-in-picture* o una lupa. La pantalla dibuja su relleno, encima la vista recortada a su contorno y encima su trazo; la opacidad y los fundidos afectan a todo. La vista no muestra la pantalla, el marco, `exclude` ni las superposiciones HUD; una pantalla que aparece dentro de otra vista muestra su pintura pero no su vista. Solo una figura cerrada 2D puede ser pantalla (rectángulo, cuadrado, círculo, punto, elipse, polígono, estrella, sector, anillo, curva, camino SVG o booleano). Lanza `ValueError` con texto, grupos, imágenes u objetos 3D, con `frame` igual a la propia pantalla, con objetos o parámetros de otra escena, con un zoom no positivo, con un nombre de capa vacío o con un `fit` o `background` desconocido. Una llamada posterior o `no_camera_view()` sustituye la vista.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`frame`	`Drawable | None`	`None`	Objeto que hace de cámara: su

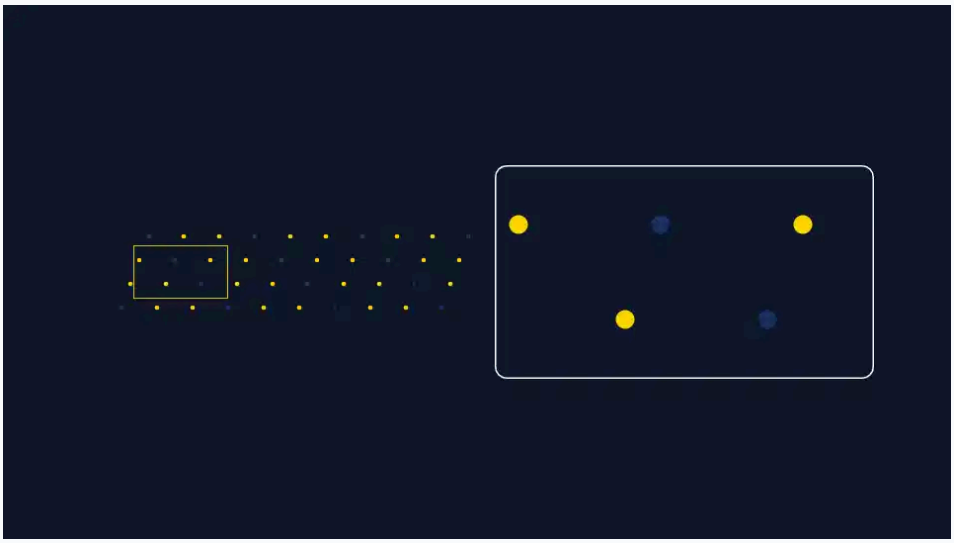
			centro marca adónde mira y su giro gira la vista en sentido contrario. Sin `zoom`, su tamaño también fija el aumento, así que reducirlo acerca. Sin `frame`, la cámara mira a través de un marco invisible.
`center`	`tuple | Drawable | AnchorPoint | None`	`None`	Adónde mira la cámara cuando no hay `frame`: un punto, el centro de un objeto o un punto de anclaje; por defecto, el origen. Con `frame` lanza `ValueError`.
`zoom`	`float | Parameter | Computed | None`	`None`	Aumento fijo en lugar del tamaño del marco. Un número da a la vista su propio zoom, que `animate.zoom_to` cambia a velocidad percibida constante; un `Parameter` se anima de forma lineal;

			un `Computed` se sigue y no se puede fijar. Sin `frame` ni `zoom`, vale 1.
`fit`	`str`	`"contain"`	Con un aumento dado por el marco: `"contain"` muestra el marco entero, `"cover"` llena la pantalla y `"stretch"` escala cada eje por separado.
`background`	`Paint | str | None`	`"canvas"`	Fondo de la vista: `"canvas"` pinta el fondo de la escena tal como lo ve la cámara, un color o `Brush` pinta la pantalla y `None` deja ver el relleno propio de la pantalla.
`exclude`	`Sequence[Drawable]`	`()`	Objetos, con sus hijos, que la vista no muestra.
`layers`	`Sequence[str]`	`()`	Capas de vista (ver `view_layer`) que la vista muestra además de los objetos normales.

Devuelve: `CameraView` — La cámara, con `pan_to`, `zoom_to`, `rotate_to`, `follow` y `animate`; ver Escena.

Ejemplo:

```
from gaanim import BLUE, GOLD, WHITE, Scene
scene = Scene(frame=(16, 9), background="#0f172a")
for i in range(40):
    scene.geometry.dot(0.04).fill(GOLD if i % 3 else BLUE).move_to(-6 + 0.15 * i, 0.4 * (i % 4) -
0.6)
frame = scene.geometry.rect(1.6, 0.9).no_fill().stroke(GOLD, 0.03).move_to(-5, 0)
screen = scene.geometry.rounded_rect(6.4, 3.6, 0.2).stroke(WHITE, 0.05).move_to(3.5, 0)
screen.camera_view(frame)
scene.play([frame.animate.move_to(-1.5, 0).duration(1.2)])
scene.play([frame.animate.scale_to(0.5).duration(0.8)])
scene.render()
```



36.1.72. Drawable.no_camera_view

MÉTODO `no_camera_view() -> Drawable`

Quita la vista de cámara desde el cursor: a partir de ahí la figura vuelve a dibujarse como una forma normal, y antes la vista se sigue viendo.

36.1.73. Drawable.view_layer

MÉTODO `view_layer(name: Optional[str]) -> Drawable`

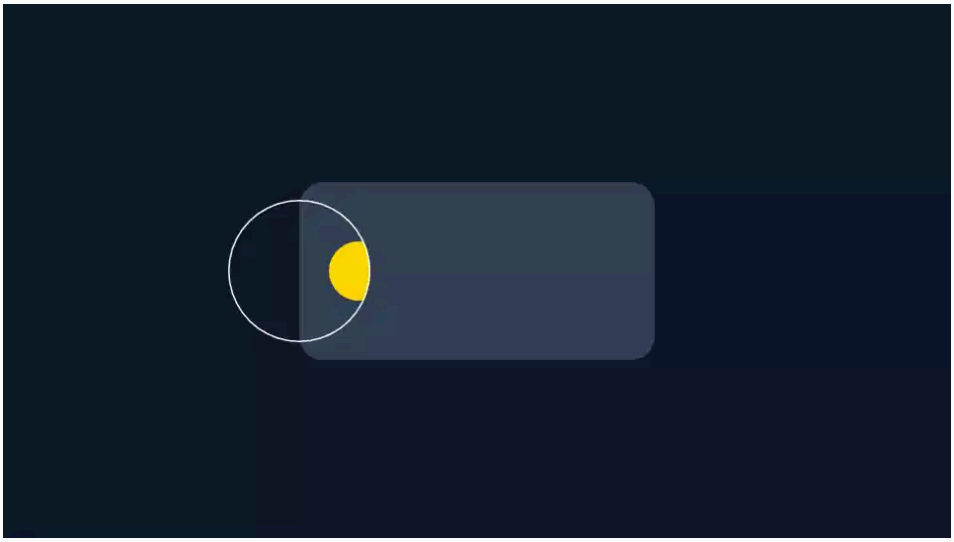
Pone el objeto y sus hijos en una capa de vista: la cámara principal no lo dibuja y solo lo muestran las vistas de cámara que listan esa capa en `layers`. Sirve para una lupa de rayos X que revela lo que hay debajo, o para etiquetas que solo aparecen en el zoom. Un nombre vacío lanza `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`name`	`str | None`	obligatorio	Nombre de la capa; `None` devuelve el objeto a ninguna capa.

Ejemplo:

```
from gaanim import GOLD, WHITE, Scene
scene = Scene(frame=(16, 9), background="#0f172a")
scene.geometry.rounded_rect(6, 3, 0.4).fill("#334155").no_stroke()
for x in (-2, 0, 2):
    scene.geometry.circle(0.5).fill(GOLD).no_stroke().move_to(x, 0).view_layer("xray")
lens = scene.geometry.circle(1.2).stroke(WHITE, 0.05).move_to(-3, 0)
view = lens.camera_view(center=(-3, 0), layers=["xray"])
scene.play([lens.animate.move_to(3, 0).duration(2.0), view.animate.pan_to(3, 0).duration(2.0)])
scene.render()
```



36.1.74. Animación

El proxy que convierte los setters en animaciones y las acciones de modelos importados.

36.1.75. Drawable.animate

PROPIEDAD `animate: Anim`

Proxy puro de animación compuesta. Encadena destinos de transformación, opacidad, relleno, trazo, color o material; todos comparten duración y easing y corren a la vez. El resultado es un `Anim` que no cambia nada hasta pasarlo a `scene.play`.

Ejemplo:

```
scene.play([node.animate.move_to(3, 0).fill(GOLD).scale_to(1.5).duration(1.2)])
```

Gaanim ya no importa glTF, así que tampoco reproduce sus acciones: `Drawable.animations()` devuelve una tupla vacía y `Drawable.animation(name, ...)` lanza `NotImplementedError`. `part` y `parts` están en Medios.

36.1.76. Relaciones reactivas

Hacen que un objeto siga a otro en el mismo fotograma sin callbacks de Python. El seguidor queda oculto hasta su animación de entrada. Para simulaciones y series muestreadas, consulta `add_updater_fn` y `drive_from_samples` en Animaciones.

36.1.77. Drawable.follow

MÉTODO `follow(source: Endpoint, *, offset: tuple[float, float] = (0.0, 0.0), offset_space: Literal["world", "local"] = "world", delay: float = 0.0) -> Self`

Sigue cualquier extremo en el mismo fotograma y devuelve el objeto. Con `delay` copia al líder con retardo, como `valueAtTime(time - delay)` de After Effects: encadena seguidores con retardos crecientes para estelas y colas (*overlapping action*). Las animaciones del líder se evalúan de nuevo en `t - delay` en cada fotograma, así que un seek a cualquier instante reproduce la estela sin estado acumulado. El líder puede seguir a su vez a otro objeto (`follow` con o sin retardo, `follow_to`, `attach_to`), así que los seguidores de seguidores forman cadenas; hasta que el segmento actual lleva `delay` segundos, el seguidor espera en la posición del líder al inicio del segmento. Offsets o retardos no finitos, retardos negativos o modos inválidos lanzan `ValueError`; un retardo positivo con una fuente que no es Drawable lanza `TypeError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`source`	`Endpoint`	obligatorio	Drawable, `AnchorPoint`, `PointRef` o tupla.
`offset`	`tuple[float, float]`	`(0.0, 0.0)`	Desplazamiento respecto a la fuente.
`offset_space`	`str`	`"world"`	`"world"` mantiene el desplazamiento alineado con la pantalla; `"local"` lo rota y escala con la fuente.
`delay`	`float`	`0.0`	Segundos de retardo: el objeto va donde estaba `source` hace `delay` segundos. Con un valor positivo `source` debe ser un Drawable.

Ejemplo:

```
from gaanim import GOLD, WHITE, Scene
scene = Scene(frame=(16, 9), background="#0f172a")
theta = scene.viz.parameter(0.2)
tip = scene.geometry.polar_point((0, 0), 1.5, theta)
bar = scene.mechanics.bar_between((0, 0), tip).stroke(WHITE, 0.09)
label = scene.text("punta").fill(GOLD).follow(tip, offset=(0, 0.225))
scene.play([bar.animate.fade_in(), label.animate.write(), theta.animate.set(2.2).duration(1.4)])
scene.render()
```

Una cola que sigue al líder con retardo:

```
from gaanim import CORAL, GOLD, Scene
scene = Scene(frame=(16, 9), background="#0f172a")
leader = scene.geometry.circle(0.35).fill(GOLD).move_to(-5, 0)
dots = [scene.geometry.circle(0.25 - 0.04 * i).fill(CORAL) for i in range(5)]
for i, dot in enumerate(dots):
    dot.follow(leader, delay=0.06 * (i + 1))
scene.play([dot.animate.fade_in().duration(0.01) for dot in dots])
scene.play(leader.animate.move_to(5, 1.5).duration(1.2))
scene.play(leader.animate.move_to(0, -2).duration(0.8))
scene.wait(0.5)
scene.render()
```

36.1.78. Drawable.attach_to

MÉTODO `attach_to(source: Drawable) -> None`

Coloca el objeto sobre el centro de `source` y lo sigue mientras se mueve. Para mantener una separación usa `follow_to` o `follow`. Devuelve `None`, así que llámalo en su propia línea.

Ejemplo:

```
from gaanim import GOLD, Scene
scene = Scene(frame=(16, 9), background="#0f172a")
from gaanim import WHITE
mass = scene.geometry.dot(0.15).fill(GOLD).move_to(-1.5, 0)
ring = scene.geometry.circle(0.4).no_fill().stroke(WHITE, 0.04)
ring.attach_to(mass)
scene.play([ring.animate.create().duration(0.3), mass.animate.shift_by(3, 0).duration(1.2)])
scene.render()
```

36.1.79. Drawable.follow_to

MÉTODO `follow_to(source: Drawable, offset: tuple[float, float]) -> None`

Sigue el centro de `source` con un desplazamiento fijo `offset`. Devuelve `None`; para encadenar, usa `follow`.

Ejemplo:

```
ball = scene.geometry.circle(0.3).fill(BLUE)
tag = scene.text("v", role="label")
tag.follow_to(ball, (0, 0.6))
scene.play([tag.animate.fade_in(), ball.animate.shift_by(3, 0)])
```

36.1.80. Drawable.bind_x_from / bind_y_from / bind_position_from

MÉTODO `bind_x_from(source) · bind_y_from(source) · bind_position_from(source, axes="xy") -> None`

Copia la coordenada X, la Y o ambas de `source` en cada fotograma, donde quiera que esté: animada, movida por un updater o colocada con `follow(...)`. `axes` acepta `"x"`, `"y"` o `"xy"`. Útil para proyecciones sobre un eje. Devuelven `None`.

Ejemplo:

```
point = scene.geometry.dot(0.1).fill(GOLD).move_to(-2, 1)
shadow = scene.geometry.dot(0.08).fill(GRAY).move_to(0, -2)
shadow.bind_x_from(point)
scene.play([shadow.animate.fade_in(), point.animate.shift_by(4, 0.5)])
```

36.1.81. Drawable.bind_rotation_from

MÉTODO `bind_rotation_from(source: Drawable, *, ratio: float = 1.0, phase: float = 0.0) -> Self`

Copia la rotación mundial de `source` como `rotación * ratio + phase`, en radianes. Es la base del acoplamiento de engranajes. Cuenta la rotación de `source` y de sus padres, también la de un `Updater.rotate` y la de un `rotate_to(Parameter)`.

Ejemplo:

```
driver = scene.mechanics.gear(0.69, 20).move_to(-0.69, 0)
driven = scene.mechanics.gear(0.41, 12).move_to(0.41, 0).bind_rotation_from(driver, ratio=-5/3)
scene.play([driven.animate.fade_in(), driver.animate.rotate_by(2.0)])
```

36.1.82. Drawable.bind_translation_from_rotation

MÉTODO `bind_translation_from_rotation(source: Drawable, *, axis: Optional[Direction] = None, scale: float = 1.0) -> Self`

Traslada el objeto a lo largo de `axis` según el giro acumulado de `source` multiplicado por `scale`, como una cremallera movida por un piñón. Cuenta las vueltas completas de un `rotate_to(Parameter)` y de un `Updater.rotate`; las demás rotaciones, hasta dos vueltas en cada sentido. Es una relación visual, no un solver de contacto.

Ejemplo:

```
pinion = scene.mechanics.gear(0.69, 20).move_to(0, 0.25)
rack = scene.mechanics.rack(2.75, 18).move_to(0, -0.81).bind_translation_from_rotation(
    pinion, axis=Direction.RIGHT, scale=0.69,
)
scene.play([rack.animate.fade_in(), pinion.animate.rotate_by(2.0)])
```

36.1.83. Drawable.add_updater

MÉTODO `add_updater(updater: Updater) -> None`

Asocia un `Updater` nativo, por ejemplo `Updater.orbit(cx, cy, radio, velocidad)`. Se evalúa en Rust y sigue siendo exacto al hacer seek.

Ejemplo:

```
planet = scene.geometry.dot(0.12).fill(BLUE)
planet.add_updater(Updater.orbit(0, 0, 2, 1.0))
scene.wait(1)
```

36.1.84. Drawable.remove_updater

MÉTODO `remove_updater() -> None`

Quita los updaters y las series muestreadas asociados al objeto desde el cursor.

36.1.85. Transformaciones 3D

EXPERIMENTAL

La API 3D es experimental: todavía tiene errores conocidos y puede cambiar entre versiones. Úsala para explorar, pero no la tomes como estable.

Los mismos setters en tres ejes, para objetos colocados en el espacio de la cámara en perspectiva. Las rotaciones de Euler usan orden XYZ y radianes.

36.1.86. Drawable.move_to_3d

MÉTODO `move_to_3d(x: ScalarSource, y: ScalarSource, z: ScalarSource) -> Self`

Coloca el objeto en una posición del mundo 3D. Añade `.billboard()` para que una etiqueta mire siempre a la cámara o `.hud()` para fijarla a la pantalla. Acepta fuentes reactivas.

Ejemplo:

```
label = scene.text("origen").move_to_3d(0, 0, 0.5).billboard()
```

36.1.87. Drawable.shift_by_3d

MÉTODO `shift_by_3d(dx: float, dy: float, dz: float) -> Self`

Traslación relativa en tres ejes como corte inmediato.

36.1.88. Drawable.scale_to_3d

MÉTODO `scale_to_3d(x: ScalarSource, y: ScalarSource, z: ScalarSource) -> Self`

Escala absoluta independiente en X, Y y Z. Acepta fuentes reactivas.

36.1.89. Drawable.scale_by_3d

MÉTODO `scale_by_3d(x: float, y: float, z: float) -> Self`

Multiplica la escala de cada eje en el cursor.

36.1.90. Drawable.rotate_to_3d

MÉTODO `rotate_to_3d(x: ScalarSource, y: ScalarSource, z: ScalarSource) -> Self`

Rotación de Euler absoluta en radianes, en orden XYZ. Acepta fuentes reactivas.

Ejemplo:

```
panel = scene.geometry.rect(2, 1).move_to_3d(0, 0, 0).rotate_to_3d(0.3, 0.6, 0)
```

36.1.91. Drawable.rotate_by_3d

MÉTODO `rotate_by_3d(axis: Literal["x", "y", "z"], radians: float) -> Self`

Suma `radians` a la rotación alrededor del eje `"x"`, `"y"` o `"z"`.

36.1.92. Drawable.with_pivot_3d

MÉTODO `with_pivot_3d(x: float, y: float, z: float) -> Self`

Fija el pivote 3D de rotación y escala.

36.1.93. Drawable.billboard

MÉTODO `billboard() -> Self`

Mantiene un objeto 3D orientado hacia la cámara en perspectiva. Útil para etiquetas y marcadores.

36.1.94. Drawable.hud

MÉTODO `hud() -> Self`

Fija el objeto a la imagen como superposición: se queda en su sitio aunque la cámara 2D se desplace, acerque o gire, y aunque se mueva la cámara 3D. Sus coordenadas son las de la cámara sin mover, así que `.move_to(0, 3.5)` lo pone arriba en un fotograma de 16 × 9. Se dibuja encima de la escena y nunca aparece dentro de una vista de cámara. Pertenece a su segmento como cualquier objeto: `wipe`, `iris`, `blinds` y las demás transiciones vectoriales lo recortan con su segmento, y `push` y `slide` lo desplazan con él. En `cross_fade`, `fade_through` y `zoom_through`, que solo funden la opacidad, se ven a la vez el HUD del segmento que sale y el del que entra; si deben ocupar el mismo sitio, usa un solo HUD con `scene.persist(...)` y cambia su contenido, o dale al que entra una entrada que empiece al terminar la transición.

Ejemplo:

```
title = scene.text("Modelo 3D", role="title").hud().move_to(0, 3.5)
```

CAPÍTULO

37. Geometría

Fábricas de scene.geometry: primitivas, líneas, flechas, trayectorias, grupos, booleanas, geometría reactiva, dibujos axonométricos y 3D

`scene.geometry` crea las formas vectoriales de la escena. Cada fábrica devuelve un `Drawable` centrado en el origen, listo para encadenar estilo, posición y animación. Las medidas están en unidades de escena: el marco lógico mide 16 × 9.

```
from gaanim import BLUE, GOLD, WHITE, Scene

scene = Scene(frame=(16, 9), background="#0f172a")
disk = scene.geometry.circle(1).fill(BLUE).move_to(-3, 0)
card = scene.geometry.rounded_rect(2.4, 1.4, 0.2).fill(GOLD).move_to(0, 0)
star = scene.geometry.star(5, 1, 0.45).no_fill().stroke(WHITE, 0.05).move_to(3, 0)
scene.play([disk.animate.grow_from_center(), card.animate.create(), star.animate.create()],
duration=1.2)
scene.render()
```

37.1.1. Primitivas

Círculos, rectángulos, elipses y puntos: la base de casi cualquier diagrama.

37.1.2. Geometry.circle

FÁBRICA `circle(radius: float) -> Drawable`

Círculo centrado en el origen. Sirve para nodos, marcadores y diagramas radiales; combínalo con `annulus` para anillos.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción

`radius`	`float`	obligatorio	Radio en unidades de escena.

Ejemplo:

```
from gaanim import BLUE, Scene
scene = Scene(frame=(16, 9), background="#0f172a")
node = scene.geometry.circle(0.625).fill(BLUE).move_to(0, 0)
scene.play([node.animate.create().duration(1.0)])
scene.render()
```

37.1.3. Geometry.rect

FÁBRICA `rect(width: float, height: float) -> Drawable`

Rectángulo de `width` × `height` centrado en el origen. Úsalo para tarjetas, paneles y barras; `rounded_rect` da esquinas suaves.

Ejemplo:

```
from gaanim import BLUE, WHITE, Scene
scene = Scene(frame=(16, 9), background="#0f172a")
card = scene.geometry.rect(2, 1.125).fill(BLUE).stroke(WHITE, 0.025).move_to(0, 0)
scene.play([card.animate.grow_from_center().duration(0.9)])
scene.render()
```

37.1.4. Geometry.rounded_rect

FÁBRICA `rounded_rect(width: float, height: float, radius: float) -> Drawable`

Rectángulo con esquinas redondeadas de radio `radius`. Útil para botones, etiquetas y tarjetas.

Ejemplo:

```
from gaanim import GOLD, Scene
scene = Scene(frame=(16, 9), background="#0f172a")
btn = scene.geometry.rounded_rect(2, 0.625, 0.15).fill(GOLD).move_to(0, 0)
label = scene.text("PULSA").move_to(0, 0)
scene.play([scene.geometry.group([btn, label]).animate.fade_in().duration(0.6)])
scene.render()
```

37.1.5. Geometry.square

FÁBRICA `square(s: float) -> Drawable`

Atajo de `rect(s, s)`. Útil para celdas de rejilla.

Ejemplo:

```
cell = scene.geometry.square(1).fill(BLUE)
```

37.1.6. Geometry.ellipse

FÁBRICA `ellipse(rx: float, ry: float) -> Drawable`

Elipse con radios `rx` y `ry`. Sirve para órbitas, resaltados y diagramas de Venn.

Ejemplo:

```
from gaanim import GOLD, Scene
scene = Scene(frame=(16, 9), background="#0f172a")
orbit = scene.geometry.ellipse(1.125, 0.625).no_fill().stroke(GOLD, 0.025).move_to(0, 0)
scene.play([orbit.animate.create().duration(1.0)])
scene.render()
```

37.1.7. Geometry.dot

FÁBRICA `dot(radius: float) -> Drawable`

Círculo pequeño relleno para marcadores, viñetas y partículas.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`radius`	`float`	obligatorio	Radio en unidades de escena; suele estar entre 0.06 y 0.15.

Ejemplo:

```
marker = scene.geometry.dot(0.1).fill(RED).move_to(1, 0.5)
```

37.1.8. Geometry.points

FÁBRICA `points(positions: Sequence[tuple[float, float]], radius: float = 0.06) -> Drawable`

Nube de puntos sin `Cartesian2D`: todos los círculos comparten un camino, así que miles de puntos (una sección de Poincaré, una muestra) siguen siendo un único objeto para `fill`, `opacity`, el layout y la animación. Listas vacías, posiciones no finitas o un radio no positivo lanzan `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`positions`	`Sequence[tuple[float, float]]`	obligatorio	Centros en coordenadas de escena; al menos uno y todos finitos.
`radius`	`float`	`0.06`	Radio de cada punto, mayor que cero.

Ejemplo:

```
import math
from gaanim import GOLD, Scene
scene = Scene(frame=(16, 9), background="#0f172a")
spiral = [(0.02 * i * math.cos(0.2 * i), 0.02 * i * math.sin(0.2 * i)) for i in range(200)]
cloud = scene.geometry.points(spiral, radius=0.04).fill(GOLD)
scene.play([cloud.animate.fade_in().duration(0.5)])
scene.render()
```

37.1.9. Líneas y flechas

Segmentos, flechas rectas y curvas, arcos y conectores. Los extremos reactivos (`Endpoint`) aceptan tuplas, objetos, `AnchorPoint` y `PointRef`. Las líneas, arcos, polilíneas, curvas y trazos seguidos nacen sin relleno: solo dibujan su trazo aunque ganen área.

37.1.10. Geometry.line

FÁBRICA `line(p1: Endpoint, p2: Endpoint) | line(x1, y1, x2, y2) | line(*, length, direction=Direction.RIGHT) -> Drawable`

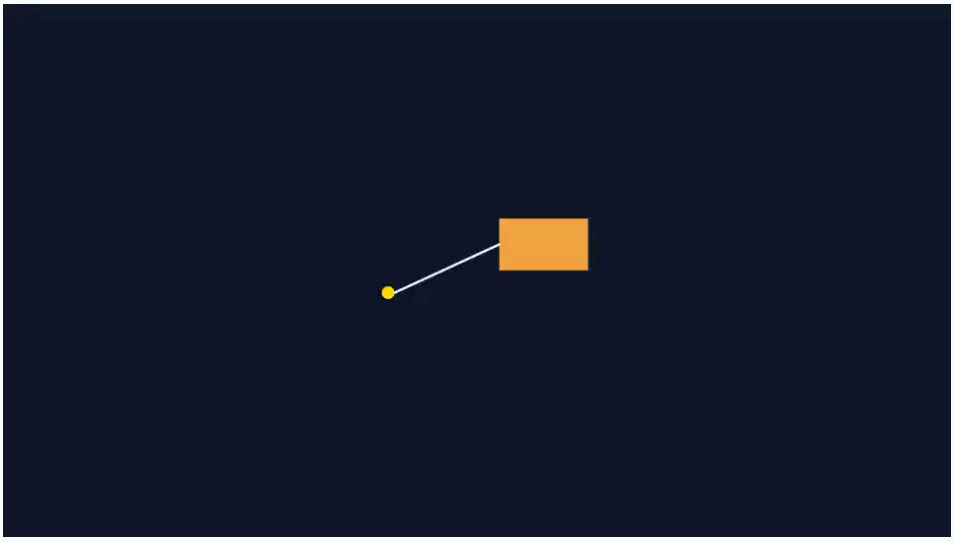
Dos tuplas crean una línea normal que se mueve con su grupo. Los extremos que son referencias se resuelven en el espacio del mundo en cada fotograma, así que la línea sigue a los objetos. Con `length`, `next_to` coloca la línea sin cambiar su longitud ni su orientación. Mezclar `length` con extremos lanza `TypeError`; longitudes o direcciones inválidas lanzan `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`p1 / p2`	`Endpoint`	obligatorio	Extremos fijos o reactivos.
`x1, y1, x2, y2`	`float`	obligatorio	Forma compatible con cuatro coordenadas fijas.
`length`	`float`	obligatorio	Longitud positiva de una línea centrada en el origen.
`direction`	`Direction`	`Direction.RIGHT`	Orientación con `length`; admite diagonales y vectores 2D no nulos.

Ejemplo:

```
from gaanim import Anchor, GOLD, WHITE, Scene
scene = Scene(frame=(16, 9), background="#0f172a")
left = scene.geometry.dot(0.11).fill(GOLD).move_to(-1.5, -0.375)
card = scene.geometry.rect(1.5, 0.875).move_to(1.125, 0.44)
connector = scene.geometry.line(
    left.anchor_point(Anchor.RIGHT),
    card.anchor_point(Anchor.LEFT),
).stroke(WHITE, 0.04)
scene.play([left.animate.shift_by(0.375, 0.75).duration(0.7)])
scene.render()
```



Con `length`, la línea es independiente de las coordenadas:

```
from gaanim import Direction, GOLD, WHITE, Scene
scene = Scene(frame=(16, 9), background="#0f172a")
title = scene.text("Una línea bajo el título", size=0.5).fill(WHITE)
rule = scene.geometry.line(length=5).stroke(GOLD, 0.035).next_to(
    title, Direction.DOWN, spacing=0.25,
)
scene.play([title.animate.fade_in(), rule.animate.create()], duration=1)
scene.render()
```

37.1.11. Geometry.arrow

FÁBRICA `arrow(x1: float, y1: float, x2: float, y2: float, *, head_length: Optional[float] = None, head_width: Optional[float] = None, body_width: Optional[float] = None, max_head_ratio: Optional[float] = None) -> Drawable`

Flecha sólida de `(x1, y1)` a `(x2, y2)`. `fill` la colorea entera: salvo que `stroke` fije otro, su contorno sigue al relleno (igual en `double_arrow`, `curved_arrow` y `curved_arrow_arc`). Para

flechas cortas añade `max_head_ratio=0.3`, y `.no_stroke()` para una silueta limpia. Extremos no finitos o medidas no positivas lanzan `ValueError`; extremos coincidentes dan un camino vacío.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`x1, y1`	`float`	obligatorio	Cola.
`x2, y2`	`float`	obligatorio	Punta.
`head_length / head_width / body_width`	`float | None`	`None`	Medidas en unidades de escena; por defecto 0.18, 0.15 y 0.036.
`max_head_ratio`	`float | None`	`None`	En `(0, 1]`, limita la punta respecto a la longitud total y escala su ancho en proporción.

Ejemplo:

```
from gaanim import GOLD, Scene
scene = Scene(frame=(16, 9), background="#0f172a")
arrow = scene.geometry.arrow(-3, 0, 3, 0).fill(GOLD).no_stroke()
scene.play([arrow.animate.create().duration(0.8)])
scene.render()
```

37.1.12. Geometry.double_arrow

FÁBRICA `double_arrow(x1: float, y1: float, x2: float, y2: float, *, head_length: Optional[float] = None, head_width: Optional[float] = None, body_width: Optional[float] = None) -> Drawable`

Flecha con punta en ambos extremos, para rangos y relaciones bidireccionales. Las puntas se acortan para no solaparse; un `body_width` fino sirve para cotas. Para una cota completa, con líneas de extensión, usa `scene.mechanics.dimension`. Medidas no positivas lanzan `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`x1, y1, x2, y2`	`float`	obligatorio	Extremos.
`head_length / head_width / body_width`	`float | None`	`None`	Igual que en `arrow`: por defecto 0.18, 0.15 y 0.036.

Ejemplo:

```
span = scene.geometry.double_arrow(-3, 0, 3, 0).fill(WHITE).no_stroke()
fine = scene.geometry.double_arrow(-3, -1, 3, -1, body_width=0.012).fill(WHITE).no_stroke()
```

37.1.13. Geometry.curved_arrow

FÁBRICA `curved_arrow(x1: float, y1: float, x2: float, y2: float, angle: float, *, head_length: Optional[float] = None, head_width: Optional[float] = None, body_width: Optional[float] = None, max_head_ratio: Optional[float] = None) -> Drawable`

Flecha curva para ciclos y rotaciones. Comparte medidas y validación con `arrow`, así que flechas rectas y curvas pueden compartir estilo. La punta se acorta en arcos cortos y se estrecha en radios pequeños.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`x1, y1, x2, y2`	`float`	obligatorio	Inicio y final.
`angle`	`float`	obligatorio	Desviación en radianes; el signo elige

		el lado de la curva.
`head_length / head_width / body_width / max_head_ratio`	`float | None`	`None` Igual que en `arrow`; la proporción se mide sobre la longitud del arco.

Ejemplo:

```
from gaanim import GOLD, WHITE, Scene
scene = Scene(frame=(16, 9), background="#0f172a")
loop = scene.geometry.curved_arrow(-3, 0, 3, 0, 0.9).fill(WHITE)
bold = scene.geometry.curved_arrow(
    -3, -1.5, 3, -1.5, -0.9, head_length=0.3, head_width=0.24, body_width=0.06,
).fill(GOLD)
scene.play([loop.animate.create().duration(0.9), bold.animate.create().duration(0.9)])
scene.render()
```

37.1.14. Geometry.curved_arrow_arc

FÁBRICA `curved_arrow_arc(cx: float, cy: float, radius: float, start_angle: float, sweep_angle: float, *, head_length: Optional[float] = None, head_width:`

`Optional[float] = None, body_width: Optional[float] = None, max_head_ratio: Optional[float] = None) -> Drawable`

Flecha a lo largo de un arco circular con centro `(cx, cy)`, radio y ángulos en radianes. Acepta las mismas medidas que `curved_arrow`; `max_head_ratio` se mide sobre `radius * abs(sweep_angle)`.

Ejemplo:

```
turn = scene.geometry.curved_arrow_arc(0, 0, 2.5, 0.2, 1.8).fill(GOLD)
```

37.1.15. Geometry.arc

FÁBRICA `arc(cx: float, cy: float, radius: float, start_angle: float, sweep_angle: float) -> Drawable`

Arco de circunferencia con centro `(cx, cy)`; `start_angle` y `sweep_angle` en radianes. Nace sin relleno, como las demás líneas; `fill()` rellena el segmento entre la cuerda y el arco.

Ejemplo:

```
from gaanim import GOLD, Scene
scene = Scene(frame=(16, 9), background="#0f172a")
arc = scene.geometry.arc(0, 0, 0.75, 0.0, 2.0).no_fill().stroke(GOLD, 0.05)
scene.play([arc.animate.create().duration(0.8)])
scene.render()
```

37.1.16. Geometry.ellipse_arc

FÁBRICA `ellipse_arc(cx: float, cy: float, rx: float, ry: float, start_angle: float, sweep_angle: float) -> Drawable`

Arco de elipse, para el borde de un cilindro, un cono o una probeta en perspectiva: media elipse es `sweep_angle=math.pi`. Nace sin relleno y admite `tip`, recortes y `create()`, como `arc`. Radios no positivos o valores no finitos lanzan `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`cx, cy`	`float`	obligatorio	Centro.
`rx, ry`	`float`	obligatorio	Radios positivos en x y en y.
`start_angle, sweep_angle`	`float`	obligatorio	Radianes, en sentido antihorario desde +x, como en `arc`.

Ejemplo:

```
import math
from gaanim import GOLD, WHITE, Scene
scene = Scene(frame=(16, 9), background="#0f172a")
front = scene.geometry.ellipse_arc(0, -1.2, 1.6, 0.45, math.pi, math.pi).stroke(WHITE, 0.05)
back = scene.geometry.ellipse_arc(0, -1.2, 1.6, 0.45, 0, math.pi).stroke(WHITE, 0.03).opacity(0.4)
top = scene.geometry.ellipse_arc(0, 1.2, 1.6, 0.45, 0, 2 * math.pi).stroke(GOLD, 0.05)
left = scene.geometry.line(-1.6, -1.2, -1.6, 1.2).stroke(WHITE, 0.05)
right = scene.geometry.line(1.6, -1.2, 1.6, 1.2).stroke(WHITE, 0.05)
scene.play([shape.animate.create().duration(0.8) for shape in (front, back, top, left, right)])
scene.render()
```

37.1.17. Geometry.dashed_line

FÁBRICA `dashed_line(p1: Endpoint, p2: Endpoint, *, dash_length: float = 0.16, gap_length: float = 0.10) -> Drawable`

Línea discontinua para guías y aristas ocultas. Acepta los mismos extremos que `line`: dos puntos, objetos, `PointRef` o `AnchorPoint`, o cuatro coordenadas; entre objetos, los sigue en cada cuadro. Los guiones miden `dash_length` y los separa `gap_length` (0.16 y 0.10 por defecto). `create()` los dibuja uno tras otro desde el inicio.

Ejemplo:

```
guide = scene.geometry.dashed_line(-1.75, 0, 1.75, 0, dash_length=0.15,
gap_length=0.1).stroke(WHITE, 0.025)
dot = scene.geometry.dot(0.1).move_to(-1, 1.5)
link = scene.geometry.dashed_line(dot, (1.75, 0)).stroke(GOLD, 0.02)
scene.play([guide.animate.create().duration(0.8), dot.animate.shift_by(2, 0)])
```

37.1.18. Geometry.connector

FÁBRICA `connector(start: Endpoint, end: Endpoint, *, via: Optional[Sequence[Endpoint]] = None, head_length: float = 0.18, head_width: float = 0.15, body_width: float = 0.036, max_head_ratio: Optional[float] = None) -> Drawable`

Flecha rellena y reactiva que sigue a sus extremos durante la animación y el seek. Usa `via` para los codos: no esquiva obstáculos. La punta se limita al último segmento no nulo. Referencias de otra escena lanzan `ValueError`. Anima el conector completo con `create()`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`start / end`	`Endpoint`	obligatorio	Cola y punta, fijas o reactivas.
`via`	`Sequence[Endpoint] | None`	`None`	Puntos intermedios en orden.
`head_length / head_width / body_width`	`float`	`0.18 / 0.15 / 0.036`	Medidas positivas en unidades de escena.

Ejemplo:

```
from gaanim import Anchor, Scene
scene = Scene(frame=(16, 9))
left = scene.geometry.rect(2, 1).move_to(-3, 0)
right = scene.geometry.rect(2, 1).move_to(3, 0)
link = scene.geometry.connector(left.anchor_point(Anchor.RIGHT), right.anchor_point(Anchor.LEFT))
scene.play(link.animate.create())
scene.play(right.animate.shift_by(0, 1))
scene.render()
```

37.1.19. Polígonos y símbolos

Polígonos arbitrarios y regulares, sectores y marcas habituales en diagramas y demostraciones.

37.1.20. Geometry.polygon

FÁBRICA `polygon(points: Sequence[tuple[float, float]]) -> Drawable`

Polígono cerrado con vértices arbitrarios.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`points`	`Sequence[tuple[float, float]]`	obligatorio	Al menos tres puntos finitos; el polígono se cierra solo.

Ejemplo:

```
from gaanim import BLUE, WHITE, Scene
scene = Scene(frame=(16, 9), background="#0f172a")
tri = scene.geometry.polygon([(0, 0.875), (-0.81, -0.625), (0.81,
-0.625)]).fill(BLUE).stroke(WHITE, 0.025)
scene.play([tri.animate.grow_from_center().duration(0.8)])
scene.render()
```

37.1.21. Geometry.regular_polygon

FÁBRICA `regular_polygon(sides: int, radius: float) -> Drawable`

Polígono regular de `sides` lados (al menos 3) inscrito en un círculo de radio `radius`. Con 6 lados obtienes un hexágono.

Ejemplo:

```
from gaanim import BLUE, WHITE, Scene
scene = Scene(frame=(16, 9), background="#0f172a")
hexa = scene.geometry.regular_polygon(6, 0.75).fill(BLUE).stroke(WHITE, 0.025)
```

```
scene.play([hexa.animate.spin_in_from_nothing().duration(0.9)])
scene.render()
```

37.1.22. Geometry.star

FÁBRICA `star(points: int, outer_radius: float, inner_radius: float) -> Drawable`

Estrella de `points` puntas (al menos 2) con radios exterior e interior.

Ejemplo:

```
from gaanim import GOLD, Scene
scene = Scene(frame=(16, 9), background="#0f172a")
star = scene.geometry.star(5, 0.875, 0.4).fill(GOLD).move_to(0, 0)
scene.play([star.animate.spin_in_from_nothing().duration(1.0)])
scene.render()
```

37.1.23. Geometry.sector

FÁBRICA `sector(cx: float, cy: float, radius: float, start_angle: float, sweep_angle: float) -> Drawable`

Porción de círculo con centro `(cx, cy)`; ángulos en radianes. Sirve para gráficos de tarta y cuñas de progreso.

Ejemplo:

```
slice_ = scene.geometry.sector(0, 0, 0.875, 0.0, 2.0).fill(GOLD)
```

37.1.24. Geometry.annulus

FÁBRICA `annulus(outer_radius: float, inner_radius: float) -> Drawable`

Anillo entre `inner_radius` (mayor que cero) y `outer_radius` (mayor que el interior). Útil para donuts, halos y dianas.

Ejemplo:

```
ring = scene.geometry.annulus(0.75, 0.425).fill(BLUE).stroke(WHITE, 0.025)
```

37.1.25. Geometry.brace

FÁBRICA `brace(x1: float, y1: float, x2: float, y2: float, height: float) -> Drawable`

Llave entre `(x1, y1)` y `(x2, y2)` con profundidad `height` distinta de cero; el signo elige el lado. Se dibuja como una llave tipográfica: brazos rectos, rizos del tamaño de su profundidad y una punta, con un grosor que se afina hacia los extremos (16 % de `height` en los brazos, al menos 0.02). Es una figura rellena y sin trazo: `fill` le da color y un tema la pinta como la familia `glyph`, con el color del texto. Para una llave bajo una parte de un texto, usa `selection.animate.brace(...)` (ver Texto).

Ejemplo:

```
from gaanim import WHITE, Scene
scene = Scene(frame=(16, 9), background="#0f172a")
brace = scene.geometry.brace(-1, -0.25, 1, -0.25, 0.3).fill(WHITE)
label = scene.text("intervalo").move_to(0, -0.69)
scene.play([brace.animate.create().duration(0.7), label.animate.fade_in().duration(0.4)])
scene.render()
```

37.1.26. Geometry.checkmark

FÁBRICA `checkmark(size: float) -> Drawable`

Marca de verificación de tamaño `size`. Combínala con `cross` para respuestas correctas e incorrectas.

Ejemplo:

```
ok = scene.geometry.checkmark(0.425).fill(GREEN)
```

37.1.27. Geometry.cross

FÁBRICA `cross(size: float) -> Drawable`

Aspa de tamaño `size` para errores, rechazos o botones de cierre. No tiene relleno y toma el estilo de línea del tema, como `line`.

Ejemplo:

```
no = scene.geometry.cross(0.425).stroke(WHITE, 0.05)
```

37.1.28. Geometry.right_angle

FÁBRICA `right_angle(arm_length: float) -> Drawable`

Marca de ángulo recto con brazos de `arm_length`. Colócala en la esquina de un triángulo. No tiene relleno y toma el estilo de línea del tema, como `line`.

Ejemplo:

```
corner = scene.geometry.right_angle(0.3).stroke(WHITE, 0.04).move_to(0, 0)
```

37.1.29. Trayectorias

Caminos libres: polilíneas, curvas de Bézier y comandos de cursor.

37.1.30. Geometry.path

FÁBRICA `path(definition: Sequence[CurvePoint] | Sequence[CurveCommand]) -> Drawable`

Entrada compacta para geometría propia: una lista de puntos crea una polilínea y una lista de comandos, una curva (como `curve`).

Ejemplo:

```
from gaanim import WHITE, Scene
scene = Scene(frame=(16, 9), background="#0f172a")
rail = scene.geometry.path([(-1.75, 0), (0, 0.75), (1.75, 0)]).no_fill().stroke(WHITE, 0.05)
scene.play([rail.animate.create().duration(1.0)])
scene.render()
```

37.1.31. Geometry.polyline

FÁBRICA `polyline(points: Sequence[tuple[ScalarSource, ScalarSource]], *, closed: bool = False) -> Drawable`

Polilínea que une los puntos en orden. Con coordenadas fijas, abierta o cerrada, `points` cambia sus vértices. Una coordenada puede ser un `Parameter`, un `Computed` o `scene.time`: entonces la polilínea se redibuja en cada fotograma con sus valores, lo que da forma a cualquier línea o contorno a partir de señales (la traza de un audio, un gráfico en vivo). Un punto cuyo valor no se puede leer queda fuera de ese fotograma.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`points`	`Sequence[tuple[ScalarSource, ScalarSource]]`	obligatorio	Puntos en orden, en coordenadas de la escena.
`closed`	`bool`	`False`	Une el último punto con el primero.

Ejemplo:

```
zig = scene.geometry.polyline([(-1.25, -0.375), (0, 0.375), (1.25,
-0.375)]).no_fill().stroke(GOLD, 0.04)
wave = scene.geometry.polyline(
    [(x / 2, computed(lambda t, x=x: 0.5 * math.sin(x / 2 + 3 * t), inputs=[scene.time])) for x in
range(-8, 9)]
).no_fill().stroke(GOLD, 0.04)
```

37.1.32. Geometry.metaballs

FÁBRICA `metaballs(balls: Sequence[Drawable], *, threshold: float = 1.0, smoothness: float = 0.4) -> Drawable`

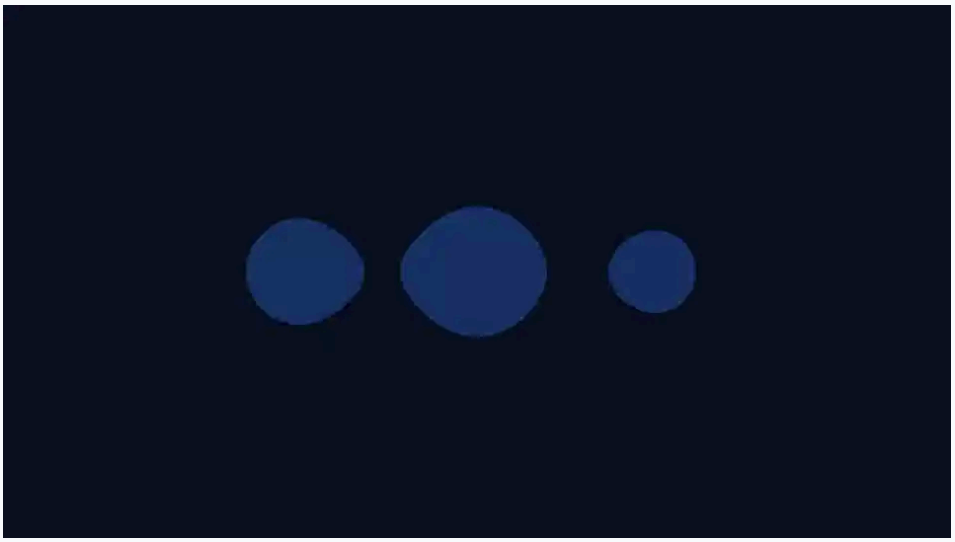
Contorno de varias bolas fundidas como gotas de líquido (efecto *gooey*). Se recalcula en cada fotograma a partir de dónde están las bolas y de su tamaño, así que animarlas anima la masa; las bolas dejan de dibujarse (opacidad 0). El resultado es un trazado vectorial normal en coordenadas de la escena: nítido a cualquier tamaño, con relleno, trazo y efectos como cualquier otro. Una lista vacía, una bola de otra escena, un `threshold` no positivo o un `smoothness` negativo lanzan `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`balls`	`Sequence[Drawable]`	obligatorio	Los objetos que forman la masa; cada uno cuenta como un círculo con el centro de su caja y la mitad de su lado mayor.
`threshold`	`float`	`1.0`	Divide cada radio: por encima de 1 adelgaza las bolas.
`smoothness`	`float`	`0.4`	Distancia en unidades de escena a la que dos bolas se unen con un cuello; 0 es una unión simple.

Ejemplo:

```python
from gaanim import BLUE, Scene
scene = Scene(frame=(16, 9), background="#0b1020")
drops = [scene.geometry.circle(r).move_to(x, 0) for x, r in ((-3, 0.9), (0, 1.1), (3, 0.7))]
blob = scene.geometry.metaballs(drops, smoothness=0.9).fill(BLUE)
scene.play([drops[0].animate.move_to(-1.2, 0.3).duration(1.5), drops[2].animate.move_to(1.3,
-0.3).duration(1.5)])
scene.render()
```



37.1.33. Geometry.bezier

FÁBRICA `bezier(start: tuple[float, float], controls: Sequence[tuple[float, float]], end: tuple[float, float]) -> Drawable`

Curva de Bézier. Se conserva como Bézier, lo que la hace útil con la geometría reactiva sobre curvas.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`start / end`	`tuple[float, float]`	obligatorio	Extremos.
`controls`	`Sequence[tuple[float, float]]`	obligatorio	Uno para una curva cuadrática, dos para una cúbica.

Ejemplo:

```python
from gaanim import WHITE, Scene
scene = Scene(frame=(16, 9), background="#0f172a")
curve = scene.geometry.bezier((-1.75, 0), [(-0.625, 1.125), (0.625, -1.125)], (1.75,
0)).no_fill().stroke(WHITE, 0.04)
scene.play([curve.animate.create().duration(1.0)])
scene.render()
```

37.1.34. Geometry.curve

FÁBRICA `curve(commands: Sequence[CurveCommand]) -> Drawable`

Curva compuesta con comandos de cursor, al estilo de Typst. Los controles de `quad` y `cubic` aceptan un punto, `None` (el control cae sobre su extremo) o `"auto"`: en un `quad` refleja respecto al cursor el control del `quad` anterior, y en el primer control de un `cubic`, el segundo control del `cubic` anterior. Tras otro tipo de tramo, o como segundo control de un `cubic`, `"auto"` equivale a `None`. `close` y `close_smooth` llevan una lista vacía.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`commands`	`Sequence[CurveCommand]`	obligatorio	Pares `(nombre, puntos)`: `move`, `line`, `quad`, `cubic`, `close` y `close_smooth`. Añade `_rel` a un comando de dibujo para usar puntos relativos al cursor.

Ejemplo:

```
shape = scene.geometry.curve([
    ("move", [(0, 0)]),
    ("cubic", [(0.625, 0.75), (1.375, -0.75), (2, 0)]),
    ("line_rel", [(0, -1)]),
    ("close", []),
]).no_fill().stroke(WHITE, 0.04)
```

37.1.35. Grupos y marcos

Agrupa objetos para moverlos como uno o dibuja un marco vivo a su alrededor.

37.1.36. Geometry.group

FÁBRICA `group(members: Sequence[Drawable]) -> Drawable`

Agrupa objetos para moverlos, rotarlos y escalarlos como uno, conservando las coordenadas locales de cada miembro (también las de sus updaters). Los miembros ya visibles no se ocultan al unirse a un grupo que contiene un trazo o una fuerza diferidos. `write`, `create` y los fundidos del grupo son entradas explícitas de todo el subárbol. Para distribuir miembros usa Layout.

Ejemplo:

```
from gaanim import BLUE, Direction, Scene
scene = Scene(frame=(16, 9), background="#0f172a")
row = scene.geometry.group([scene.geometry.dot(0.125).fill(BLUE),
scene.text("Etiqueta").move_to(0.9, 0)]).move_to(0, 0)
scene.play([row.animate.fade_in_from(Direction.DOWN, distance=0.3).duration(0.6)])
scene.render()
```

37.1.37. Geometry.surrounding_rect

FÁBRICA `surrounding_rect(targets: Drawable | TextSelection | Sequence[Drawable | TextSelection], *, padding: Padding = 0.12, corner_radius: float = 0.08) -> SurroundingRect`

Marco vivo alrededor de los límites de sus objetivos en el mundo: los sigue al moverse, escalarse, rotar o reordenarse en un layout. Usa el trazo del tema y ningún relleno. Su posición y geometría pertenecen al vínculo: anima los objetivos o cambia de objetivo con `retarget`. Objetivos vacíos, de otra escena o medidas inválidas lanzan `TypeError` o `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`targets`	`Drawable | TextSelection | Sequence`	obligatorio	Uno o varios objetos, partes de

			texto o partes de ecuación.
`padding`	`float | (v, h) | (t, r, b, l)`	`0.12`	Margen no negativo alrededor de la unión.
`corner_radius`	`float`	`0.08`	Radio no negativo, limitado al tamaño del marco.

Ejemplo:

```
from gaanim import GOLD, Scene, part
scene = Scene(frame=(16, 9), background="#0f172a")
eq = scene.text.equation("E =", part("mass", "m"), part("light", "c^2"))
frame = scene.geometry.surrounding_rect(eq["mass"]).stroke(GOLD, 0.04)
scene.play([eq.animate.fade_in(), frame.animate.create()])
scene.play([frame.retarget(eq["light"]).duration(0.8)])
scene.render()
```

37.1.38. SurroundingRect.retarget

MÉTODO `retarget(targets: Drawable | TextSelection | Sequence[Drawable | TextSelection]) -> Anim`

Devuelve un `Anim` que interpola los cuatro bordes del marco hacia nuevos objetivos y después los sigue. Origen y destino pueden seguir moviéndose durante la transición; los seeks reproducen la misma geometría. Configura el tiempo con `duration` y `easing`.

Ejemplo:

```
equation = scene.text("$", part("lhs", "x + 3"), " = ", part("result", "7"), "$")
frame = scene.geometry.surrounding_rect(equation["lhs"])
scene.play([frame.retarget(equation["result"]).duration(0.9).easing(Easing.spring(stiffness=90,
damping=12))])
```

37.1.39. Repetidores y distribuciones

Copias de una figura colocadas por una transformación acumulada o por una distribución, como el Repeater de After Effects o el Duplicator de Cavalry. El resultado es un grupo cuyos miembros son las copias, en orden: `connect` las recorre y el estilo del grupo llega a todas. La figura original es la copia 0; los grupos y textos se copian con sus miembros. `count` y `animate.count` muestran solo las primeras copias y funden la siguiente con la parte fraccionaria, para construir el patrón copia a copia.

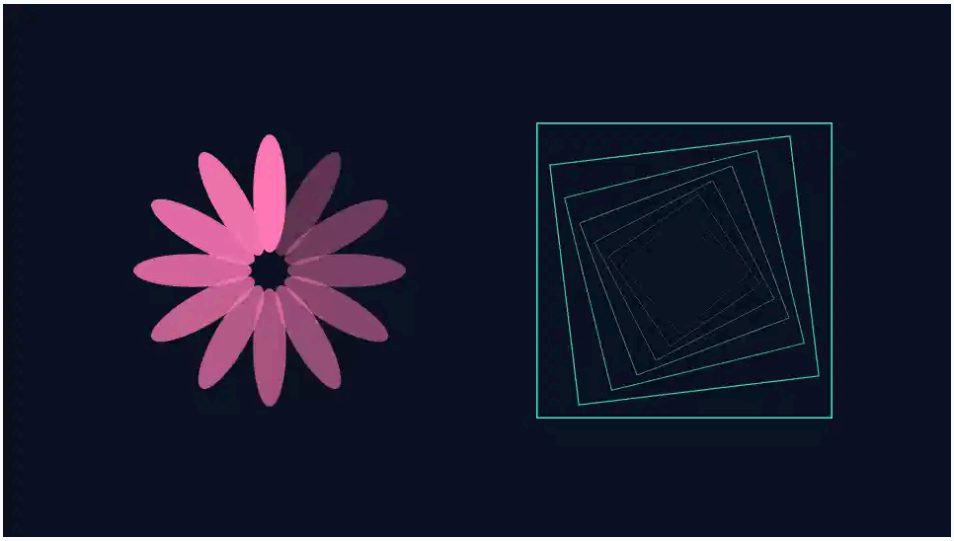
37.1.40. Geometry.repeat

FÁBRICA `repeat(shape, count, *, rotate=0.0, scale=1.0, offset=(0.0, 0.0), opacity=(1.0, 1.0), about=None) -> Drawable`

`count` copias (1 a 10 000): la copia `i` se desplaza `i * offset`, gira `i * rotate` radianes y escala `scale ** i`, alrededor de `about` si se da (así los pétalos orbitan el centro de la flor) o de su propio pivote. Su opacidad va de `opacity[0]` en la primera a `opacity[1]` en la última. Valores no finitos, una escala no positiva, opacidades fuera de [0, 1] o una figura de otra escena lanzan `ValueError`.

Ejemplo:

```
import math
from gaanim import CYAN, PINK, Scene
scene = Scene(frame=(16, 9), background="#0d1322")
petal = scene.geometry.ellipse(0.28, 1.0).fill(PINK).move_to(-3.5, 1.3)
flower = scene.geometry.repeat(petal, 12, rotate=math.tau / 12, about=(-3.5, 0), opacity=(1.0,
0.35))
square = scene.geometry.square(5.0).no_fill().stroke(CYAN, 0.05).move_to(3.5, 0)
tunnel = scene.geometry.repeat(square, 8, rotate=0.12, scale=0.82, opacity=(1.0, 0.2))
scene.play([flower.animate.rotate_by(1.0).duration(1.5),
tunnel.animate.rotate_by(-0.6).duration(1.5)])
scene.render()
```



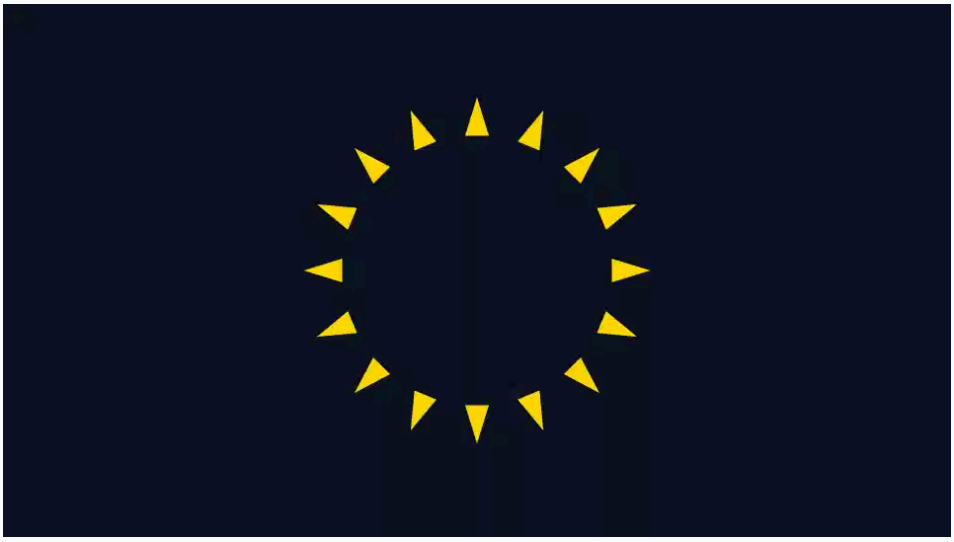
37.1.41. Geometry.duplicate

FÁBRICA `duplicate(shape, distribution) -> Drawable`

Una copia de `shape` centrada en cada punto de `distribution`; la figura original va al primero. Con una distribución orientada, cada copia gira para seguirla. Una figura de otra escena lanza `ValueError`.

Ejemplo:

```
from gaanim import GOLD, Distribution, Scene
scene = Scene(frame=(16, 9), background="#0d1322")
arrow = scene.geometry.polygon([(-0.3, -0.2), (0.35, 0.0), (-0.3, 0.2)]).fill(GOLD)
ring = scene.geometry.duplicate(arrow, Distribution.circle(16, 2.6, orient=True))
scene.play([ring.animate.rotate_by(1.2).duration(1.5)])
scene.render()
```



37.1.42. Distribution

CLASE `Distribution.grid(columns, rows, spacing=0.5, *, center=(0, 0)) · circle(count, radius=2.0, *, center=(0, 0), start=0.0, orient=False) · along(points, count, *, orient=False) · random(count, bounds=(-6, -3, 6, 3), *, seed=0) · phyllotaxis(count, spacing=0.2, *, center=(0, 0))`

Dónde van las copias de `Geometry.duplicate`. `grid`: `columns` × `rows` puntos separados `spacing` (un número o `(x, y)`), fila a fila desde arriba a la izquierda. `circle`: puntos en una circunferencia desde `start` radianes en sentido antihorario; con `orient` cada copia mira hacia fuera. `along`: puntos repartidos por longitud a lo largo de una polilínea, extremos incluidos; con `orient` siguen su dirección. `random`: puntos uniformes dentro de `(xmin, ymin, xmax, ymax)`, iguales para la misma `seed`. `phyllotaxis`: espiral de girasol, el punto `i` a `spacing * sqrt(i)` girado por el ángulo áureo. `points` devuelve las posiciones. Menos de 1 o más de 10 000 puntos, o valores no finitos, lanzan `ValueError`.

Ejemplo:

```
grid = Distribution.grid(12, 7, 0.5)
seeds = scene.geometry.duplicate(scene.geometry.dot(0.04), Distribution.phyllotaxis(300, 0.15))
print(len(grid), grid.points[0])

84 (-2.75, 1.5)
```

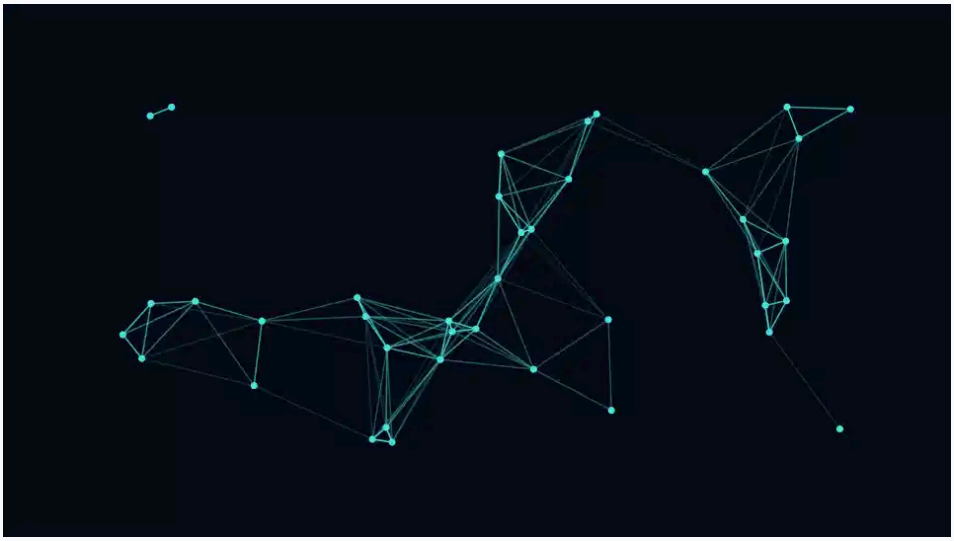
37.1.43. Geometry.connect

FÁBRICA `connect(points, max_distance=1.5, *, mode="range", neighbors=2, fade_by_distance=True) -> Drawable`

Líneas vivas entre puntos, como el Connect Shape de Cavalry (efecto *plexus*). `points` es un grupo (sus miembros) o una lista de objetos; cada punto es el centro de sus límites y las líneas los siguen al moverse, en cada fotograma y búsqueda. `mode` elige los pares no más largos que `max_distance`: `"range"` une todos, `"nearest"` cada punto con sus `neighbors` más cercanos y `"sequential"` cada punto con el siguiente. Los puntos ocultos se saltan y los pares salen en un orden fijo. Con `fade_by_distance` las conexiones largas son más tenues. Por defecto son líneas blancas finas (o el estilo de línea del tema); cámbialas con `stroke`. Menos de dos puntos, un `max_distance` no positivo o `neighbors` menor que 1 lanzan `ValueError`.

Ejemplo:

```
from gaanim import CYAN, Distribution, Scene
scene = Scene(frame=(16, 9), background="#060b16")
cloud = scene.geometry.duplicate(scene.geometry.dot(0.06).fill(CYAN), Distribution.random(40,
(-6.5, -3.0, 6.5, 3.0), seed=7))
links = scene.geometry.connect(cloud, max_distance=2.2).stroke(CYAN, 0.025)
scene.play([cloud.animate.rotate_by(0.6).duration(2.0)])
scene.render()
```



37.1.44. Operaciones booleanas

Unión, intersección, diferencia y diferencia simétrica de áreas vectoriales cerradas.

Los operandos se conservan como objetos independientes y el resultado copia el estilo que tiene el primero al crearlo (relleno, trazo, opacidad y efectos); los cambios de estilo posteriores de los operandos no lo cambian, ni siquiera con `live=True`. Con `live=True`, el resultado se reconstruye cuando cambian los caminos o las transformaciones de sus fuentes. `tolerance` es el error máximo, en unidades de escena, al aproximar las curvas; el valor predeterminado (`0.0025`, unos 0.3 px a 1080p) mantiene suaves los círculos. Súbelo solo para acelerar resultados `live=True` muy complejos. Debe ser finita y positiva. `rule` acepta `"nonzero"` o `"evenodd"`; operandos de otra escena o valores inválidos lanzan `ValueError`.

37.1.45. Geometry.union

FÁBRICA `union(*operands: Drawable, live: bool = False, tolerance: float = 0.0025, rule: Literal["nonzero", "evenodd"] = "nonzero") -> Drawable`

Unión de al menos dos áreas.

Ejemplo:

```
from gaanim import BLUE, Scene
scene = Scene(frame=(16, 9), background="#0f172a")
left = scene.geometry.circle(1).move_to(-0.5, 0)
right = scene.geometry.circle(1).move_to(0.5, 0)
merged = scene.geometry.union(left, right).fill(BLUE)
left.opacity(0)
right.opacity(0)
scene.play([merged.animate.create().duration(1.0)])
scene.render()
```

37.1.46. Geometry.intersection

FÁBRICA `intersection(*operands: Drawable, live: bool = False, tolerance: float = 0.0025, rule: Literal["nonzero", "evenodd"] = "nonzero") -> Drawable`

Área común de al menos dos objetos.

Ejemplo:

```
lens = scene.geometry.intersection(left, right).fill(GOLD)
```

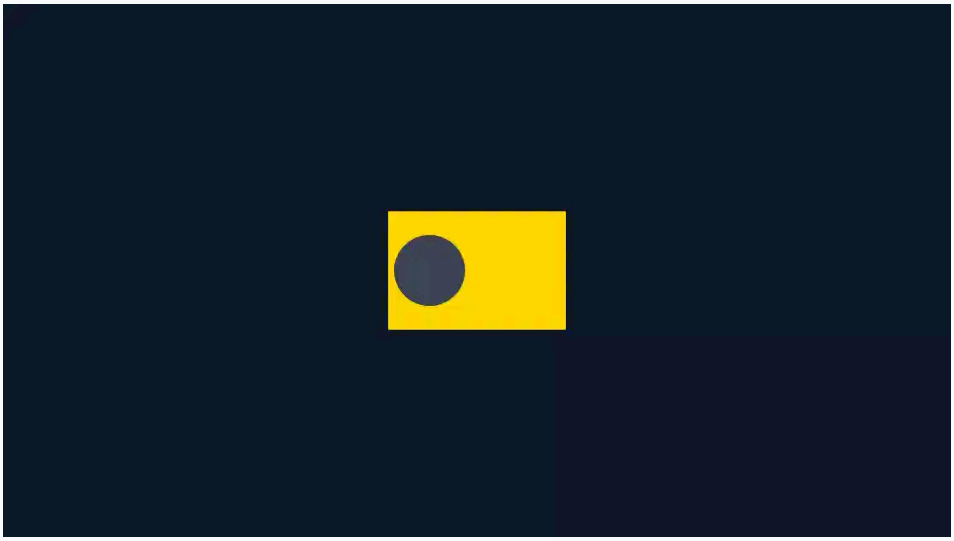
37.1.47. Geometry.difference

FÁBRICA `difference(subject: Drawable, *clips: Drawable, live: bool = False, tolerance: float = 0.0025, rule: Literal["nonzero", "evenodd"] = "nonzero") -> Drawable`

Resta cada recorte del sujeto, de izquierda a derecha.

Ejemplo:

```
from gaanim import GOLD, WHITE, Scene
scene = Scene(frame=(16, 9), background="#0f172a")
plate = scene.geometry.rect(3, 2).fill(GOLD)
hole = scene.geometry.circle(0.6).fill(WHITE).move_to(-0.8, 0)
cut = scene.geometry.difference(plate, hole, live=True)
plate.opacity(0)
hole.opacity(0.2)
scene.play([hole.animate.shift_by(1.6, 0).duration(1.2)])
scene.render()
```



37.1.48. Geometry.xor

FÁBRICA `xor(*operands: Drawable, live: bool = False, tolerance: float = 0.0025, rule: Literal["nonzero", "evenodd"] = "nonzero") -> Drawable`

Diferencia simétrica: las áreas cubiertas por un número impar de operandos.

Ejemplo:

```
either = scene.geometry.xor(left, right, rule="evenodd")
```

37.1.49. Relleno por nivel

Llena una silueta hasta un porcentaje, como un depósito de agua o una barra de progreso con forma.

37.1.50. Geometry.fill_level

FÁBRICA `fill_level(mask: Drawable, paint: Paint, level: Optional[ScalarSource] = None, *, direction: Literal["up", "down", "left", "right"] = "up", keep_outline: bool = True) -> Drawable`

Genera un interior vectorial que se intersecta con la silueta en cada fotograma. Oculta la máscara original si su relleno no debe tapar el nivel. Con una fuente reactiva, las muestras finitas se limitan a `[0, 1]`; una muestra no finita genera un diagnóstico y usa el valor de respaldo. Anima el nivel con `animate.fill_level(...)`; los saltos en la línea de tiempo restauran vínculos, valores y cortes.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`mask`	`Drawable`	obligatorio	Silueta que recorta

			el relleno; conserva su visibilidad propia.
`paint`	`Paint`	obligatorio	Color o pincel del interior.
`level`	`float | Parameter | Variable | Computed | TimeInput | None`	`None`	Nivel en `[0, 1]`; `None` empieza vacío.
`direction`	`str`	`"up"`	Hacia dónde crece el relleno: `up` (desde abajo), `down`, `left` (desde la derecha) o `right` (desde la izquierda).
`keep_outline`	`bool`	`True`	Añade una copia reactiva que solo usa el trazo de

la máscara.

Ejemplo:

```
drop = scene.media.svg("drop.svg").scale_to(2.5).no_fill().stroke("#dbeafe", 0.05).opacity(0)
water = scene.geometry.fill_level(drop, "#38bdf8", 0.0)
scene.play([water.animate.fill_level(0.72).duration(1.4)])
```

Con una fuente reactiva, el nivel sigue a un parámetro que también puede mostrarse:

```
amount = scene.viz.parameter(0)
fraction = computed(lambda value: value / 100, inputs=[amount])
water = scene.geometry.fill_level(drop, "#38bdf8", fraction)
label = scene.viz.readout(amount, format=".1f", suffix="%")
scene.play(amount.animate.set(72), duration=1.4)
```

37.1.51. Drawable.set_fill_level

MÉTODO `set_fill_level(level: ScalarSource) -> Drawable`

Cambia el nivel de un objeto creado con `fill_level` desde el cursor. Otra fuente reactiva reemplaza el vínculo; un número en `[0, 1]` lo termina con un corte reversible. Mientras esté vinculado, anima la fuente: `animate.fill_level(...)` se rechaza. Valores fijos inválidos, fuentes de otra escena u otros tipos de objeto lanzan `ValueError`.

Ejemplo:

```
water.set_fill_level(0.4)
```

37.1.52. Geometría reactiva

Relaciones que se recalculan en el mismo fotograma: «este extremo es la esquina del rectángulo» o «este punto está al 35 % de la curva».

En lugar de recalcular una línea desde Python, declaras la relación y el motor regenera solo la geometría afectada después de aplicar las transformaciones de sus fuentes. La relación se evalúa para cualquier instante, así que el seek es exacto. Los objetos reactivos suelen empezar ocultos: anima su entrada (`fade_in`, `create` o `write`) junto con lo que los conduce. Las magnitudes reactivas (`Parameter`, `Variable`, `Computed`) están en Visualización y la guía Reactividad enseña a combinarlas.

37.1.52.1. Puntos de referencia

Puntos no dibujados que se aceptan en cualquier lugar donde se admite un `Endpoint`. Sus coordenadas aceptan números o fuentes reactivas.

37.1.53. Geometry.point_ref

FÁBRICA `point_ref(x: _ReactiveScalar, y: _ReactiveScalar) -> PointRef`

Punto de escena cuyas coordenadas pueden ser fuentes reactivas.

Ejemplo:

```
x = scene.viz.parameter(-3)
probe = scene.geometry.point_ref(x, 1)
marker = scene.geometry.dot(0.1).fill(GOLD).follow(probe)
scene.play([marker.animate.fade_in(), x.animate.set(3).duration(1.5)])
```

37.1.54. Geometry.offset_point

FÁBRICA `offset_point(origin: Endpoint, dx: _ReactiveScalar, dy: _ReactiveScalar) -> PointRef`

Punto desplazado `(dx, dy)` desde un origen móvil, en el espacio de la escena.

Ejemplo:

```
body = scene.geometry.circle(0.4).move_to(-2, 0)
above = scene.geometry.offset_point(body, 0, 0.8)
tag = scene.text("m", role="label").follow(above)
```

37.1.55. Geometry.point_between

FÁBRICA `point_between(from_: Endpoint, to: Endpoint, *, alpha: _ReactiveScalar = 0.5, offset: tuple[float, float] = (0.0, 0.0)) -> PointRef`

Punto afín entre dos extremos (`alpha=0` en el primero, `1` en el segundo) más un desplazamiento en el mundo. `alpha` puede ser un número, un `Parameter`, una `Variable` o un `Computed`.

Ejemplo:

```
a = scene.geometry.dot(0.1).move_to(-3, 0)
b = scene.geometry.dot(0.1).move_to(3, 1)
middle = scene.geometry.point_between(a, b, alpha=0.5, offset=(0, 0.3))
label = scene.text("M", role="label").follow(middle)
```

37.1.56. Geometry.polar_point

FÁBRICA `polar_point(origin: Endpoint, radius: _ReactiveScalar, angle: _ReactiveScalar) -> PointRef`

Punto a distancia `radius` y ángulo `angle` (radianes) de un origen; ambos pueden ser reactivos.

Ejemplo:

```
theta = scene.viz.parameter(0.0)
tip = scene.geometry.polar_point((0, 0), 1.5, theta)
hand = scene.geometry.tracking_line((0, 0), tip).stroke(WHITE, 0.05)
scene.play([hand.animate.create(), theta.animate.set(3.14).duration(1.5)])
```

37.1.56.1. Líneas y marcas que siguen a otros objetos

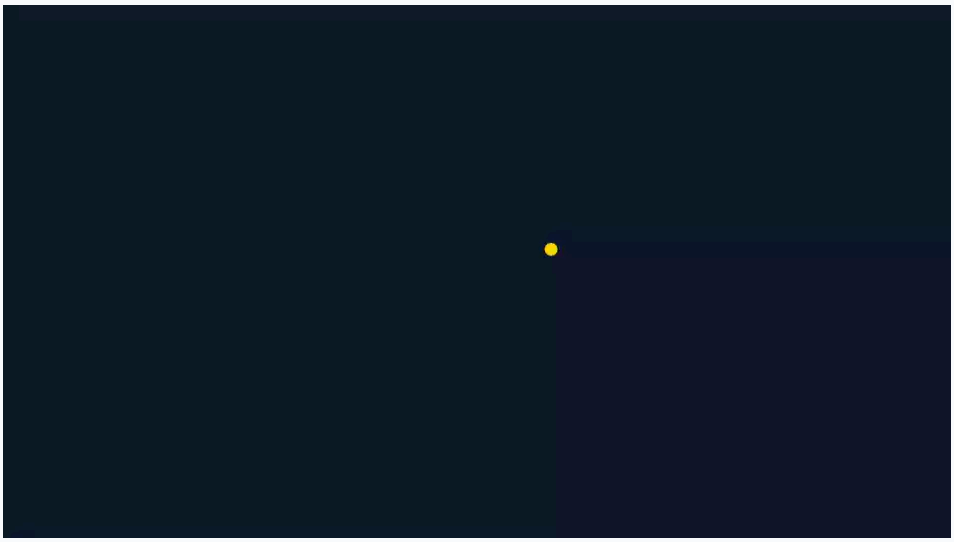
37.1.57. Geometry.tracking_line

FÁBRICA `tracking_line(from_: Endpoint, to: Endpoint) -> Drawable`

Línea oculta cuyos extremos siguen a sus fuentes en el mismo fotograma y que conserva el progreso de `create` o `write` activos. Revélala con una animación de entrada.

Ejemplo:

```
from gaanim import GOLD, WHITE, Scene
scene = Scene(frame=(16, 9), background="#0f172a")
tip = scene.geometry.dot(0.11).fill(GOLD).move_to(1.25, 0.375)
rod = scene.geometry.tracking_line((-1.25, -0.375), tip).no_fill().stroke(WHITE, 0.05)
scene.play([rod.animate.create().duration(0.8), tip.animate.shift_by(-0.5, 1).duration(0.8)])
scene.play([rod.animate.write().duration(0.8), tip.animate.shift_by(1, -0.625).duration(0.8)])
scene.render()
```



37.1.58. Geometry.point_on_curve

FÁBRICA `point_on_curve(curve: Drawable, tracker: _ReactiveScalar) -> Drawable`

Punto oculto que recorre la curva por longitud de arco, sin callbacks de Python durante la reproducción. Lee la curva tal como se dibuja: si se mueve, gira, escala o anima, o está dentro de un grupo como unos ejes, el punto sigue sobre ella. Lo mismo vale para `tangent_on_curve`, `normal_on_curve` y `curvature_on_curve`.

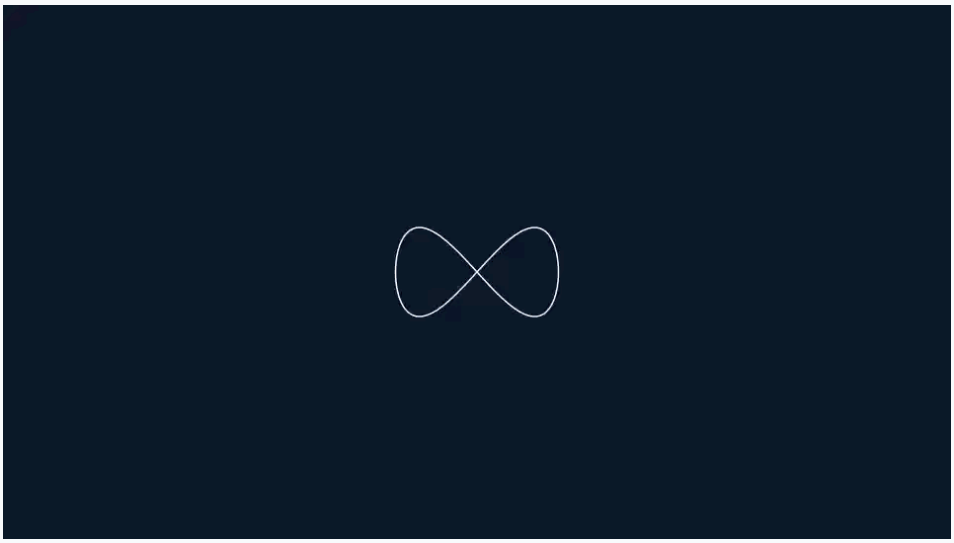
Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`curve`	`Drawable`	obligatorio	Polilínea, Bézier, arco, círculo o polígono.
`tracker`	`float | Parameter | Variable | Computed`	obligatorio	Posición de 0 a 1 por

			longitud de arco; se limita a ese rango.

Ejemplo:

```
from gaanim import GOLD, WHITE, Scene
from math import cos, sin, pi
scene = Scene(frame=(16, 9), background="#0f172a")
t = scene.viz.parameter(0.0)
curve = scene.geometry.polyline([(1.375*cos(u), 0.75*sin(2*u)) for u in (2*pi*i/240 for i in
range(241))]).no_fill().stroke(WHITE, 0.025)
dot = scene.geometry.point_on_curve(curve, t).fill(GOLD)
scene.play([dot.animate.fade_in().duration(0.3), t.animate.set(1.0).duration(1.6)])
scene.render()
```



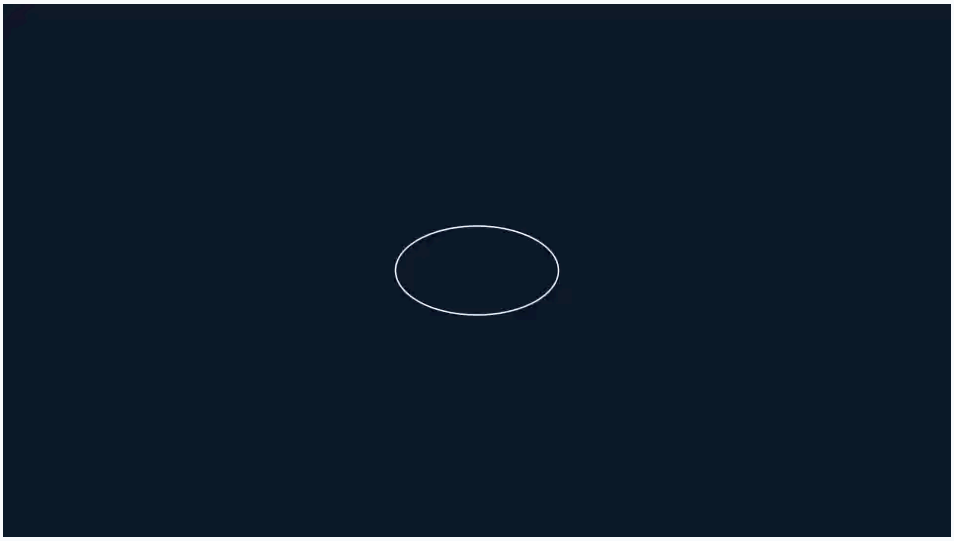
37.1.59. Geometry.tangent_on_curve

FÁBRICA `tangent_on_curve(curve: Drawable, tracker: _ReactiveScalar, length: float = 0.8) -> Drawable`

Segmento oculto de longitud `length`, centrado en el punto de la curva y orientado según la tangente. Usa el mismo muestreo por longitud de arco que `point_on_curve`.

Ejemplo:

```
from gaanim import GOLD, WHITE, Scene
from math import cos, sin, pi
scene = Scene(frame=(16, 9), background="#0f172a")
t = scene.viz.parameter(0.35)
curve = scene.geometry.polyline([(1.375*cos(u), 0.75*sin(u)) for u in (2*pi*i/240 for i in
range(241))]).no_fill().stroke(WHITE, 0.025)
tangent = scene.geometry.tangent_on_curve(curve, t, length=0.875).stroke(GOLD, 0.04)
scene.play([tangent.animate.fade_in().duration(0.3), t.animate.set(0.9).duration(1.4)])
scene.render()
```



37.1.60. Geometry.normal_on_curve

FÁBRICA `normal_on_curve(curve: Drawable, tracker: _ReactiveScalar, length: float = 0.8) -> Drawable`

Como `tangent_on_curve`, girado 90° en sentido antihorario.

Ejemplo:

```
t = scene.viz.parameter(0.2)
curve = scene.geometry.polyline([(math.cos(u / 40), math.sin(u / 40)) for u in range(252)])
normal = scene.geometry.normal_on_curve(curve, t, length=0.6).stroke(RED, 0.04)
```

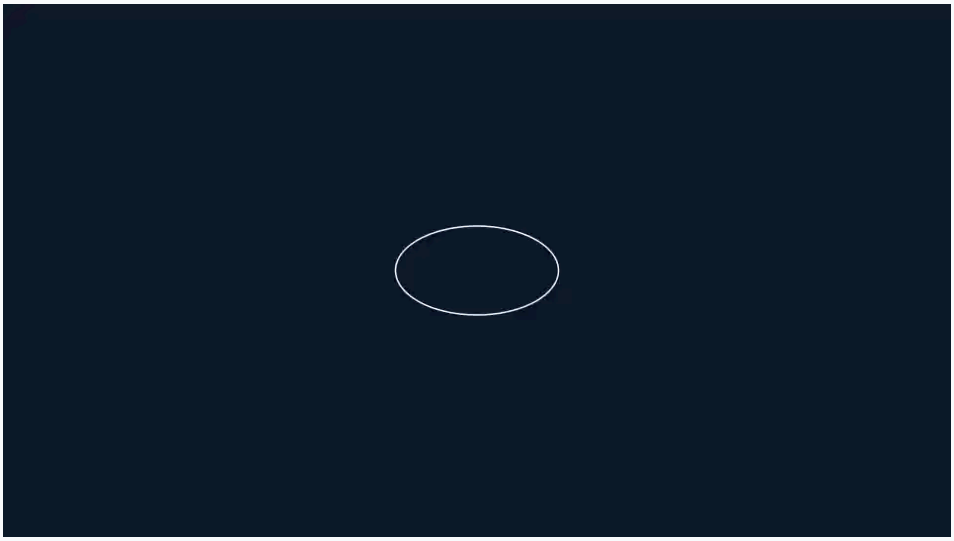
37.1.61. Geometry.curvature_on_curve

FÁBRICA `curvature_on_curve(curve: Drawable, tracker: _ReactiveScalar, window: float = 0.02) -> Drawable`

Círculo osculador oculto, estimado a partir de tres muestras separadas `window` (una fracción de la longitud de arco). En los extremos de una curva abierta las muestras se desplazan hacia dentro, y en una curva cerrada pasan por la costura, así que el círculo existe también en 0 y en 1. En un tramo recto no se dibuja. Dale estilo con `no_fill().stroke(...)`.

Ejemplo:

```
from gaanim import RED, WHITE, Scene
from math import cos, sin, pi
scene = Scene(frame=(16, 9), background="#0f172a")
t = scene.viz.parameter(0.25)
curve = scene.geometry.polyline([(1.375*cos(u), 0.75*sin(u)) for u in (2*pi*i/240 for i in range(241))]).no_fill().stroke(WHITE, 0.025)
circle = scene.geometry.curvature_on_curve(curve, t).no_fill().stroke(RED, 0.025)
scene.play([circle.animate.fade_in().duration(0.3), t.animate.set(0.7).duration(1.4)])
scene.render()
```



37.1.62. Geometry.always_redraw_arc

FÁBRICA `always_redraw_arc(tracker: _ReactiveScalar, cx: float, cy: float, radius: float, start_angle: float, sweep_scale: float = 1.0, sweep_offset: float = 0.0) -> Drawable`

Flecha en arco oculta que se regenera en cada fotograma a partir de un escalar, sin callbacks de Python. Ideal para marcar un ángulo que cambia. Como toda flecha sólida, `fill` la colorea entera; `stroke` fija su contorno.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`tracker`	`float | Parameter | Variable | Computed`	obligatorio	Conduce el ángulo de barrido.
`cx, cy, radius, start_angle`	`float`	obligatorio	Centro, radio y ángulo inicial en radianes.
`sweep_scale / sweep_offset`	`float`	`1.0 / 0.0`	El barrido es `tracker * sweep_scale`

		+ sweep_offset.

Ejemplo:

```
from gaanim import WHITE, Scene
scene = Scene(frame=(16, 9), background="#0f172a")
theta = scene.viz.parameter(0.3)
rot = scene.geometry.always_redraw_arc(theta, 0, 0, 0.69, 0.0).fill(WHITE)
scene.play([rot.animate.fade_in().duration(0.3), theta.animate.set(5.0).duration(1.6)])
scene.render()
```

37.1.62.1. Trazos

37.1.63. Geometry.traced_path

FÁBRICA `traced_path(source: Drawable, *, dissipating_time: Optional[float] = None, max_points: Optional[int] = None, min_distance: float = 0.01) -> Drawable`

Estela 2D que dibuja el recorrido de un objeto. El muestreo empieza en el cursor donde se declara, así que los segmentos anteriores y los seeks no la rellenan de antemano. Revélala con `trail.animate.fade_in()`. Con `dissipating_time` las muestras caducan por la cola en la vista previa, los seeks, las capturas y la exportación. Tras un seek, la estela se reconstruye si la fuente la mueven updaters, `drive_from_samples`, vínculos reactivos o `follow`; una fuente movida solo con `.animate` empieza una estela nueva.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`source`	`Drawable`	obligatorio	Objeto cuya posición se muestrea.
`dissipating_time`	`float | None`	`None`	Segundos que dura cada muestra en la estela; positivo.
`max_points`	`int | None`	`None`	Límite positivo de muestras retenidas.
`min_distance`	`float`	`0.01`	Distancia mínima entre muestras.

Ejemplo:

```
from gaanim import GOLD, RED, Scene, Updater
scene = Scene(frame=(16, 9), background="#0f172a")
dot = scene.geometry.dot(0.09).fill(GOLD).move_to(1.5, 0)
dot.add_updater(Updater.orbit(0, 0, 1.5, 2.0))
trail = scene.geometry.traced_path(dot, dissipating_time=1.0).stroke(RED, 0.04).no_fill()
scene.play([trail.animate.fade_in().duration(0.2)])
scene.wait(1.6)
scene.render()
```

37.1.64. Dibujos axonométricos

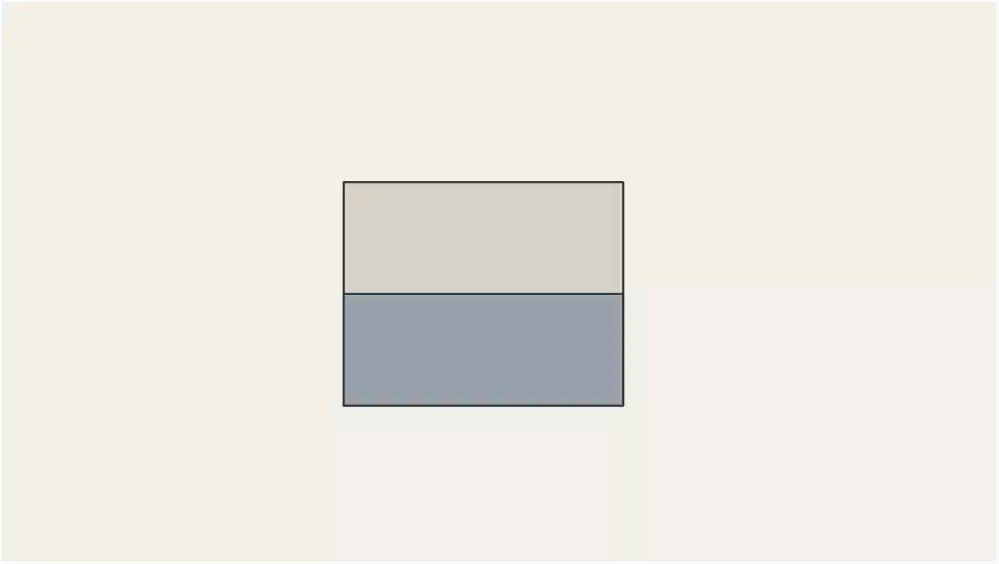
Para planos técnicos en isométrica o dimétrica, con el estilo plano del resto de la escena. `scene.geometry.axonometric(...)` devuelve una proyección que dibuja puntos de un modelo 3D como polígonos y polilíneas 2D normales: se rellenan, se trazan y se animan como cualquier otro `Drawable`. No usa el 3D experimental de la sección siguiente.

El modelo tiene `z` hacia arriba y su planta en `x`/`y`, como un modelo estructural. Un metro del eje menos acortado mide `scale` unidades de escena, así que la isométrica conserva la longitud de los tres ejes, como un dibujo isométrico, y el origen del modelo se dibuja en `origin`. Los trazos de estos dibujos van centrados en el contorno, de modo que una cara vista de canto (un muro en planta) se sigue viendo como línea.

`depth_sort` ordena las caras como un pintor: donde dos dibujos se solapan en pantalla, el más cercano queda encima. Llámalo antes de mostrarlos, como `z_index`, con la vista en la que importa el orden (`view=` toma la de otra proyección, como la vista final de una animación): el orden no se recalcula al cambiar de vista. Tres caras que se tapan en ciclo, o dos que se atraviesan, no tienen un orden exacto; parte esas caras en piezas. `animate_to` lleva todos los dibujos de la proyección a otra vista, vértice a vértice, y la proyección dibuja desde entonces con esa vista.

```
from gaanim import Scene

scene = Scene(frame=(16, 9), background="#f5f1ea")
plan = scene.geometry.axonometric("plan", origin=(-2.5, -2), scale=0.9)
iso = scene.geometry.axonometric("isometric", origin=(-2.5, -2), scale=0.9)
H = 2.5
faces = [
    plan.polygon([(0, 0, 0), (5, 0, 0), (5, 4, 0), (0, 4, 0)]).fill("#d8d2c6"),
```

```
    plan.polygon([(0, 4, 0), (5, 4, 0), (5, 4, H), (0, 4, H)]).fill("#c46a45"),
    plan.polygon([(0, 0, 0), (0, 4, 0), (0, 4, H), (0, 0, H)]).fill("#9c4f32"),
    plan.polygon([(0, 0, H), (5, 0, H), (5, 2, H), (0, 2, H)]).fill("#9aa3ad"),
]
for face in faces:
    face.stroke("#2b2b2b", 0.03)
columns = [plan.line((x, y, 0), (x, y, H)).stroke("#4e535c", 0.06) for x, y in [(5, 2), (5, 4)]]
plan.depth_sort(faces + columns, z_index=10, view=iso)
scene.wait(0.4)
scene.play(plan.animate_to(iso).defaults(duration=1.6))
scene.wait(0.6)
scene.render()
```



37.1.65. Geometry.axonometric

FÁBRICA `axonometric(view: Literal["isometric", "dimetric", "plan", "front", "side"] = "isometric", *, azimuth: Optional[float] = None, elevation: Optional[float] = None, origin: tuple[float, float] = (0.0, 0.0), scale: float = 1.0) -> Axonometric`

Un ángulo u origen no finito, un `scale` que no sea positivo u otro nombre de vista lanzan `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`view`	`str`	`"isometric"`	`"isometric"`, `"dimetric"` (la 2:1, cuyos ejes que se alejan suben una unidad cada dos), `"plan"`, `"front"` (mirando hacia +y) o `"side"` (desde +x).
`azimuth`	`float | None`	`None`	Radianes que la vista gira alrededor de z desde la vista frontal; sustituye al de `view`.

`elevation`	`float | None`	`None`	Radianes que la vista sube sobre el horizonte; negativa, se ve desde abajo. Sustituye a la de `view`.
`origin`	`tuple[float, float]`	`(0, 0)`	Punto de la escena donde se dibuja el origen del modelo.
`scale`	`float`	`1`	Unidades de escena por unidad del modelo en el eje menos acortado.

Devuelve: `Axonometric` — Proyección que dibuja y anima puntos del modelo.

Ejemplo:

```
iso = scene.geometry.axonometric("isometric", origin=(0, -1), scale=0.5)
below = scene.geometry.axonometric("isometric", elevation=-0.5)
```

37.1.66. Axonometric.polygon

MÉTODO `polygon(points: Sequence[tuple[float, float, float]]) -> Drawable`

Menos de tres puntos o una coordenada no finita lanzan `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`points`	`Sequence[tuple[float, float, float]]`	obligatorio	Vértices en coordenadas del modelo, como una cara.

Devuelve: `Drawable` — Un `polygon` en coordenadas de escena.

Ejemplo:

```
wall = iso.polygon([(0, 0, 0), (5, 0, 0), (5, 0, 3), (0, 0, 3)]).fill(GRAY)
```

37.1.67. Axonometric.polyline

MÉTODO `polyline(points: Sequence[tuple[float, float, float]], *, closed: bool = False) -> Drawable`

Menos de dos puntos o una coordenada no finita lanzan `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`points`	`Sequence[tuple[float, float, float]]`	obligatorio	Puntos del modelo, en orden.

`closed`	`bool`	`False`	Une el último punto con el primero.

Devuelve: `Drawable` — Una `polyline` en coordenadas de escena.

37.1.68. Axonometric.line

MÉTODO `line(start: tuple[float, float, float], end: tuple[float, float, float]) -> Drawable`

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`start`	`tuple[float, float, float]`	obligatorio	Un extremo, en el modelo.
`end`	`tuple[float, float, float]`	obligatorio	El otro extremo.

Devuelve: `Drawable` — Una polilínea de dos puntos, que `animate.points` también mueve.

37.1.69. Axonometric.point

MÉTODO `point(x: float, y: float, z: float) -> tuple[float, float]`

Sirve para colocar etiquetas y cotas junto al dibujo; un texto colocado así no sigue a `animate_to`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`x`	`float`	obligatorio	Coordenadas del punto del modelo.
`y`	`float`	obligatorio	
`z`	`float`	obligatorio	

Devuelve: `tuple[float, float]` — Dónde se dibuja, en unidades de escena, con la vista actual.

Ejemplo:

```
label = scene.text("A").move_to(*iso.point(0, 0, 3))
```

37.1.70. Axonometric.depth

MÉTODO `depth(x: float, y: float, z: float) -> float`

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`x`	`float`	obligatorio	Coordenadas del punto del modelo.

y	float	obligatorio	
z	float	obligatorio	

Devuelve: `float` — Cuán cerca del observador está el punto: mayor es más cerca.

37.1.71. Axonometric.depth_sort

MÉTODO `depth_sort(drawables: Sequence[Drawable], *, z_index: int = 0, view: Optional[Axonometric] = None) -> None`

Asigna `z_index` de atrás hacia delante en la vista actual o en la de `view`. Donde dos dibujos se solapan en pantalla, el más cercano allí queda encima; el resto sigue su profundidad media, aunque solo se toquen en un vértice o una arista. Un dibujo de otra proyección, uno repetido o un `z_index` sin sitio para todos lanzan `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`drawables`	`Sequence[Drawable]`	obligatorio	Dibujos hechos con esta proyección.
`z_index`	`int`	`0`	`z_index` del dibujo más lejano; cada siguiente suma uno.
`view`	`Axonometric | None`	`None`	Ordena según la vista de esta proyección en lugar de la actual.

Devuelve: `None`

Ejemplo:

```
floor = iso.polygon([(0, 0, 0), (4, 0, 0), (4, 4, 0), (0, 4, 0)])
slab = iso.polygon([(0, 0, 3), (4, 0, 3), (4, 4, 3), (0, 4, 3)])
column = iso.line((0, 4, 0), (0, 4, 3))
iso.depth_sort([slab, column, floor], z_index=10)
```

37.1.72. Axonometric.animate_to

MÉTODO `animate_to(other: Axonometric) -> Composition`

Lleva cada vértice en línea recta a donde lo dibuja la vista de `other`; fija la duración con `defaults(duration=...)`. Desde entonces esta proyección dibuja con esa vista, y `other` no se lleva los dibujos. Una proyección que no ha dibujado nada lanza `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`other`	`Axonometric`	obligatorio	Proyección cuya vista se adopta.

Devuelve: `Composition` — Un `animate.points` por dibujo, en paralelo.

Ejemplo:

```
plan = scene.geometry.axonometric("plan")
scene.play(iso.animate_to(plan).defaults(duration=1.5))
```

37.1.73. Axonometric.azimuth

PROPIEDAD `azimuth: float`

Devuelve: `float` — Giro de la vista actual alrededor de `z`, en radianes. `elevation`, `origin` y `scale` dan el resto de la vista.

37.1.74. Geometría 3D

EXPERIMENTAL

La API 3D es experimental: todavía tiene errores conocidos y puede cambiar entre versiones. Úsala para explorar, pero no la tomes como estable.

Primitivas iluminadas con `Material3D`, líneas en el espacio y estelas 3D. Gaanim las dibuja con Vello: proyecta cada triángulo con la cámara, los ordena de atrás hacia delante y calcula la luz en la CPU. No hay sombras ni búfer de profundidad, así que las superficies que se cruzan pueden mostrar errores de orden, y todo el 3D queda debajo del contenido 2D. Gaanim usa un mundo dextrógiro con Y hacia arriba: cilindros y conos crecen a lo largo de Y y los planos yacen en XZ. Los puntos 3D son tuplas `(x, y, z)`. La guía Cámara y 3D explica la cámara en perspectiva.

`Primitive3D` extiende `Drawable`, así que posición, rotación, escala, opacidad y material se animan y son deterministas al hacer seek. Los parámetros de geometría quedan fijos tras la construcción. `create()` hace crecer la malla desde su centro mientras aparece; `write()`, que es solo vectorial, se rechaza.

```
from gaanim import BLUE, GOLD, Material3D, Scene

scene = Scene(frame=(16, 9))
cube = scene.geometry.cube(2, material=Material3D.matte(BLUE))
sphere = scene.geometry.sphere(1, segments=32, rings=16,
                     material=Material3D.metal(GOLD)).move_to_3d(3, 0, 0)
floor = scene.geometry.plane(10, 8, subdivisions=(4, 4)).move_to_3d(0, -2, 0)
scene.geometry.lighting_3d("studio", intensity=1.0)
scene.camera.look_at(eye=(6, 4, 8), target=(1, 0, 0))
scene.play([cube.animate.create(), sphere.animate.create(), floor.animate.fade_in()])
scene.play([cube.animate.material(Material3D.metal(GOLD)).duration(1.0)])
scene.render()
```

37.1.75. Geometry.cube

FÁBRICA `cube(size: float = 2.0, *, material: Optional[Material3D] = None) -> Primitive3D`

Cubo centrado de caras planas.

37.1.76. Geometry.sphere

FÁBRICA `sphere(radius: float = 1.0, *, segments: int = 32, rings: int = 16, material: Optional[Material3D] = None) -> Primitive3D`

Esfera UV con sombreado suave; `segments` y `rings` controlan la resolución.

37.1.77. Geometry.cylinder

FÁBRICA `cylinder(radius: float = 1.0, height: float = 2.0, *, segments: int = 32, caps: bool = True, material: Optional[Material3D] = None) -> Primitive3D`

Cilindro a lo largo de Y; `caps=False` lo deja abierto.

37.1.78. Geometry.cone

FÁBRICA `cone(radius: float = 1.0, height: float = 2.0, *, segments: int = 32, cap: bool = True, material: Optional[Material3D] = None) -> Primitive3D`

Cono a lo largo de Y; `cap=False` quita la base.

37.1.79. Geometry.plane

FÁBRICA `plane(width: float = 2.0, height: float = 2.0, *, subdivisions: tuple[int, int] = (1, 1), material: Optional[Material3D] = None) -> Primitive3D`

Plano en XZ con normales hacia arriba, subdividido en `subdivisions` celdas.

37.1.80. Geometry.lighting_3d

MÉTODO `lighting_3d(preset: Literal["studio", "none"] = "studio", intensity: float = 1.0, shadows: bool = True) -> None`

Configura el único equipo automático de luces 3D de la escena. `"studio"` ilumina las primitivas con material al estilo de Gaanim (luz del cielo y del suelo, una luz principal suave desde arriba a la izquierda y un brillo leve en los bordes), escalada por `intensity`; `"none"` deja solo la emisión. `shadows` se acepta por compatibilidad, pero el contenido 3D no proyecta sombras. Las superficies, los gráficos 3D y las líneas no reciben luz. Las luces de `light_3d` sustituyen a la luz principal.

37.1.81. Geometry.light_3d

FÁBRICA `light_3d(kind="point", *, color=None, intensity=1.0) -> Drawable`

Una luz para las primitivas con material, en el origen. No dibuja nada; es un dibujable, así que `move_to_3d` la coloca y `animate`, los grupos y `follow` la mueven. Una luz `"point"` alumbra desde su posición en todas direcciones; una `"directional"`, con rayos paralelos desde la dirección de su posición vista desde el origen, como el sol. Mientras la escena muestra luces, sustituyen a la luz principal fija de `lighting_3d`, cuya luz del cielo y del suelo se mantiene, y cada una alumbra en su `color` (blanco por defecto) con su `intensity`, atenuada por su opacidad: `fade_in` y `fade_out` la encienden y la apagan. Con `lighting_3d("none")` solo alumbran las

luces. Las superficies, los gráficos 3D y las líneas no reciben luz. Una `intensity` negativa o no finita, u otro `kind`, lanzan `ValueError`.

Ejemplo:

```
lamp = scene.geometry.light_3d(color=GOLD, intensity=1.5).move_to_3d(3, 2, 4)
scene.play([lamp.animate.move_to_3d(-3, 2, 4).duration(2.0)])
```

37.1.82. Material3D.matte / metal / emissive / color / roughness / metallic / emissive_color / emissive_strength

CLASE `Material3D(color=WHITE, roughness=0.55, metallic=0.0, emissive=None, emissive_strength=0.0)`

Material de una primitiva 3D iluminada, sombreado por Gaanim: la rugosidad y la metalicidad dan forma al brillo y la emisión se suma a la luz. Sus propiedades numéricas se interpolan en espacio lineal y cuyos rangos se validan. Los atajos `Material3D.matte(color)`, `Material3D.metal(color)` y `Material3D.emissive(color, strength=1.0)` cubren los aspectos habituales. Las propiedades `color`, `roughness`, `metallic`, `emissive_color` y `emissive_strength` son de solo lectura.

Ejemplo:

```
glass = Material3D(color="#93c5fd", roughness=0.1, metallic=0.0)
lamp = Material3D.emissive(GOLD, strength=2.0)
```

37.1.83. Geometry.extrude

FÁBRICA `extrude(source: Drawable, depth: float = 0.2, *, bevel: float = 0.0, material: Optional[Material3D] = None, tolerance: float = 0.01, keep_source: bool = False) -> Primitive3D`

El contorno se lee al compilar la escena, así que funcionan textos, ecuaciones, grupos y SVG importados, y se rellena como se dibuja (regla nonzero): las letras conservan sus huecos. La malla queda centrada donde está el origen en el plano z = 0, con la tapa frontal hacia la cámara predeterminada. Los trazos de menos del doble del chaflán conservan aristas rectas. El origen sigue siendo un drawable propio, oculto desde el cursor (el 2D se dibuja encima del 3D) salvo con `keep_source`; cambiarlo después no altera la malla. Gira la cámara para ver la profundidad. Un origen de otra escena, 3D, imagen, video o Lottie, una `depth` o `tolerance` no positiva o un `bevel` negativo o de al menos la mitad de `depth` lanzan `ValueError`; un contorno sin área avisa y deja la malla vacía.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`source`	`Drawable`	obligatorio	Drawable 2D cuyo relleno se extruye: forma, texto, ecuación, grupo o SVG.

`depth`	`float`	`0.2`	Profundidad de la malla, centrada en z = 0.
`bevel`	`float`	`0.0`	Ancho del chaflán a 45° en ambas tapas; menor que la mitad de `depth`.
`material`	`Material3D | None`	`None`	Material; sin él, mate del color de relleno del origen.
`tolerance`	`float`	`0.01`	Tolerancia al aplanar las curvas, en unidades de escena; menor es más suave y pesado.
`keep_source`	`bool`	`False`	Mantiene visible el drawable de origen.

Devuelve: `Primitive3D` — Malla cerrada con tapas, paredes y chaflán.

Ejemplo:

```
from gaanim import *

scene = Scene(frame=(16, 9))
word = scene.geometry.extrude(scene.text("HOLA", size=1.6), depth=0.4, bevel=0.04,
                              material=Material3D.metal(GOLD))
star = scene.geometry.extrude(scene.geometry.star(5, 1.2, 0.5).fill(BLUE).move_to(4.5, 0),
depth=0.3)
scene.camera.perspective(fov_y=0.785)
scene.camera.look_at(eye=(2, 3, 12), target=(1, 0, 0))
scene.play([word.animate.create(), star.animate.create()])
scene.play([scene.camera.animate.orbit(delta_yaw=0.6, delta_pitch=0.0).duration(2)])
scene.render()
```

37.1.84. Primitive3D.material

MÉTODO `material(material: Material3D) -> Self`

Cambia el material al instante; `.animate.material(...)` lo interpola.

37.1.85. Geometry.polyline_3d

FÁBRICA `polyline_3d(points: Sequence[tuple[float, float, float]], color: Optional[ColorLike] = None, *, colors: Optional[Sequence[Color]] = None, colormap: Optional[str] = None) -> Drawable`

Tira de líneas 3D en el espacio del mundo. La lista `colors` debe tener la misma longitud que `points`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`points`	`Sequence[tuple[float, float, float]]`	obligatorio	Al menos dos puntos finitos del mundo.
`color`	`Color | None`	`None`	Color uniforme.
`colors`	`Sequence[Color] | None`	`None`	Un color por punto; tiene prioridad sobre `colormap`.
`colormap`	`str | None`	`None`	Degradado ordenado a lo largo de la línea, por ejemplo `"inferno"`, `"viridis"` o `"plasma"`.

Ejemplo:

```
points = [(math.cos(t / 8), math.sin(t / 8), t / 40 - 1) for t in range(81)]
helix = scene.geometry.polyline_3d(points, colormap="inferno")
```

37.1.86. Geometry.traced_path_3d

FÁBRICA `traced_path_3d(source: Drawable, *, colormap: Optional[str] = None, dissipating_time: Optional[float] = None, max_points: Optional[int] = None, min_distance: float = 0.1) -> Drawable`

Estela 3D de la posición de un objeto en el mundo. Funciona con `Updater` y `add_updater_fn`; el muestreo empieza donde se declara. Revélala con `fade_in` o `create`. `dissipating_time` hace caducar la cola, `max_points` limita la memoria y `min_distance` (0.1 por defecto) filtra muestras casi iguales. Admite los mapas `"inferno"`, `"viridis"` y `"plasma"`.

Ejemplo:

```
dot = scene.geometry.dot(0.09).move_to_3d(1, 0, 0)
dot.add_updater(Updater.orbit(0, 0, 1, 1.5))
trail = scene.geometry.traced_path_3d(
```

```
    dot, colormap="viridis", dissipating_time=2.0, max_points=600
)
scene.play([trail.animate.fade_in()])
```

Este ejemplo combina ejes 3D, una polilínea, un punto que orbita con su estela y la cámara en perspectiva:

```
from gaanim import Axis, BLACK, GOLD, RED, WHITE, Scene, Updater
scene = Scene(frame=(16, 9), background=BLACK)

axes = scene.viz.cartesian_3d(
    Axis.linear(-3, 3).ticks(1).label("x").style(color=WHITE),
    Axis.linear(-3, 3).ticks(1).label("y").style(color=WHITE),
    Axis.linear(-2, 2).ticks(1).label("z").style(color=WHITE),
    size=(6, 6, 4),
)
path = scene.geometry.polyline_3d([(-2, -1, -1), (0, 1, 0), (2, -1, 1)], color=RED)
dot = scene.geometry.dot(0.1).fill(GOLD).move_to_3d(1, 0, 0).billboard()
dot.add_updater(Updater.orbit(0, 0, 1, 1.2))
trail = scene.geometry.traced_path_3d(dot, colormap="viridis", max_points=120)

scene.camera.perspective(fov_y=0.785, near=0.1, far=100.0)
scene.camera.look_at(eye=(7, 5, 6), target=(0, 0, 0))
scene.play([
    axes.animate.create().duration(0.8),
    path.animate.create().duration(0.8),
    dot.animate.fade_in().duration(0.4),
    trail.animate.fade_in().duration(0.4),
])
scene.play([scene.camera.animate.orbit(delta_yaw=0.5, delta_pitch=0.1).duration(0.8)])
scene.play([scene.camera.animate.dolly(factor=0.85).duration(0.5)])
scene.wait(0.5)
scene.render()
```

CAPÍTULO

38. Texto

scene.text: prosa, ecuaciones, partes semánticas, estilo, flujo, selecciones y documentos Typst

`scene.text(...)` crea prosa, títulos, párrafos, matemáticas y contenido mixto; `scene.text.equation(...)` es su atajo para ecuaciones en bloque. Ambas devuelven un `Text`: un `Drawable` vectorial que conserva la estructura semántica, se mide con el mismo motor que Layout y permite seleccionar y animar partes sueltas.

```
from gaanim import GOLD, Scene, TextFlow, part
scene = Scene(frame=(16, 9), background="#0f172a")
formula = part("formula", "$E = ", part("mass", "m", color=GOLD), " c^2$")
copy = scene.text(
    "La energía es ", formula,
    role="body",
    flow=TextFlow(wrap=5, align="center", line_spacing=1.2),
).move_to(0, 0)
scene.play([copy.animate.write(by="part").duration(1.0)])
scene.render()
```

Cada pieza tiene una responsabilidad:

- `TextStyle` controla el aspecto de los glifos y las métricas tipográficas.
- `TextFlow` controla cómo se componen las líneas dentro del texto.
- Layout controla la caja exterior: relleno, ajuste, crecimiento y posición.
- `TextSelection` señala glifos dentro de un `Text`; nunca es un hijo independiente de Layout.

38.1.1. Crear texto

Las dos fábricas de texto y los roles tipográficos del tema.

38.1.2. Scene.text

FÁBRICA `scene.text(*content, role=None, style=None, flow=None, font=None, math_font=None, size=None, weight=None, italic=None, color=None, opacity=None, letter_spacing=None, word_spacing=None, baseline=None, wrap=None, text_align=None, line_spacing=None, max_lines=None, overflow=None, direction=None, hyphenate=None, lang=None, markup=None) -> Text`

`$…$` activa la matemática y `\$` escribe un dólar literal. `color` acepta un `Color`, una cadena CSS/hex o una tupla RGB(A); `None` hereda el color del estilo o del tema. `lang="es"` con `hyphenate=True` da párrafos justificados con guiones en español. Contenido, delimitadores, roles, métricas o flujos inválidos lanzan `TypeError` o `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`content`	`str | TextPart | TextParts`	obligatorio	Una o varias cadenas, partes semánticas o grupos de partes. El resultado no puede estar vacío.
`role`	`str | None`	`None`	Rol tipográfico. Si todo es matemático se infiere `math`; si no, `body`.
`style / flow`	`TextStyle | None / TextFlow | None`	`None`	Estilo y flujo reutilizables.
`font … baseline`	`argumentos con nombre`	`None`	Ajustes directos de estilo: fuente, métricas, color, opacidad, espaciado y línea base. Sustituyen a `style`.

`wrap … lang`	`argumentos con nombre`	`None`	Ajustes directos de flujo: ajuste de línea, alineación, límite de líneas, desbordamiento, dirección, guiones e idioma. Sustituyen a `flow`.
`markup`	`bool | None`	`None`	Interpreta `*negrita*` y `_cursiva_`. `False` deja * y _ literales (la matemática `$…$` sigue activa); `None` usa `text_markup` del tema, que por defecto es `True`.

Ejemplo:

```
title = scene.text("Movimiento circular", role="title")
note = scene.text("Radio $r = 1.5$", size=0.36, color="#94a3b8").move_to(0, -1)
```

38.1.3. Typography.equation

FÁBRICA `equation(*content: TextContent, role: Optional[TextRole] = None, style: Optional[TextStyle] = None, flow: Optional[TextFlow] = None, font: Optional[str] = None, math_font: Optional[str] = None, size: Optional[float] = None, weight: Optional[int] = None, italic: Optional[bool] = None, color: Optional[ColorLike] = None, opacity: Optional[float] = None, letter_spacing: Optional[float] = None, word_spacing: Optional[float] = None, baseline: Optional[float] = None, wrap: Optional[TextWrap] = None, text_align: Optional[TextAlign] = None, line_spacing: Optional[float] = None, max_lines: Optional[int] = None, overflow: Optional[TextOverflow] = None, direction: Optional[TextDirection] = None, hyphenate: Optional[bool] = None) -> Text`

Envuelve el contenido como `$ … $`: esos espacios hacen que Typst componga una ecuación en bloque. Cada frontera entre piezas es un espacio normal de Typst, así que Typst decide el espaciado de operadores e identificadores. Sin rol ni tamaño explícitos usa el tamaño matemático de 0.44 unidades. Las filas de una ecuación partida con `\` guardan la separación de Typst entre filas con `line_spacing=1` y cada unidad de más la ensancha un tamaño de letra, como entre líneas de texto. Contenido vacío o matemática inválida lanzan `ValueError` al crear la ecuación.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`content`	`str | TextPart | TextParts`	obligatorio	Ecuación sin los delimitadores `$`.
`opciones`	`igual que scene.text`	`None`	Comparte todo el estilo y el flujo de `scene.text` (salvo `lang` y `markup`).

Ejemplo:

```
from gaanim import GOLD, Scene, part, parts
scene = Scene(frame=(16, 9), background="#0f172a")
equation = scene.text.equation(
    part("sum_force", "sum F_t"),
    "=",
    parts(mass="m", acceleration="a_t"),
).move_to(0, 0)
equation["acceleration"].fill(GOLD)
scene.play([equation.animate.write(by="part").duration(1.0)])
scene.render()
```

38.1.4. quantity

FUNCIÓN `quantity(value: int | float, unit: str = "", *, decimals: int | None = None, decimal_separator: str = ".") -> str`

Escribe cantidades con unidades de forma consistente, como `siunitx` en LaTeX: un espacio fino entre número y unidad (salvo en °, ′ y ″), unidades en redonda, exponentes y una barra en lugar de una fracción. Así `1 m³` y `181 L` quedan espaciados igual. Una unidad mal formada lanza `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`value`	`int | float`	obligatorio	El número; un entero conserva sus cifras.
`unit`	`str`	`""`	Factores separados por espacios, `*`, `·` o `.`, cada uno con un exponente opcional (`m^3`, `s^-1`); `/` divide por los que siguen: `"kg m/s^2"`.
`decimals`	`int | None`	`None`	Decimales fijos; sin él, la forma más corta del número.
`decimal_separator`	`str`	`"."`	Un carácter, por ejemplo `","`.

Devuelve: `str` — Matemática Typst para `scene.text.equation`.

Ejemplo:

```
from gaanim import Scene, quantity
scene = Scene(frame=(16, 9), background="#0f172a")
volumen = scene.text.equation("V =", quantity(1, "m^3"), "=", quantity(1000, "L")).move_to(0, 0.8)
gravedad = scene.text.equation("g =", quantity(9.81, "m/s^2", decimal_separator=",")).move_to(0,
-0.6)
scene.play([volumen.animate.write(), gravedad.animate.write()])
scene.render()
```

38.1.4.1. Roles

Los roles son `title`, `subtitle`, `kicker`, `heading`, `body`, `caption`, `label`, `code` y `math`. Sus tamaños predeterminados, en unidades de escena, son: `title` 0.64, `subtitle` 0.48, `kicker` 0.32, `heading` 0.48, `body` 0.40, `caption` 0.32, `label` 0.36, `code` 0.36 y `math` 0.44. La prosa usa New Computer Modern, el código JetBrains Mono y la matemática New Computer Modern Math. Las tres vienen con Gaanim,

así que el texto se ve igual en cualquier equipo y en el navegador. Con un tema, `kicker` toma el color `accent`, ideal para la línea corta sobre un título:

```
kicker = scene.text("MISMO TERREMOTO. TRES EDIFICIOS.", role="kicker").move_to(0, 3.77)
title = scene.text("¿Cuál sufrirá más?", role="title").move_to(0, 3)
```

El estilo se resuelve en este orden, de menor a mayor prioridad: rol y tema, `TextStyle`/`TextFlow`, argumentos directos de `scene.text`, estilo local de cada `part` y, por último, `selection.fill(...)` o `text.fill(...)` posteriores.

38.1.5. Partes semánticas y matemáticas

Nombra fragmentos del contenido para colorearlos, animarlos o transformarlos después sin contar caracteres.

38.1.6. part

FUNCIÓN `part(name, *content, style=None, font=None, math_font=None, size=None, weight=None, italic=None, color=None, opacity=None, letter_spacing=None, word_spacing=None, baseline=None) -> TextPart`

Subárbol semántico inmutable. La ruta de nombres anidados es estable y la usan las selecciones y las transiciones de texto. Nombres repetidos o vacíos, métricas inválidas o contenido anidado inválido lanzan `ValueError` o `TypeError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`name`	`str`	obligatorio	Nombre no vacío y único entre sus hermanos.
`content`	`str | TextPart | TextParts`	obligatorio	Contenido anidado.
`style / estilo directo`	`TextStyle | argumentos con nombre`	`None`	Tipografía local que hereda todo el subárbol.

Ejemplo:

```
from gaanim import BLUE, GOLD, Scene, part
scene = Scene(frame=(16, 9), background="#0f172a")
formula = part(
    "formula",
```

```
    "$",
    part("variable", "x", color=BLUE),
    " dot 5 = ",
    part("result", "25", color=GOLD),
    "$",
)
text = scene.text("Resultado: ", formula).move_to(0, 0)
scene.play([text.animate.write(by="part", stagger=0.05).duration(1.1)])
scene.render()
```

38.1.7. parts

FUNCIÓN `parts(mapping=None, /, **content: str) -> TextParts`

Grupo ordenado de partes simples, equivalente a varias llamadas a `part()`. Dentro de `$…$`, las entradas contiguas son tokens distintos de Typst y conservan su espaciado ajustado. Usa `part()` para estilos locales o anidamiento. Entradas vacías, repetidas o mezclar un mapa con argumentos con nombre lanzan `ValueError`; nombres o valores que no son cadenas, `TypeError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`mapping`	`Mapping[str, str] | None`	`None`	Nombres y texto en orden. Admite nombres que no son identificadores de Python, como `"tb:dist"`.
`content`	`str con nombre`	obligatorio	Atajo para nombres que son identificadores: `parts(mass="m")` equivale a `parts({"mass": "m"})`.

Ejemplo:

```
from gaanim import GOLD, Scene, parts
scene = Scene(frame=(16, 9), background="#0f172a")
equation = scene.text.equation(
    "-",
    parts(mass_left="m", gravity="g sin(theta)"),
    "=",
    parts(mass_right="m", length="L", acceleration="theta''"),
).move_to(0, 0)
scene.play([equation.animate.write(by="part").duration(1.2)])
scene.play([equation["gravity"].animate.indicate().duration(0.6)])
scene.play([equation["acceleration"].animate.fill(GOLD).duration(0.6)])
scene.render()
```

Un mapa admite nombres que los argumentos con nombre no pueden expresar:

```
label = scene.text(parts({"tb:dist": "d = ", "x-1": "4.2 m"}))
label["tb:dist"].fill(GOLD)
```

38.1.7.1. Énfasis en línea

Las cadenas admiten un marcado mínimo inspirado en Typst: `*negrita*`, `_cursiva_` y ambos anidados, `*_así_*`. Se compila en las mismas pasadas que `TextStyle`, así que funciona con ajuste de línea, selecciones, animaciones, partes y `bounds()`.

```
from gaanim import GOLD, Scene

scene = Scene(frame=(16, 9), background="#0f172a")
copy = scene.text(
    "Normal, _cursiva_, *negrita* y *_ambas_*.",
    size=0.36,
).move_to(0, 0)
scene.play([copy.animate.write(by="word", stagger=0.05).duration(1.2)])
scene.play([copy.words[2].animate.indicate().duration(0.6)])
scene.render()
```

- `\\*` y `\\_` escriben los delimitadores literales.
- El marcado puede cruzar cadenas contiguas y fronteras de `part()` sin añadir huecos.
- Dentro de `$…$` los marcadores son sintaxis matemática: `x_1` es un subíndice y `*` una multiplicación.
- Los guiones bajos dentro de palabras (`snake_case`), los marcadores repetidos (`__init__`) y una expresión con espacios como `5 * 4` quedan literales.
- Un delimitador de apertura sin cierre, o un anidamiento cruzado, lanza `ValueError`.
- Para contenido técnico lleno de `_` o `*` (`tb:dist_comp`, `X1_2`) pasa `markup=False`: todo queda literal, barras invertidas incluidas, y `$…$` sigue siendo matemática. `text.become(...)` conserva el modo salvo que pases `markup=`.
- `Theme(text_markup=False)` lo hace predeterminado para `scene.text`, `badge` y `chip`.

```
label = scene.text("Valores: V_e del piso 1 (tb:agriet_xy)", markup=False)
```

38.1.7.2. Delimitadores matemáticos

- `$…$` cambia el compositor Typst a matemáticas; `$$…$$` usa el mismo compositor y no crea un objeto distinto.
- `scene.text.equation(...)` añade `$ … $`: no escribas los delimitadores.
- `\$` escribe un dólar literal y un delimitador sin pareja lanza `ValueError`.
- Con `role="code"` el texto es literal: todo `$`, `*`, `_` y barra invertida se muestra tal cual, así que `scene.text("$ gaanim init", role="code")` o `scene.text("stagger(*grows)", role="code")` no necesitan escapes.
- Si todo el contenido es matemático, el rol inferido es `math`; si mezcla prosa y matemáticas, `body`.
- Dentro de la matemática, cada frontera entre piezas es un espacio normal de Typst, así que no hace falta escribir `"= "`. Fuera de la matemática las fronteras son exactas.
- Los estilos locales de una parte quedan dentro de la misma ecuación Typst: cambiar su color, fuente o tamaño nunca añade huecos artificiales.
- La sintaxis puede cruzar fronteras: `part("x", "x"), "_1"` se compila como `x _1` y Typst mantiene `_1` como subíndice.

38.1.8. Estilo y flujo

Aspecto de los glifos y composición de las líneas, como valores reutilizables.

38.1.9. TextStyle

CLASE `TextStyle(*, font=None, math_font=None, fallbacks=(), size=None, weight=None, italic=None, color=None, stroke=None, stroke_width=None, opacity=None, letter_spacing=None, word_spacing=None, decorations=(), baseline=None)`

Estilo tipográfico inmutable y reutilizable. No tiene ancho, alto, relleno ni ajuste de caja: eso es de Layout. El estilo raíz se aplica a todos los glifos y el de una parte, a los suyos: fuente, tamaño, peso, cursiva, color, contorno, espaciado, decoraciones y línea base. `opacity` se aplica a todo el texto. Valores inválidos lanzan `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`font / math_font`	`str | None`	`None`	Familias de prosa y de matemáticas.
`fallbacks`	`Sequence[str]`	`()`	Familias de respaldo en orden.
`size`	`float | None`	`None`	Tamaño positivo en unidades de escena.
`weight`	`int | None`	`None`	Peso numérico de 1 a 1000.
`color / stroke / stroke_width`	`Color | None / Color | None / float | None`	`None`	Relleno de los glifos y contorno opcional, de `stroke_width` de

			ancho (0,02 si solo se da `stroke`).
`opacity`	`float | None`	`None`	Opacidad de todo el texto, de 0 a 1.
`letter_spacing / word_spacing`	`float | None`	`None`	Espaciado adicional no negativo, en unidades de escena.
`decorations`	`Sequence[str]`	`()`	`underline`, `strike` o `strikethrough`.
`baseline`	`float | None`	`None`	Desplazamiento de la línea base; positivo sube los glifos dentro de su línea.

Ejemplo:

```
display = TextStyle(
    font="New Computer Modern",
```

```
    math_font="New Computer Modern Math",
    size=0.42,
    weight=650,
    color=GOLD,
    letter_spacing=0.02,
    decorations=("underline",),
)
heading = scene.text("Resultados", style=display)
```

38.1.10. TextFlow

CLASE `TextFlow(*, wrap="auto", align="left", line_spacing=1.0, max_lines=None, overflow="clip", direction="auto", hyphenate=False, lang=None)`

Opciones inmutables de composición de líneas. Los argumentos directos de `scene.text` sustituyen a este objeto.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`wrap`	`"auto" | False | float`	`"auto"`	`"auto"` usa el ancho que ofrece Layout (o el área segura); `False` mantiene una línea salvo saltos explícitos; un número limita el ancho tipográfico.
`align`	`str`	`"left"`	`left`, `center`, `right` o `justify`.
`line_spacing`	`float`	`1.0`	Distancia entre líneas base, en múltiplos del tamaño de letra, como `line-height` en CSS: con `1.2` y letra de 0.4, las líneas quedan a 0.48. Con el valor por defecto, `1.0`, las líneas quedan a un tamaño de letra, sin solaparse; súbelo para párrafos largos. Debe ser positivo.
`max_lines / overflow`	`int | None / str`	`None / "clip"`	Límite de líneas y qué pasa con el

			resto: `clip` conserva las palabras que caben (una primera palabra que no cabe se corta entre caracteres), `ellipsis` las termina con «…» y `visible` dibuja todas las líneas.
`direction`	`str`	`"auto"`	`auto`, `ltr` o `rtl`.
`hyphenate / lang`	`bool / str | None`	`False / None`	Guiones de Typst y código ISO 639 en minúsculas (`"es"`, `"en"`…) que elige los patrones; `None` usa inglés.

Ejemplo:

```
from gaanim import GOLD, Scene, TextFlow
scene = Scene(frame=(16, 9), background="#0f172a", margin=0.375)
body = scene.text(
    "El mismo texto se mide con el ancho que ofrece su tarjeta de Layout v2.",
    role="body",
    flow=TextFlow(wrap="auto", align="justify", line_spacing=1.25),
)
page = scene.layout.column(
    [scene.text("Texto responsive", role="heading").fill(GOLD), body],
    within="safe", width="fill", height="fill", padding=0.35, gap=0.225,
)
scene.play([page.animate.fade_in().duration(0.7)])
scene.render()
```

Sin `flow` ni `text_align`, el anclaje de `move_to` también alinea un texto con saltos de línea explícitos: `Anchor.*_LEFT` y `TextAnchor.BASELINE_LEFT` alinean a la izquierda, `*_RIGHT` a la derecha y los anclajes centrados centran. Un `flow` o `text_align` explícitos siempre ganan; el texto de una sola línea conserva la alineación izquierda.

```
note = scene.text("Fuente: ensayo 3\nEscala 1:50").move_to(7.5, -4, Anchor.BOTTOM_RIGHT)
```

38.1.11. Posición

Colocación por línea base para que palabras y ecuaciones se alineen.

38.1.12. Text.move_to

MÉTODO `move_to(x, y, anchor: Anchor | TextAnchor | None = None) | move_to(reference) | move_to(point) -> Text`

Una sola línea se coloca por defecto con `TextAnchor.BASELINE_CENTER`: el centro horizontal visual en `x` y la línea base en `y`. Así, textos y ecuaciones con distintas ascendentes, fracciones o tamaños comparten línea base. Un bloque de varias líneas sin anclaje usa su centro visual, y un `TextAnchor` explícito usa la primera línea. Los anclajes geométricos (`Anchor.TOP_LEFT`...) se basan en los límites y no garantizan una línea base común. Un `Drawable` o un `AnchorPoint` centran el texto sobre la referencia sin seguirla. Un texto gestionado por Layout lanza `LayoutOwnershipError`; `text.animate.move_to(...)` sí lo mueve.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`x / y`	`float | fuente reactiva`	obligatorio	Punto de destino en unidades de escena.
`anchor`	`Anchor | str | TextAnchor | None`	`None`	Anclaje geométrico (un `Anchor` o su nombre, como `"top_left"`), o `TextAnchor.BASELINE_LEFT`, `BASELINE_CENTER` o `BASELINE_RIGHT` sobre la línea base.

Ejemplo:

```
word = scene.text("Tipografía").move_to(0, 1)
equation = scene.text.equation("frac(x_1^2, y_2) = 1").move_to(0, -0.5,
TextAnchor.BASELINE_CENTER)
left = scene.text("alineado a la izquierda").move_to(-4, -2, TextAnchor.BASELINE_LEFT)
corner = scene.text("esquina").move_to(-7, 4, Anchor.TOP_LEFT)
```

`Text` hereda todos los métodos de `Drawable` y conserva su tipo al encadenarlos.

38.1.13. Text.glow / blur / shadow / no_effects / blend

MÉTODO `glow(color, radius=0.16, intensity=1.0) · blur(sigma=0.04) · shadow(color, x=0.08, y=-0.08, blur=0.06) · no_effects() · blend(mode="normal") -> Text`

Los efectos y el modo de fusión de `Drawable`, redefinidos para devolver el `Text` y conservar el anclaje tipográfico de un `move_to` posterior.

Ejemplo:

```
title = scene.text("Gaanim", role="title").glow(BLUE).move_to(0, 1, TextAnchor.BASELINE_CENTER)
```

38.1.14. Selecciones

Señalan grafemas, palabras, líneas o partes de un texto para darles estilo o animarlas sin separarlas del texto.

38.1.15. TextSelection

CLASE `text[name] · text[index] · text[start:end] · selection[name] -> TextSelection`

Selección diferida dentro de su `Text`. Un rango de grafemas, palabras o líneas selecciona el texto desde la primera hasta la última unidad, puntuación incluida. Cadenas que no son parte ni texto lanzan `KeyError`; índices fuera de rango, `IndexError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`name`	`str`	obligatorio	Una parte de primer nivel; sigue indexando para bajar por partes anidadas. Una cadena que no es parte selecciona cada aparición literal de ese texto, sin distinguir mayúsculas ni espacios.
`index / slice`	`int | slice`	obligatorio	Grafemas renderizados; admite índices negativos y rangos contiguos con paso 1.

Ejemplo:

```
from gaanim import BLUE, GOLD, Scene, part
scene = Scene(frame=(16, 9), background="#0f172a")
copy = scene.text(
    "La ", part("concept", "energía"), " depende de ",
    part("formula", "$", part("mass", "m"), " c^2$"),
).move_to(0, 0)
copy["concept"].fill(GOLD)
copy["formula"]["mass"].fill(BLUE)
scene.play([copy.animate.write(by="word").duration(1.0)])
scene.play(copy.words[1].animate.pulse().duration(0.6))
scene.play(copy["formula"]["mass"].animate.focus().duration(0.6))
scene.render()
```

38.1.16. Text.graphemes / words / lines / parts

PROPIEDAD `graphemes · words · lines · parts -> TextQuery`

Vistas indexables (`TextQuery`) sobre las unidades del texto. `words` sigue los límites de palabra de Unicode, así que la puntuación no es una palabra: `"uno dos, tres"` tiene tres y `words[1:3]` selecciona `dos, tres`. `lines` sigue los saltos `\n` explícitos, no las líneas creadas por el ajuste. `parts` recorre el árbol semántico en profundidad. `len(query)` cuenta las unidades y `"mass" in text.parts` comprueba si existe una parte, también con rutas anidadas como `"formula.mass"`.

Ejemplo:

```
copy = scene.text("uno dos, tres")
tail = copy.words[1:3]
if "mass" in copy.parts:
    copy["mass"].fill(GOLD)
```

38.1.17. TextSelection.fill

MÉTODO `fill(color: ColorLike) -> TextSelection`

Colorea los glifos seleccionados de forma persistente: antes de que el texto se muestre es el color declarado de la selección, y después los glifos cambian de color en el cursor, como con cualquier cambio inmediato. En matemáticas, la parte sigue en la misma ecuación Typst: cambiar su color no añade espacios ni mueve los términos vecinos.

Ejemplo:

```
formula = scene.text.equation("E = ", part("mass", "m"), " c^2")
formula["mass"].fill(GOLD)
```

38.1.18. TextSelection.marker

MÉTODO `marker(color: ColorLike | None = None, *, skew: float = 0.05, blend: Literal["normal", "multiply"] = "normal", opacity: float = 0.45, padding: float | None = None) -> TextSelection`

Coloca al instante bandas de rotulador detrás de cada línea seleccionada, con los mismos argumentos que `animate.marker`. Aparecen en el cursor (el inicio de la escena durante la declaración) y no siguen movimientos posteriores del texto.

Ejemplo:

```
quote = scene.text("Lo que no se mide no se puede mejorar")
quote.words[0:2].marker(opacity=0.35)
```

38.1.19. TextSelection.annotate

PROPIEDAD `annotate: Annotations`

Las anotaciones a mano alzada de `Drawable.annotate` alrededor de una parte del texto. Siguen los glifos aunque el texto se mueva.

Devuelve: `Annotations` — Constructor de anotaciones alrededor de los glifos seleccionados.

Ejemplo:

```
eq = scene.text.equation("E = ", part("rhs", "m c^2"))
scene.play([eq["rhs"].annotate.underline(color=RED, seed=1).animate.create()])
```

38.1.20. TextSelection.animate / TextSelectionAnimation.fill / opacity

PROPIEDAD

Proxy `TextSelectionAnimation` limitado a los glifos seleccionados. Acepta `fill` y `opacity`, combinables (`animate.fill(RED).opacity(0.6)`); los destinos de transformación, escala, rotación, material o trazo lanzan `TypeError`. Cada método devuelve un `Anim` para `scene.play`.

Ejemplo:

```
eq = scene.text("$E = ", part("mass", "m"), " c^2$")
scene.play([eq.animate.write().duration(0.8),
eq["mass"].animate.fill(RED).opacity(0.7).duration(0.8)])
```

38.1.21. TextSelectionAnimation.indicate / pulse / wiggle / wave / highlight / focus

MÉTODO `indicate() · pulse() · wiggle() · wave() · highlight() · focus() -> Anim`

Énfasis transitorios sobre la selección. `indicate` hace crecer los glifos 1,2 veces y los tiñe del `"accent"` del tema (o de su `"foreground"` si ya tienen el color del acento); `pulse` los hace crecer dos veces seguidas, como un latido, sin teñirlos; `wiggle` los sacude de lado a lado; `wave` levanta cada glifo un tercio de su altura y lo baja, uno tras otro; `highlight` los tiñe del acento y los devuelve a su color sin moverlos; `focus` atenúa el resto del texto. No cambian la medición del texto ni provocan reflow.

Ejemplo:

```
eq = scene.text.equation("F = ", part("mass", "m"), " a")
scene.play([eq["mass"].animate.highlight().duration(0.8)])
scene.play([eq["mass"].animate.wave().duration(0.8)])
```

38.1.22. TextSelectionAnimation.cancel

MÉTODO `cancel() -> Anim`

Tacha la selección con una marca diagonal y atenúa sus glifos. La siguiente transición de texto que reemplace el contenido retira ambos.

38.1.23. TextSelectionAnimation.reveal

MÉTODO `reveal(style: Literal["fade", "wipe", "from_below"] = "fade") -> Anim`

Hace aparecer solo los glifos seleccionados con `"fade"`, `"wipe"` (traza y luego rellena) o `"from_below"` (fundido con una subida corta), para que un término entre en una ecuación ya visible. Los glifos seleccionados siguen ocultos hasta el `reveal`, también mientras se escribe el resto. Un estilo desconocido lanza `ValueError`. Para revelar un texto completo por líneas o palabras, usa `Anim.reveal` (ver Animaciones).

Ejemplo:

```
eq = scene.text.equation("a^2 + b^2 = ", part("rhs", "c^2"))
scene.play([eq["rhs"].animate.reveal("from_below").duration(0.6)])
```

38.1.24. TextSelectionAnimation.brace

MÉTODO `brace(label: str = "", *, above: bool = False) -> Anim`

Dibuja una llave bajo la selección (sobre ella con `above=True`) y hace aparecer `label`. La llave y la etiqueta son objetos nuevos, del color de los glifos, que permanecen en pantalla.

Ejemplo:

```
eq = scene.text.equation("E = ", part("mass", "m"), " c^2")
scene.play([eq["mass"].animate.brace("masa")])
```

38.1.25. TextSelectionAnimation.annotate

MÉTODO `annotate(label: str, offset: tuple[float, float] = (0.0, 0.6)) -> Anim`

Coloca `label` a `offset` del centro de la selección con una línea guía. La línea y la etiqueta permanecen; un desplazamiento no finito lanza `ValueError`.

Ejemplo:

```
eq = scene.text.equation("E = m ", part("c", "c"), "^2")
scene.play([eq["c"].animate.annotate("velocidad de la luz", offset=(0, 0.8))])
```

38.1.26. TextSelectionAnimation.marker

MÉTODO `marker(color: ColorLike | None = None, *, skew: float = 0.05, blend: Literal["normal", "multiply"] = "normal", opacity: float = 0.45, padding: float | None = None) -> Anim`

Pasa un rotulador detrás de cada línea que abarca la selección. Cada banda cubre la altura de los glifos de su línea más `padding` y crece desde la izquierda; las líneas se suceden y se reparten la duración según su longitud. Las bandas quedan detrás de los glifos pero delante de lo creado antes del texto, y permanecen. Un `blend` desconocido, un `skew` no finito, una opacidad fuera de `[0, 1]` o un `padding` negativo lanzan `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`color`	`Color | None`	`None`	Color del rotulador; por defecto un amarillo cuyo alfa se multiplica por `opacity`.
`skew`	`float`	`0.05`	Inclinación de cada banda en radianes (positivo sube hacia la derecha), limitada para cubrir líneas largas.
`blend`	`str`	`"normal"`	`"normal"` pinta la banda encima de lo que tiene debajo; `"multiply"` la multiplica con el fondo o la tarjeta, como tinta sobre papel (sobre negro no se ve). En ambos casos va detrás de los glifos.
`opacity`	`float`	`0.45`	De 0 a 1.
`padding`	`float | None`	`None`	Margen en unidades del mundo; `None` usa el 10 % de la altura de línea.

Ejemplo:

```
from gaanim import Scene
scene = Scene(frame=(16, 9), background="#f8f5ee")
quote = scene.text("Lo que no se mide no se puede mejorar", color="#1f2937").move_to(0, 0)
scene.play([quote.words[4:9].animate.marker(skew=0.04).duration(1.2)])
scene.render()
```

38.1.27. TextSelectionAnimation.morph_to / copy_to

MÉTODO `morph_to(target: TextSelection) · copy_to(target: TextSelection) -> Anim`

Transiciones locales entre partes: `morph_to` sustituye la selección por el destino y `copy_to` conserva el origen mientras una copia viaja hasta el destino.

Ejemplo:

```
energy = scene.text("$E = ", part("mass", "m"), " c^2$").move_to(0, 1)
momentum = scene.text("$p = ", part("mass", "m"), " v$").move_to(0, -1)
scene.play([energy["mass"].animate.copy_to(momentum["mass"]).duration(0.8)])
```

Las animaciones de selección resuelven el código matemático escrito a los glifos que emitió Typst. Primero se busca el fragmento literal; si no aparece, el analizador matemático de Typst convierte nombres de símbolo (`theta`, `sum`, `arrow.r.long`), atajos (<=) y primas (`theta'`) al mismo Unicode que la ecuación renderizada.

38.1.28. Animación de texto completo

La escritura por grupos, los revelados con máscara, el desenfoque de entrada, el tracking, la máquina de escribir y el efecto de decodificación están en Animaciones (`Anim.write`, `Anim.reveal`, `Anim.conceal`, `Anim.blur_in`, `Anim.tracking`, `Anim.typewriter`, `Anim.scramble`). Aquí están el animador de rango, que los generaliza, y el tracking inmediato.

38.1.29. Text.animator

MÉTODO `animator(by: Literal["grapheme", "word", "line", "part"] = "grapheme", shape: Literal["square", "ramp", "smooth", "ease_in", "ease_out", "triangle", "round"] = "smooth", order: Literal["forward", "reverse", "center", "random"] = "forward", seed: int = 0) -> TextAnimator`

Selector de rango por glifo al estilo de los *Text Animators* de After Effects. Rust evalúa la influencia de cada unidad a partir de su índice y la forma del selector, así que un seek coincide

con la reproducción y no se llama a Python por fotograma. `reveal`, `conceal` y `blur_in` son preajustes de este animador. Nombres inválidos lanzan `ValueError`; sobre un objeto que no es `Text`, `TypeError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`by`	`str`	`"grapheme"`	Unidad: `grapheme`, `word`, `line` (explícita) o `part`. La puntuación se une a su vecina.
`shape`	`str`	`"smooth"`	Perfil del selector. `square`, `ramp`, `smooth`, `ease_in` y `ease_out` llevan cada unidad del estado «fuera» al reposo (revelado); `triangle` y `round` suben al estado «fuera» y vuelven (ola).
`order`	`str`	`"forward"`	Qué unidad alcanza primero el rango: `forward`, `reverse`, `center` o `random`.
`seed`	`int`	`0`	Fija la permutación de `order="random"`.

Ejemplo:

```
from gaanim import Scene, parallel

scene = Scene(frame=(16, 9), background="#0f172a")
title = scene.text("Tipografía cinética", role="title").move_to(0, 0)
wave = title.animator(by="grapheme", shape="smooth", order="forward", seed=0)
wave.set(offset=(0, -0.4), opacity=0.0, scale=0.6, rotation=0.2)  # estado "fuera"
scene.play([wave.animate.sweep().duration(1.2)])  # recorre el rango 0 → 1

bob = title.animator(by="word", shape="round").set(offset=(0, 0.3))
scene.play(parallel(bob.animate.sweep().duration(1.0)))  # una ola que pasa
scene.render()
```

38.1.30. TextAnimator.set / animate

MÉTODO

Define el estado «fuera» (influencia 1) y devuelve el animador. Los valores omitidos conservan lo definido antes; los canales nunca definidos quedan en reposo. Cada barrido captura el estado al llamar a `sweep()`. Valores inválidos lanzan `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`offset`	`(float, float) | None`	`None`	Desplazamiento en unidades de escena.
`opacity`	`float | None`	`None`	Opacidad absoluta de 0 a 1.
`scale / rotation`	`float | None`	`None`	Alrededor del centro de cada unidad; la rotación

			en radianes, menos de media vuelta.
`blur / tracking`	`float | None`	`None`	Sigma gaussiana y espacio extra entre glifos, en unidades de escena.
`color`	`Color | None`	`None`	Relleno sólido.

38.1.31. TextAnimatorAnimation.sweep

MÉTODO `sweep(start: float = 0.0, end: float = 1.0, *, stagger: Optional[float] = None) -> Anim`

Accesible como `animator.animate.sweep(...)`. Devuelve un `Anim` que mueve el rango de `start` a `end`: `0` está antes de la primera unidad y `1` después de la última, así que `sweep(1, 0)` lo recorre hacia atrás. `stagger` es el retraso en segundos entre unidades (`None` lo adapta) y `easing` suaviza cada unidad. Funciona en `scene.play`, `parallel`, `sequence` y `stagger`; los barridos de entrada mantienen su primer fotograma hasta empezar.

38.1.32. Text.tracking

MÉTODO `tracking(value: float) -> Self`

Fija al instante `value` unidades de escena extra entre glifos vecinos, sin recomponer el párrafo: cada fila crece desde su borde izquierdo, su centro o su borde derecho según la alineación. `0` restaura el espaciado. Anímalo con `animate.tracking(value)`.

Ejemplo:

```
title = scene.text("Tipografía", role="title").tracking(0.4)
scene.play([title.animate.tracking(0.0).duration(1.2)])
```

38.1.32.1. Texto sobre una trayectoria

38.1.33. Typography.on_path

FÁBRICA `on_path(content: TextContent, path: Drawable, *, align: Literal["start", "center", "end"] = "start", orient: bool = True, reverse: bool = False, offset: float = 0.0, role: Optional[TextRole] = None, style: Optional[TextStyle] = None, font: Optional[str] = None, size: Optional[float] = None, weight: Optional[int] = None, italic: Optional[bool] = None, color: Optional[ColorLike] = None, opacity: Optional[float] = None, letter_spacing: Optional[float] = None, word_spacing: Optional[float] = None, markup: Optional[bool] = None) -> Text`

Cada glifo conserva su forma y su espaciado: su centro sobre la línea base se coloca en la trayectoria a la misma longitud de arco que tenía en la línea. Los glifos quedan a la izquierda del sentido de avance; un círculo avanza en sentido antihorario, así que el texto se lee por dentro, y `reverse=True` lo pone por fuera. En una trayectoria cerrada el texto da la vuelta; en una abierta, los glifos que pasan un extremo siguen en línea recta. La trayectoria se lee en coordenadas de

escena donde está `path` en ese momento y pasa a ser del texto: moverlo la arrastra, y cambiar `path` después no mueve los glifos. `path` sigue siendo un drawable normal. Acepta los argumentos de estilo de `scene.text`. Un `align` desconocido, un `offset` no finito, un `path` de otra escena o un grupo (que no tiene trayectoria propia) y un contenido sin caracteres visibles lanzan `ValueError`.

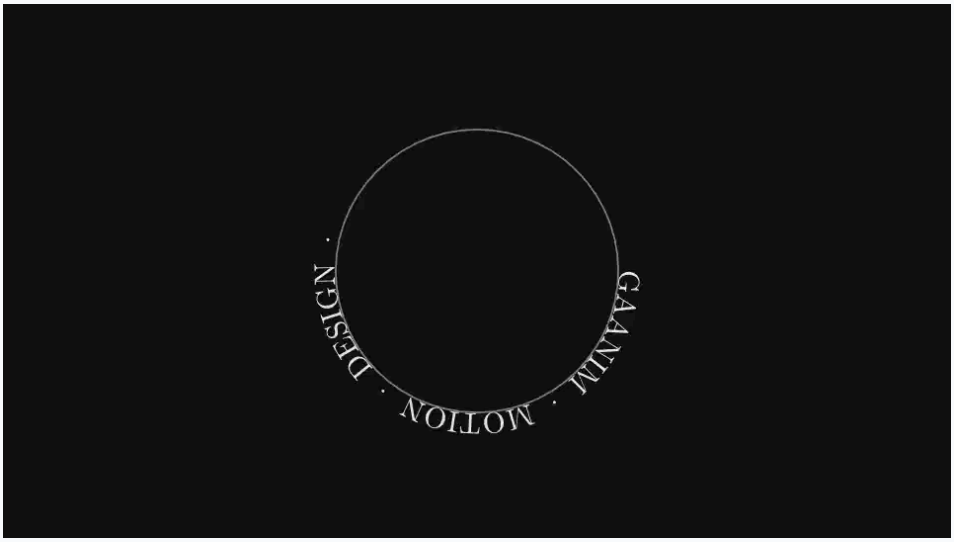
Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`content`	`TextContent`	obligatorio	Una línea de texto; nunca se parte.
`path`	`Drawable`	obligatorio	Drawable cuya trayectoria siguen los glifos.
`align`	`str`	`"start"`	`"start"`, `"center"` o `"end"`: el texto empieza, se centra o termina en la trayectoria.
`orient`	`bool`	`True`	Gira cada glifo con la dirección de la trayectoria; con `False` quedan derechos.
`reverse`	`bool`	`False`	Recorre la trayectoria del final al principio.
`offset`	`float`	`0.0`	Desplazamiento inicial, como fracción de la longitud de la trayectoria.

Devuelve: `Text` — Texto cuyos glifos siguen la trayectoria.

Ejemplo:

```
from gaanim import *

scene = Scene(frame=(16, 9))
circle = scene.geometry.circle(2.4).no_fill().stroke(GRAY, 0.03)
ring = scene.text.on_path("GAANIM · MOTION · DESIGN · ", circle, reverse=True, size=0.5)
scene.play([ring.animate.path_offset(1.0).duration(4).easing(Easing.LINEAR)])
scene.render()
```



38.1.34. Text.path_offset

MÉTODO `path_offset(value: float) -> Self`

Desplaza al instante un texto creado con `on_path` por su trayectoria, como fracción de su longitud. Declarado antes del primer `play` es el desplazamiento inicial; después corta en el cursor. Anímalo con `animate.path_offset(value)`.

Ejemplo:

```
wave = scene.geometry.path([("move", [(-6, 0)]), ("cubic", [(-2, 3), (2, -3), (6, 0)])])
label = scene.text.on_path("una ola de texto", wave, align="center").path_offset(-0.2)
scene.play([label.animate.path_offset(0.2).duration(2)])
```

38.1.34.1. Transiciones estructurales

`text.animate.transform_to(target)` (ver `Anim.transform_to` en Animaciones) empareja las partes semánticas de dos textos: los términos compartidos viajan a su nuevo sitio y el resto aparece o desaparece. Cada glifo conserva su color, también los de `part()`, y al terminar queda visible el destino. La transición usa la misma duración para el texto y para el reflow de su Layout. Un destino de otra escena lanza `ValueError`. Entre textos gestionados por Layouts distintos (por ejemplo, celdas de dos matrices) la transformación es un morph de forma única: el origen conserva su identidad y adopta el contorno, el color y la línea base del destino.

```
from gaanim import GOLD, Scene, part
scene = Scene(frame=(16, 9), background="#0f172a")
before = scene.text("$x + ", part("obsolete", "3"), " = 7$").move_to(0, 0)
after = scene.text("$x = ", part("result", "4", color=GOLD), "$").move_to(0, 0)
scene.play([before.animate.write().duration(0.8)])
scene.play([before["obsolete"].animate.cancel().duration(0.5)])
scene.play([before.animate.transform_to(after).duration(0.8)])
scene.render()
```

38.1.35. Cambiar el contenido

Sustituye el contenido de un texto sin crear otro objeto.

38.1.36. Text.become

MÉTODO `become(*content: TextContent, role: Optional[TextRole] = None, style: Optional[TextStyle] = None, flow: Optional[TextFlow] = None, markup: Optional[bool] = None) -> None`

Reemplaza el contenido conservando la identidad del `Text`, incrementa su versión estructural y pide reflow a su Layout. Si el texto es libre y ya se mostró, el nuevo contenido lo reemplaza en el cursor, colocado como se declaró el texto (`move_to`, `place`), y las animaciones posteriores del `Text` actúan sobre él. Es una llamada inmediata de autoría, no un `Anim`: no la pongas en `scene.play`. `markup=None` conserva el modo de marcado actual. Contenido o delimitadores inválidos lanzan `ValueError`.

Ejemplo:

```
copy = scene.text("Calculando…")
scene.wait(0.5)
copy.become("Resultado: ", part("value", "$42$", color=GOLD))
```

38.1.37. Documentos Typst y código

Para lo que no cabe en un `Text`: documentos Typst completos y bloques de código. Para medir cualquier objeto, texto incluido, usa `bounds()` (ver Drawable).

38.1.38. Typography.typst

FÁBRICA `typst(source: str | os.PathLike[str], *, width: Optional[str | float | int] = None) -> Drawable`

Compila un documento Typst arbitrario (tablas con celdas combinadas, estructuras matemáticas propias, paquetes `@preview/…`) como un `Drawable`. Conserva sus proporciones y se escala para que su texto de 11 pt mida lo mismo que el rol `body`. No ofrece las selecciones de `Text`. Su texto toma el color del rol `body` del tema, como `scene.text`, y también los trazos que Typst dibuja en negro por defecto (tablas, `rect`, `line`, formas de CeTZ). El documento empieza con `theme`, un diccionario con los colores del tema por nombre (`theme.accent`, `theme.foreground`, `theme.panel`, `theme.rule`…): los paquetes pensados para papel blanco eligen ellos mismos el negro y el blanco, así que pásales estos colores por sus opciones (ver Colores del tema en Typst). Los trazos que continúan uno a otro, como una curva que un paquete de gráficas dibuja muestra a muestra, son una sola pieza para `write`. Una fuente vacía lanza `ValueError` y un archivo ilegible, `RuntimeError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`source`	`str | os.PathLike`	obligatorio	Marcado Typst en línea o la ruta de un archivo `.typ` (relativa a la carpeta de assets).
`width`	`str | float | int | None`	`None`	Ancho de página Typst antes de escalar: `"16cm"`, `"800pt"` o un número en puntos. Sin él, la página se ajusta al contenido:

			`#align(center)` centra dentro del contenido y los párrafos solo cortan línea donde la fuente lo indica. Pásalo para que un párrafo se ajuste o para columnas `1fr`.

Ejemplo:

```
from gaanim import Scene
scene = Scene(frame=(16, 9), background="#0f172a")
tbl = scene.text.typst('#table(columns: 2, [*Método*], [*Error*], [Base], [0.18], [GPU], [0.04])')
scene.play([tbl.animate.fade_in().duration(0.6)])
scene.render()
```

También carga archivos:

```
from pathlib import Path
title = scene.text.typst(Path("assets/title.typ"))
```

38.1.38.1. Colores del tema en Typst

Cada documento empieza con `theme`, los colores del tema por nombre, los mismos que `scene.canvas.color(...)`. El texto ya toma el color del tema; úsalos para lo que el documento pinta por su cuenta:

```
from gaanim import Scene
scene = Scene()
table = scene.text.typst("""
#table(
  columns: 3,
  stroke: 0.6pt + theme.rule,
  fill: (x, y) => if y == 0 { theme.panel },
  [*x*], [*y*], [*x + y*],
  [1], [2], [#text(fill: theme.accent)[3]],
)
""")
scene.play([table.animate.write()])
scene.render()
```

Los paquetes de Typst Universe están pensados para papel blanco y eligen ellos mismos el negro y el blanco de ejes, marcas y fondos de etiqueta. Dales los colores del tema por sus opciones de estilo, como en `examples/text/typst_packages.py`:

```
#import "@preview/simple-plot:1.0.0": plot
#plot(
  style: (
    axis: (stroke: theme.rule + 0.8pt, arrow: (symbol: "stealth", fill: theme.rule)),
    ticks: (stroke: theme.rule + 0.6pt, label-fill: theme.muted, label-bg: none),
    labels: (fill: theme.foreground, bg: none),
  ),
  (fn: x => x * x, stroke: theme.accent + 1.2pt),
)
```

38.1.39. Typography.code

FÁBRICA `code(source: str, *, language: str = "text", width: float = 7.6, height: float = 3.0, font_size: float = 0.2, background: Optional[ColorLike] = None, color: Optional[ColorLike] = None, accent: Optional[ColorLike] = None, theme: str | os.PathLike[str] | None = None, highlight: bool = True, font: Optional[str] = None) -> Drawable`

Panel de código: el nombre del lenguaje sobre una regla y el código debajo, alineado a la izquierda, con sus tokens resaltados. Las palabras clave, las cadenas, los números, los comentarios, las funciones y los tipos toman los colores de código del tema: los temas de editores (`dracula`, `nord`, `solarized-*`, `gruvbox-dark`, `tokyo-night`, `catppuccin-*`) usan su esquema original, y `technical`, `presentation`, `paper` y los temas propios los derivan de la paleta (palabras clave en `accent`, comentarios en `muted`). Los tokens `code`, `code_keyword`, `code_string`, `code_number`, `code_comment`, `code_function` y `code_type` de `Theme(colors=...)` los sustituyen. Sobre un `background` en el que los colores del tema no se leen (un panel oscuro en un tema claro, o al revés), el texto y cada token se ajustan hasta leerse; los que ya se leen no cambian. Cada carácter sigue siendo un glifo propio, coloreado o no. Una fuente vacía, un lenguaje con caracteres distintos de letras y dígitos ASCII, `-` o `_`, una `font` vacía o un tamaño no

positivo lanzan `ValueError`; un `theme` inexistente lanza `FileNotFoundError`, y uno que no es un tmTheme, `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`source`	`str`	obligatorio	El código. Los tabuladores y los saltos de línea se conservan.
`language`	`str`	`"text"`	Nombre o extensión del lenguaje: `"python"`, `"rust"`, `"js"`, `"typst"`, `"json"`… Uno desconocido, como `"text"`, queda sin resaltar.
`width`	`float`	`7.6`	Ancho del panel.
`height`	`float`	`3.0`	Alto del panel.
`font_size`	`float`	`0.2`	Tamaño base del texto; el código se dibuja al 80 %.
`background`	`ColorLike | None`	`None`	Fondo del panel; por defecto, el `panel` del tema (o el de `theme`).
`color`	`ColorLike | None`	`None`	Color del texto que no se resalta: nombres, operadores y puntuación; por defecto, el `foreground` del tema (o el de `theme`), o su `background` si el `foreground` no se lee sobre el fondo del bloque.
`accent`	`ColorLike | None`	`None`	Color del borde, la regla y el nombre del lenguaje; por defecto, el `accent` del tema, aclarado u oscurecido si no se lee sobre un `background` propio.
`theme`	`str | os.PathLike | None`	`None`	Archivo `.tmTheme` (el formato de TextMate, Sublime Text y bat) con los colores de los tokens, resuelto como los demás assets.
`highlight`	`bool`	`True`	Con `False`, todo el código va en `color`.
`font`	`str | None`	`None`	Familia monoespaciada. Sin ella, o si no está instalada, JetBrains Mono, incluida en

			Gaanim, para que el bloque se vea igual en cualquier equipo.

Ejemplo:

```
from gaanim import Scene
scene = Scene(frame=(16, 9))
snippet = scene.text.code(
    "def area(radius):\n"
    "    # circle area\n"
    "    return 3.14159 * radius ** 2",
    language="python",
    width=8,
    height=2.4,
    font_size=0.3,
)
scene.play([snippet.animate.fade_in().duration(0.5)])
scene.render()
```

Con otro esquema, sin resaltar o con otra fuente:

```
papel = scene.text.code("let x = 42; // respuesta", language="rust", theme="assets/
codigo.tmTheme")
plain = scene.text.code("x = 42", language="python", highlight=False)
fira = scene.text.code("x = 42", language="python", font="Fira Code")
```

38.1.40. Integración con Layout

Un `Text` libre y el mismo `Text` dentro de una fila, columna, grid o stack usan el mismo medidor. `wrap="auto"` lo hace sensible al ancho ofrecido; un ajuste numérico queda además limitado por ese ancho. Un texto que cabe conserva sus líneas naturales, y uno que se ajusta mantiene las líneas con las que se midió: un contenedor que abraza su contenido (`"hug"`) nunca lo reparte en más líneas. Los cambios de métrica, `become` y `animate.transform_to` invalidan la medición y piden reflow; los énfasis transitorios (`indicate`, `pulse`, `wiggle`, `wave`, `highlight`, `focus`) no.

Una caja decide dónde descansa un `Text` hijo: colocarlo de forma inmediata con `move_to`, `shift_by` o `next_to` lanza `LayoutOwnershipError`; ajusta su `text.item(...)` o las propiedades de la caja. Todas

las animaciones de `.animate`, de posición incluidas, siguen disponibles y parten de ese sitio. Consulta Layout.

38.1.41. Errores y casos límite

- `TypeError`: contenido que no es `str` ni `TextPart`, o claves de consulta del tipo equivocado.
- `ValueError`: contenido vacío, matemáticas desequilibradas, partes hermanas repetidas, rol, estilo o flujo inválidos, agrupación u orden inválidos, rangos con paso distinto de 1.
- `KeyError`: una cadena que no es parte semántica ni texto del `Text`.
- `IndexError`: índice fuera de rango o rango vacío.
- `LayoutOwnershipError`: posicionar a mano un texto gestionado, o una transición entre escenas o propietarios incompatibles.

CAPÍTULO

39. Layout

scene.layout: cajas, filas, columnas, grids, capas, estilos, clases, zonas, animación de estructura y restricciones

`scene.layout` compone la escena con **cajas**, igual que CSS compone una página con `div`s: cada caja mide su contenido, reparte el espacio entre sus hijos y los coloca, con padding, gap, borde, radio, sombra y recorte. Las **zonas** dividen el fotograma en regiones con nombre para colocar objetos que siguen siendo libres. La guía Layout enseña a usarlas paso a paso; esta página es la referencia de cada pieza.

```
from gaanim import BoxStyle, Scene

scene = Scene(frame=(16, 9), theme="paper", background="#f1f5f9", margin=0.6)
L = scene.layout
L.classes(
    card=BoxStyle(padding="26px", gap="12px", radius="22px", background="white",
                  shadow={"color": "#0f172a22", "y": "-8px", "blur": "26px"}),
    tag=BoxStyle(padding=("4px", "12px"), radius="full", font_size="18px",
                 background="#e0e7ff", color="#3730a3"),
)
article = L.box(
    L.row(L.box("Layout", class_="tag"), L.box("CSS", class_="tag"), gap="8px"),
    L.box("Cajas que se colocan solas", font_size="40px", weight=700),
    L.box("Padding, gap, grids, estilos y zonas: la escena se compone sin coordenadas "
          "y se reorganiza al animar.", font_size="24px", color="#475569", line_spacing=1.4),
    class_="card", max_width="760px",
)
scene.play([article.animate.fade_in().duration(0.6)])
scene.render()
```


El modelo tiene tres capas, como en CSS:

1. **La caja** decide el flujo (`direction`), su tamaño, el espacio interior (`padding`, `gap`) y cómo reparte y alinea a sus hijos (`justify`, `align`).

2. **Cada hijo** declara con `.item(...)` cómo consume la caja que recibe: crecer, alinearse, ocupar celdas, márgenes, ajuste.
3. **El contenido** se mide dentro de esa caja: el texto recompone sus líneas y las imágenes se ajustan con `fit`.

39.1.1. Unidades

Todas las propiedades aceptan las mismas unidades:

- **Números**: unidades de escena (el fotograma por defecto mide 16 × 9).
- **`"Npx"`**: píxeles de diseño. El fotograma mide `design_resolution` (1080 por defecto) píxeles de alto, así que `"24px"` se ve igual en cualquier resolución de salida: `Scene(design_resolution=1920)` cambia la referencia.
- **Tokens del tema**: cualquier otro nombre, como `"space_md"` o `"page_padding"`, se lee de `theme.layout`.
- **Tamaños** (`width`, `height`, `basis`): además `"N%"` del padre, `"hug"` (o `"auto"`, el tamaño del contenido) y `"fill"` (el espacio libre).
- **Pistas de grid**: además `"auto"` y fracciones `"Nfr"`.
- **Relleno y margen**: uno, dos, tres o cuatro valores, como el atajo de CSS.

Una unidad desconocida, un tamaño negativo o un valor no finito lanzan `ValueError`.

39.1.2. Cajas

39.1.3. LayoutBuilder.box

FÁBRICA `box(*children: Any, style: Optional[BoxStyle] = None, **props: Any) -> Box`

Crea una caja. Sus hijos son objetos, cajas o cadenas (que se convierten en `Text` con la tipografía de la caja); las listas se aplanan y `None` se ignora. La dirección por defecto es `"column"`; `direction` acepta `"row"`, `"grid"` y `"stack"`. Las propiedades se describen en la sección Propiedades. `style=` recibe un `BoxStyle` y `class_=` nombres de clases.

Ejemplo:

```
pill = scene.layout.box("Nuevo", padding=("6px", "16px"), radius="full",
                        background="#4f46e5", color="white", font_size="24px")
```

39.1.4. LayoutBuilder.row / column

FÁBRICA

Atajos de `box(direction="row")` y `box(direction="column")`. Su nombre manda sobre el `direction` de un `style=` o de una clase; solo un `direction` escrito en la llamada lo cambia (igual en `grid` y `stack`). En una fila el eje principal es horizontal y `align` actúa en vertical; en una columna, al revés. Con `wrap=True` los hijos que no caben pasan a la línea siguiente.

Ejemplo:

```
tags = scene.layout.row(*[scene.layout.box(t, padding="6px", background="#e0e7ff") for t in
                          ("rust", "gpu", "vello", "taffy", "python")],
                        gap="8px", wrap=True, max_width="300px")
```

39.1.5. LayoutBuilder.grid

FÁBRICA `grid(*children: Any, style: Optional[BoxStyle] = None, **props: Any) -> Box`

Caja de rejilla. `columns` y `rows` son un número de pistas o una lista de pistas (`"160px"`, `"25%"`, `"auto"`, `"1fr"`). `gap`, `row_gap` y `column_gap` separan las pistas; `auto_flow="column"` rellena por columnas. Los hijos se colocan en orden, o en la celda que pidan con `item(row=..., column=..., row_span=..., column_span=...)`, y se estiran por defecto (`align="stretch"`).

Ejemplo:

```
bento = scene.layout.grid(
    scene.layout.box("A", style=cell).item(row_span=2),
    scene.layout.box("B", style=cell).item(column_span=2),
    scene.layout.box("C", style=cell),
    scene.layout.box("D", style=cell),
    columns=["160px", "1fr", "2fr"], gap="12px", within="safe", width="fill",
)
```

39.1.6. LayoutBuilder.stack

FÁBRICA `stack(*children: Any, style: Optional[BoxStyle] = None, **props: Any) -> Box`

Caja de capas: todos los hijos comparten la misma área y se superponen en orden. Cada hijo se coloca con `item(anchor=..., offset=...)` (centro por defecto) o se ajusta con `item(fit=...)`. Ideal para un texto sobre una imagen, una insignia en una esquina o un fondo.

Ejemplo:

```
cover = scene.layout.stack(
    photo.item(fit="cover"),
    scene.layout.box("Portada", padding="8px", background="white").item(anchor="bottom_left",
                                                                        offset=("16px", "16px")),
    width="420px", height="260px", radius="16px", clip=True,
)
```

39.1.7. Propiedades

Las mismas propiedades valen para `box`, `row`, `column`, `grid`, `stack`, `BoxStyle` y `Box.set`. Un nombre desconocido lanza `TypeError`.

Contenedor

- `direction`: `"column"`, `"row"`, `"grid"` o `"stack"`.
- `width`, `height`, `min_width`, `max_width`, `min_height`, `max_height`, `aspect_ratio`.
- `padding`, `gap`, `row_gap`, `column_gap`.
- `align` (eje transversal): `start`, `center`, `end`, `stretch`, `baseline`. `justify` (eje principal): `start`, `center`, `end`, `between`, `around`, `evenly`.
- `wrap` (filas y columnas), `columns`, `rows`, `auto_flow` (grids).
- `within`: solo en la raíz; `"safe"` (el fotograma menos el margen, por defecto) o `"frame"` (a sangre).

Hijo (también en `Drawable.item`)

- `grow`, `shrink`, `basis`: reparto del eje principal. `width="fill"` en una fila equivale a `grow=1`. En una caja que se ajusta a su contenido no hay espacio libre que repartir: un hijo que crece mide su contenido aunque tenga `basis`, y un texto se ajusta al ancho que le deje el padre.
- `align_self`: alineación propia en el eje transversal (en un grid, la vertical dentro de la celda).
- `margin`: longitudes o `"auto"`, que se queda con el espacio libre.
- `row`, `column`, `row_span`, `column_span`: celda en un grid.
- `fit`: `none`, `contain`, `cover`, `stretch`, `scale_down`. `cover` recorta siempre lo que sale de la celda, sin `clip=True`, como `object-fit: cover` en CSS, también si después se mueve la caja.
- `anchor`, `offset`: posición en una capa o en su celda.
- `absolute`: fuera del flujo; no ocupa espacio.

Decoración

- `background`, `border`, `border_width`, `radius` (`"full"`: píldora), `shadow` (`True` o `{"color", "x", "y", "blur"}`), `clip`. `shadow=True` usa el color `shadow` del tema, que también es el `color` por defecto del diccionario: negro suave sobre fondos claros y casi opaco sobre fondos oscuros, donde uno suave no se vería.

Tipografía (la heredan las cadenas hijas)

- `color`, `font`, `font_size`, `weight`, `italic`, `role`, `text_align`, `line_spacing`, `letter_spacing`, `max_lines`, `overflow`, `markup`, `text_box`. `markup` es `False` por defecto: `$` y `*` se escriben tal cual.
- `text_box` elige la caja con que se mide y coloca el texto, como `text-box` en CSS: `"line"` (por defecto) ocupa líneas completas, de la ascendente a la descendente, así que textos del mismo estilo comparten altura; `"cap"` va de la altura de las mayúsculas a la línea base (de un texto de una línea), para que las mayúsculas toquen el borde de la caja como en un diseño calcado; `"ink"` se ajusta a la tinta de los glifos. Las fórmulas `$…$` usan siempre su tinta.

39.1.8. Estilos y clases

39.1.9. BoxStyle

CLASE `BoxStyle(**props)`

Conjunto inmutable de propiedades, como una regla CSS. Se aplica con `style=`. La prioridad es: clases, después `style=` y al final las propiedades escritas en la llamada.

Ejemplo:

```
pill = BoxStyle(padding=("4px", "12px"), radius="full", background="#4f46e5", color="white")
badge = scene.layout.box("v2", style=pill)
```

39.1.10. BoxStyle.but

MÉTODO `but(**props: Any) -> BoxStyle`

Devuelve una copia con algunas propiedades cambiadas. `to_dict()` devuelve las propiedades.

Ejemplo:

```
ghost = pill.but(background="#e0e7ff", color="#3730a3")
```

39.1.11. LayoutBuilder.classes

MÉTODO `classes(**styles: BoxStyle) -> None`

Registra estilos con nombre para la escena. `class_="pill soft"` aplica varias en orden, y la última gana. Una clase no registrada lanza `ValueError`.

Ejemplo:

```
scene.layout.classes(card=BoxStyle(padding="20px", radius="16px", background="white"))
panel = scene.layout.box("Hola", class_="card")
```

39.1.12. Reglas por hijo

39.1.13. Drawable.item

MÉTODO `item(*, duration: Optional[float] = None, advance: bool = True, **props: Any) -> Self`

Declara cómo se coloca un objeto dentro de la caja que lo contiene (las propiedades de **Hijo**) y devuelve el propio objeto, así que se usa en línea. `width`/`height` sustituyen al tamaño propio del objeto en el layout. Si el objeto ya está en una caja, esta se reorganiza, animada con `duration`.

Ejemplo:

```
row = scene.layout.row("Carga", bar.item(grow=1, height="12px"), "72%", gap="16px",
                       align="center", width="fill")
```

39.1.14. Cajas vivas

Una `Box` es un `Drawable`: `move_to`, `scale_by`, `animate.fade_in()` o `place` actúan sobre la caja entera, y su escala y su giro se aplican a los hijos sobre los suyos, como una transformación CSS. Decide la posición de reposo de sus hijos, que siguen aceptando cualquier animación de `.animate`: parten de ese sitio, no mueven a sus hermanos y conservan su desplazamiento cuando la caja se reorganiza. Los cambios de estructura se registran en la línea de tiempo; con `duration=` se animan (los hijos se deslizan, lo que entra aparece con un fundido y lo que sale se desvanece) y avanzan el cursor como `scene.play`. Con `advance=False` el siguiente cambio empieza a la vez.

39.1.15. Box.add / remove / replace

MÉTODO

`add(hijo, at=None)` inserta un hijo (o una cadena) y lo devuelve; con `duration`, entra con un fundido, salvo un hijo que viene de otra caja (sacado con `detach`), que se desliza desde donde está. `remove(hijo)` lo quita con un fundido; `replace(viejo, nuevo)` pone `nuevo` en su lugar, heredando sus reglas de `item` si no trae las suyas. Un índice fuera de rango lanza `IndexError`.

Ejemplo:

```
board.add(scene.layout.box("Nueva", style=chip), at=1, duration=0.6)
board.remove(board[3], duration=0.6)
board.replace(board[0], scene.layout.box("Otra", style=chip), duration=0.6)
```

39.1.16. Box.set

MÉTODO `set(*, duration: Optional[float] = None, advance: bool = True, style: Optional[BoxStyle] = None, **props: Any) -> Self`

Cambia cualquier propiedad de la caja (las mismas que al crearla, también `style=`) y la reorganiza; con `duration`, los hijos se mueven a su sitio nuevo en ese tiempo. Los cambios de decoración se aplican también. `color` recolorea los textos que la caja creó a partir de cadenas, como al crearla, y con `duration` lo anima; los textos y cajas que le pasaste como hijos conservan su color. El resto de la tipografía (`font_size`, `weight`…) vale para las cadenas que se añadan después.

Ejemplo:

```
grid.set(columns=2, gap="30px", duration=0.8)
```

39.1.17. Box.detach

MÉTODO `detach(child: Drawable, *, duration: Optional[float] = None, advance: bool = True) -> None`

Saca a un hijo de la caja sin ocultarlo: conserva su posición, deja de ocupar sitio y vuelve a aceptar `move_to` y las animaciones de posición. Desde ese instante ya no sigue a la caja si esta se mueve, escala o se desvanece; antes sí. `otra.add(hijo)` lo mete en otra caja.

39.1.18. Box.children / background

PROPIEDAD

`children` es la lista de hijos directos (también `len(box)`, `box[i]` e iteración); `background` es el objeto que dibuja fondo, borde y radio, o `None`.

39.1.19. Box.walk

MÉTODO `box.walk(*, boxes=False) -> list[Drawable]`

Devuelve una lista con las piezas de la caja a cualquier profundidad, en orden de dibujo: de cada caja, primero su fondo y luego sus hijos, entrando en las cajas hijas. Con `boxes=True` incluye también cada caja anidada, antes de sus piezas; la propia caja no aparece nunca. Sirve para revelar una tarjeta pieza por pieza sin recorrer el árbol a mano ni olvidar los fondos.

Ejemplo:

```
for piece in card.walk():
    scene.play(piece.animate.fade_in().duration(0.1))
scene.render()
```

39.1.20. Box.stagger

MÉTODO `box.stagger(make, *, boxes=False, each=0.1, total=None, origin=None, grid=None, easing=None, seed=0) -> Composition`

Anima cada pieza de `walk(boxes=boxes)` como un solo `stagger`. `make` recibe la pieza y devuelve su animación, o `None` para saltarla; el resto de argumentos son los de `stagger`. Lanza `TypeError` si `make` no es invocable y `ValueError` si no queda ninguna pieza.

Ejemplo:

```
scene.play(card.stagger(lambda piece: piece.animate.fade_in(), each=0.08))
scene.render()
```

39.1.21. Box.find / find_all

MÉTODO `box.find(*, type=None, where=None, text=None, boxes=True) -> Drawable | None`

Buscan entre las piezas de `walk(boxes=boxes)`: `find` devuelve la primera que cumple, o `None`; `find_all` las devuelve todas en orden de dibujo. Una pieza cumple si es instancia de `type` (una clase o una tupla de clases), es un `Text` cuyo `content` es igual a `text`, y `where(pieza)` es verdadero; un filtro omitido lo acepta todo. Por defecto también buscan entre las cajas anidadas. Lanzan `TypeError` si `where` no es invocable.

Ejemplo:

```
title = card.find(text="Título")
letters = card.find_all(type=Text, where=lambda piece: piece is not title)
scene.play(title.animate.fade_in())
scene.render()
```

39.1.22. Box.each

MÉTODO `box.each(make, *, type=None, where=None, text=None, boxes=False) -> Box`

Llama a `make(pieza)` para cada pieza que cumple los filtros (los de `find_all`, pero `boxes` vale `False` por defecto) y devuelve la caja, así que se encadena. Sirve para aplicar un ajuste inmediato a toda una tarjeta en una línea. Lanza `TypeError` si `make` no es invocable.

Ejemplo:

```
card.each(lambda piece: piece.fill("#0f172a"), type=Text)
scene.render()
```

39.1.23. Box.reveal

MÉTODO `reveal(*, each=0.06, duration=0.4, direction=None, distance=0.3, rules="grow", rules_from=None) -> Composition`

Es la política que casi toda diapositiva pide: los fondos y demás objetos aparecen con un fundido (también un `rolling_number` o un `progress_ring`, aunque su propio `animate` anime el valor), los textos entran deslizándose y los filetes crecen. Una caja sin hijos que no dibuja nada no se anima. Para elegir la animación de cada pieza usa `Box.cascade` o `Box.stagger`. Un `rules` desconocido, una `duration` no positiva o una caja sin nada que mostrar lanzan `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`each`	`float`	`0.06`	Segundos entre el inicio de una pieza y el de la siguiente.
`duration`	`float`	`0.4`	Duración de cada pieza; debe ser positiva.
`direction, distance`	`Direction | None, float`	`Direction.UP, 0.3`	De dónde entran los textos y desde qué distancia, en unidades de escena.
`rules, rules_from`	`"grow" | "fade", Direction | None`	`"grow", Direction.LEFT`	Cómo entran las cajas sin

			hijos que dibujan algo (filetes, barras): crecen desde `rules_from` o aparecen con un fundido.

Devuelve: `Composition` — Un `stagger` sobre las piezas.

Ejemplo:

```
scene.play(card.reveal())
scene.render()
```

39.1.24. Box.cascade

MÉTODO `box.cascade(*, boxes=False, each=0.1, total=None, origin=None, grid=None, easing=None, seed=0) -> BoxCascade`

Elige las piezas de `walk(boxes=boxes)` y devuelve un `BoxCascade`. Cada animación de `Drawable.animate` (`fade_in`, `grow_from_center`, `shift_by`…) se puede llamar sobre él con sus mismos argumentos y devuelve un `stagger` sobre todas las piezas; un `rolling_number` o un `progress_ring` se anima como dibujo, no su valor. El resto de argumentos son los de `stagger`. Lanza `ValueError` si la caja no tiene piezas y `AttributeError` si una pieza no tiene esa animación; para saltarte piezas usa `Box.stagger`.

Ejemplo:

```
scene.play(card.cascade(each=0.08).fade_in())
scene.render()
```

39.1.25. Box.move_child / swap / reverse

MÉTODO `box.move_child(child, to, *, duration=None, advance=True) · box.swap(a, b, ...) · box.reverse(...)`

Cambian el orden de los hijos: `move_child(hijo, a)` lo lleva a la posición `a` (negativa cuenta desde el final), `swap(a, b)` intercambia dos y `reverse()` invierte todos. Con `duration` los hijos se deslizan a su nuevo sitio; `advance=False` empieza el siguiente cambio a la vez. En un `stack` la

posición no cambia qué hijo se dibuja encima: para eso usa `z_index`. `move_child` lanza `ValueError` si el objeto no es hijo de la caja e `IndexError` si la posición no existe.

Ejemplo:

```
row.move_child(row[2], 0, duration=0.6)
row.swap(row[0], row[2], duration=0.6)
row.reverse(duration=0.6)
scene.render()
```

39.1.26. Box.reflow

MÉTODO `reflow(*, duration: Optional[float] = None, advance: bool = True) -> None`

Vuelve a calcular la disposición cuando un hijo cambió de tamaño por su cuenta. Mide cada hijo de la caja, a cualquier profundidad, con la escala y la rotación que tiene ahora: un hijo escalado con `scale_to` o `animate.scale_to` recibe sitio para su nuevo tamaño. Las demás reorganizaciones (`add`, `set`, un texto que cambia…) conservan el tamaño con el que se colocó, como un `transform` de CSS. Los cambios de texto y de estructura ya reorganizan solos.

39.1.27. LayoutOwnershipError

CLASE

Se lanza al colocar de forma inmediata (`move_to`, `shift_by`, `next_to`…, sin `.animate`) un hijo de una caja o al meter en una caja un objeto que ya pertenece a otra. Usa `item(offset=...)`, anima al hijo con `.animate`, mueve la caja entera o `detach` al hijo. `scale_to`, `rotate_to` y los demás ajustes de forma no lo colocan y sí funcionan sobre un hijo.

39.1.28. Zonas

Las zonas dividen una región en rectángulos con nombre. No adoptan objetos: colocan una vez (o animan) objetos que siguen siendo libres, así que una misma plantilla sirve para el fotograma, para una zona o para el interior de un objeto.

39.1.29. Zones.rows / columns / grid

FÁBRICA

Plantillas reutilizables. `rows(pistas)` y `columns(pistas)` cortan en una dirección; `grid(rows=..., columns=...)` en las dos. Aceptan `names=`, `gap=` (y `row_gap`, `column_gap` en `grid`) y `padding=`. Las pistas son las de los grids.

Ejemplo:

```
page = Zones.rows(["90px", "1fr", "60px"], names=["cabecera", "cuerpo", "pie"], gap="16px")
split = Zones.columns(["1fr", "2fr"], names=["lateral", "principal"], gap="24px")
```

39.1.30. LayoutBuilder.zones

MÉTODO `zones(template: Zones, *, within: Optional[Literal["safe", "frame"] | Zone | Drawable] = None) -> ZoneSet`

Aplica una plantilla y devuelve un `ZoneSet`. Por defecto usa el área segura, que respeta el `margin` de la escena; `within` acepta `"frame"`, una `Zone` o un objeto (su caja actual). `scene.layout.safe` y `scene.layout.frame` son zonas ya hechas.

Ejemplo:

```
z = scene.layout.zones(page)
title = scene.text("Título", size=0.5).place(z["cabecera"], anchor="left")
```

39.1.31. ZoneSet.fill

MÉTODO `fill(*objects: Drawable, anchor: Optional[AnchorName | Anchor] = None, fit: Optional[Fit] = None, padding: Optional[Padding] = None) -> None`

Un `ZoneSet` se indexa por nombre o posición, se recorre y tiene `names`; una porción como `zonas[1:]` es otro `ZoneSet`. `fill(*objetos, anchor=..., fit=..., padding=...)` coloca un objeto en cada zona, en orden. Un nombre desconocido lanza `KeyError`.

39.1.32. Zone.point / inset / split / arrange

MÉTODO

Una `Zone` expone `left`, `right`, `top`, `bottom`, `width`, `height` y `center` en unidades de escena. `point(anchor)` da uno de sus puntos; `inset(padding)` la encoge; `split(plantilla)` aplica otra plantilla dentro; `arrange(*objetos, direction="row", gap=..., align=..., justify=...)` alinea objetos libres sin adoptarlos; `direction` es `"row"`, `"column"` o `"wrap"` (filas que saltan a la siguiente cuando el objeto no cabe).

Ejemplo:

```
dots = [scene.geometry.circle(0.3) for _ in range(4)]
scene.layout.safe.inset("80px").arrange(*dots, gap="24px")
```

39.1.33. Drawable.place

MÉTODO `place(target: Zone | Drawable, *, anchor: Optional[AnchorName | Anchor] = None, fit: Optional[Fit] = None, padding: Optional[Padding] = None, offset: Optional[tuple[Length, Length]] = None) -> Self`

Coloca el objeto en una zona o sobre la caja de otro objeto: su `anchor` (centro por defecto) coincide con el mismo ancla del destino, reducido por `padding` y desplazado por `offset`. `fit` lo escala al destino. `obj.animate.place(...)` hace el mismo movimiento como animación.

Ejemplo:

```
logo = scene.geometry.circle(1).place(zone, anchor="top_right", fit="contain", padding="20px")
scene.play([logo.animate.place(zone, anchor="bottom_left").duration(0.8)])
```

39.1.34. LayoutBuilder.scatter

MÉTODO `scatter(items, region=None, *, avoid=(), gap=None, seed=0) -> None`

Coloca los objetos en posiciones aleatorias deterministas, dentro de `region` y sin solaparse entre sí ni con `avoid`. Los más grandes se colocan primero y cada uno va donde tiene más sitio, así se reparten por toda la región. Si no caben (un objeto mayor que la región, o sin hueco con ese `gap`), lanza `ValueError` y no mueve nada.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`items`	`Sequence[Drawable]`	obligatorio	Los objetos que se reparten. Conservan su tamaño; solo se mueven, sin quedar adoptados.
`region`	`"safe" | "frame" | Zone | Drawable | None`	`None`	Dónde caben: el área segura por defecto, el encuadre, una zona o la caja de un objeto.
`avoid`	`Sequence[Drawable | Zone]`	`()`	Formas que hay que dejar libres, por su caja. Para no tapar una figura sin vaciar toda su caja, evita sus partes: `avoid=[persona.part("cabeza`
`gap`	`Length | None`	`0.2`	Separación mínima

			entre objetos y respecto a cada forma de `avoid`.
`seed`	`int`	`0`	Semilla: la misma semilla da siempre el mismo reparto.

Ejemplo:

```
from gaanim import WHITE, Scene, stagger

scene = Scene(frame=(16, 9), background="#0f172a")
persona = scene.media.svg("assets/personaje.svg").scale_to(2.0)
partes = [persona.part(p) for p in ("cabeza", "torso", "piernas", "brazo-izq", "brazo-der")]
formulas = [
    scene.text.equation(f).fill(WHITE).scale_to(0.75)
    for f in ["E = m c^2", "F = m a", "pi r^2", "sqrt(2)", "e^(i pi) + 1 = 0",
              "a^2 + b^2 = c^2", "Delta x", "nabla dot E", "x!", "lambda", "oo", "phi"]
]
scene.layout.scatter(formulas, avoid=partes, gap=0.25, seed=1)
scene.play([persona.animate.fade_in().duration(0.4)])
scene.play(stagger(*[f.animate.write().duration(0.6) for f in formulas], each=0.05))
scene.render()
```

39.1.35. Componentes

`component` convierte una función `(scene, *, huecos...) -> Box` en un componente reutilizable: comprueba sus argumentos con la firma (un hueco que falta o sobra lanza `TypeError` con el nombre del componente), expone `slots` y conserva los tipos para el editor. Cada llamada crea objetos nuevos. Las plantillas de presentación `title_slide`, `lecture`, `comparison`, `vertical_short`, `minimal`, `lower_third` y `credits` son componentes y se usan también con `scene.segment(..., template=...)` y `Segment.bind` (ver Escena).

```python
from gaanim import Scene, component

scene = Scene(frame=(16, 9), theme="paper")

@component
def tag(scene, *, text: str, color: str = "#4f46e5"):
    return scene.layout.box(text, padding=("4px", "12px"), radius="full",
                            background=color, color="white")

row = scene.layout.row(tag(scene, text="Nuevo"), tag(scene, text="Beta", color="#0891b2"), gap="8px")
```

Las plantillas incluidas usan tokens de espaciado del tema. Léelos con `scene.canvas.layout_token(name)` y cámbialos con `Theme(..., layout={"page_padding": 0.7, "column_gap": 0.6})`.

39.1.36. Restricciones lineales

Relaciones entre la geometría de ramas distintas del árbol, como alinear una etiqueta externa con el centro de un gráfico. Cada objeto expone las expresiones `left`, `right`, `top`, `bottom`, `center_x`, `center_y`, `width` y `height`. Complementan a las cajas: para una cuadrícula, usa `grid`.

39.1.37. LayoutExpression / Drawable.left / right / top / bottom / center_x / center_y / width / height

PROPIEDAD `drawable.left | right | top | bottom | center_x | center_y | width | height`

Expresión lineal de geometría: admite suma y resta con otras expresiones o números y multiplicación o división por escalares finitos. ==, <= y >= crean un `LayoutConstraint`. Mezclar objetos de escenas distintas lanza `ValueError`.

39.1.38. LayoutConstraint.strong / medium / weak / named

MÉTODO `strong() · medium() · weak() · named(label) -> LayoutConstraint`

Las restricciones son obligatorias por defecto; estos métodos devuelven una copia con prioridad fuerte, media o débil, o con una etiqueta para los diagnósticos. Las débiles incumplidas aparecen en `check_layout`. Un objeto conserva su ancho y su alto frente a las restricciones medias y débiles: solo una fuerte u obligatoria lo redimensiona. Las relaciones entre posiciones mueven objetos y nunca encogen uno para resolver un conflicto.

39.1.39. LayoutBuilder.constrain

MÉTODO `constrain(*constraints: LayoutConstraint) -> ConstraintSet`

Registra restricciones y devuelve un `ConstraintSet` (su atributo `count` es el número registrado). Los conflictos entre restricciones obligatorias y las referencias entre escenas lanzan `ValueError` al instante.

Ejemplo:

```python
relations = scene.layout.constrain(
    (label.left == chart.right + 0.3).strong(),
    label.center_y == chart.center_y,
```

```
    (label.width <= page.width * 0.30).weak().named("etiqueta estrecha"),
)
```

39.1.40. LayoutBuilder.check_layout

MÉTODO `check_layout() -> list[str]`

Resuelve el layout de la escena hasta ese punto y devuelve sus diagnósticos: restricciones débiles incumplidas, fallos de composición de texto (que no detienen la recarga del editor) y cajas con fondo o borde que quedan con ancho o alto cero, como una barra vacía sin `width="fill"`. Cada aviso empieza por la ruta de la caja, por ejemplo `column[2] > row[1]`: el segundo hijo de la fila que es el tercer hijo de la columna raíz. `box.diagnostics()` filtra los de una caja y sus cajas anidadas, y `gaanim check` los muestra como avisos.

Ejemplo:

```
for message in scene.layout.check_layout():
    print(message)
```

39.1.41. Errores frecuentes

- **Mover un hijo con `move_to()`.** Expresa la intención con `align`, `justify`, `margin`, `anchor` u `offset`, o usa una zona si el objeto debe ser libre.
- **Poner `"fill"` en todos los niveles.** Decide qué caja es dueña del espacio y deja que el resto use su tamaño natural.
- **Confundir `align` con `justify`.** Identifica primero el eje principal: en una fila es horizontal y en una columna, vertical.
- **Medidas en unidades de escena para UI.** `"24px"` y los tokens del tema mantienen la proporción en cualquier formato; los números sirven para geometría con significado.
- **Construir una cuadrícula con restricciones.** Usa `grid` y reserva las restricciones para relaciones entre ramas.

CAPÍTULO

40. Visualización

scene.viz: espacios de coordenadas, funciones, cálculo, datos, valores reactivos, gráficos declarativos y campos vectoriales

`scene.viz` reúne lo que convierte números en imágenes: espacios de coordenadas tipados, funciones y construcciones de cálculo, series de datos, valores reactivos, gráficos declarativos y campos vectoriales. Todo trabaja en coordenadas de datos: al mover o escalar el espacio, curvas, puntos y etiquetas lo acompañan. Las matrices están en Matrices.

```
import math
from gaanim import Axis, BLUE, Scene

scene = Scene(frame=(16, 9), background="#0f172a")
plane = scene.viz.cartesian_2d(Axis.linear(-6, 6).ticks(1), Axis.linear(-3, 3).ticks(1))
amplitude = scene.viz.parameter(1.0)
wave = plane.plot(lambda x, a: a * math.sin(x), inputs=[amplitude]).stroke(BLUE, 0.05)
scene.play([plane.animate.create(), wave.animate.create()], duration=1.0)
scene.play([amplitude.animate.set(2.5).duration(1.0)])
scene.render()
```

Elige la herramienta según los datos:

- Un **espacio tipado** (`cartesian_2d`, `polar`, `complex`, `number_line`) cuando las coordenadas tienen significado matemático continuo: funciones, cálculo, campos y geometría propia.
- `ChartSpec` cuando cada fila de una tabla es una observación: marcas, codificaciones, escalas y transiciones con identidad estable.

40.1.1. Espacios cartesianos

Un plano con ejes, rejilla, marcas y números que conoce su escala. Todas las curvas y marcas que crea viven en sus coordenadas de datos.

40.1.2. Visualization.cartesian_2d

FÁBRICA `cartesian_2d(x: Axis, y: Axis, *, width: Optional[float] = None, height: Optional[float] = None, grid: bool = True, axes: bool = True, ticks: bool = True, numbers: bool = True, labels: bool = True, x_axis: Optional[bool] = None, y_axis: Optional[bool] = None, x_grid: Optional[bool] = None, y_grid: Optional[bool] = None, x_ticks: Optional[bool] = None, y_ticks: Optional[bool] = None, x_numbers: Optional[bool] = None, y_numbers: Optional[bool] = None, x_labels: Optional[bool] = None, y_labels: Optional[bool] = None) -> Cartesian2D`

Devuelve un `Cartesian2D` (un `CoordinateSpace`). `numbers` controla el texto de las marcas y `labels`, los títulos de `Axis.label`. Una capa desactivada sigue disponible como un `Drawable` vacío, así que el conjunto de capas es estable para componer y animar. Los números negativos usan el signo menos tipográfico (−2). Los colores salen del tema (`axes/axis`, `axes/grid`, `axes/numbers`…).

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`x / y`	`Axis`	obligatorio	Ejes con dominio, marcas y títulos (ver «Ejes»).
`width / height`	`float | None`	`None`	Tamaño en unidades de escena; por defecto, relativo al área segura.
`grid / axes / ticks / numbers / labels`	`bool`	`True`	Interruptores globales de cada capa.
`x_axis / y_axis, x_grid / y_grid, x_ticks / y_ticks, x_numbers / y_numbers, x_labels / y_labels`	`bool | None`	`None`	Sustituyen el interruptor global en

			un eje; `None` lo hereda. El global es `axes` y el de cada eje `x_axis` y `y_axis`, en singular.

Ejemplo:

```
plane = scene.viz.cartesian_2d(
  Axis.linear(-4, 4).ticks(1).label("x"),
  Axis.linear(-2, 2).ticks(1).label("y"),
  grid=False, x_grid=True,
  numbers=False, y_numbers=True,
)
scene.play([plane.animate.write().duration(1.2)])
```

40.1.3. CoordinateSpace.plot

MÉTODO `plot(function: Callable[..., float], domain: Optional[tuple[float, float]] = None, *, samples: Optional[int] = None, tolerance: float = 0.0075, derivative: Optional[Callable[..., float]] = None, inputs: Sequence[Parameter | Variable | Computed | TimeInput] = ()) -> Drawable`

Gráfica de `y = f(x)`. Con `inputs`, la curva se regenera cuando cambia el valor resuelto por la línea de tiempo, también al hacer seek. Se muestrea con más detalle si el plano tiene vistas ampliadas con `view_to`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`function`	`Callable[..., float]`	obligatorio	Recibe la coordenada x y

			después los valores de `inputs`.
`domain`	`(float, float) | None`	`None`	Intervalo de x; sin él cubre la unión de todas las ventanas de `view_to` del plano.
`samples / tolerance`	`int | None / float`	`None / 0.0075`	Muestras fijas, o muestreo adaptativo con esa tolerancia.
`derivative`	`Callable | None`	`None`	Derivada explícita con la misma firma; no hay diferenciación implícita.
`inputs`	`Sequence[Parameter | Variable | Computed | TimeInput]`	`()`	Dependencias reactivas en orden.

Ejemplo:

```
plane = scene.viz.cartesian_2d(Axis.linear(-4, 4), Axis.linear(-2, 2))
curve = plane.plot(math.sin, domain=(-math.pi, math.pi)).stroke(BLUE, 0.04)
```

40.1.4. CoordinateSpace.parametric

MÉTODO `parametric(function: Callable[..., tuple[float, float]], domain: tuple[float, float], *, samples: Optional[int] = None, tolerance: float = 0.0075, inputs: Sequence[Parameter | Variable | Computed | TimeInput] = ()) -> Drawable`

Curva paramétrica `t → (x, y)` sobre `domain`, con el mismo muestreo y las mismas `inputs` que `plot`.

Ejemplo:

```
plane = scene.viz.cartesian_2d(Axis.linear(-3, 3), Axis.linear(-2, 2))
lissajous = plane.parametric(lambda t: (2 * math.sin(3 * t), 1.5 * math.sin(2 * t)), (0,
math.tau))
```

40.1.5. CoordinateSpace.implicit

MÉTODO `implicit(function: Callable[[float, float], float], *, resolution: tuple[int, int] = (96, 64)) -> Drawable`

Curva de nivel cero de `f(x, y)`, muestreada en una rejilla de `resolution` celdas.

Ejemplo:

```
plane = scene.viz.cartesian_2d(Axis.linear(-3, 3), Axis.linear(-2, 2))
ellipse = plane.implicit(lambda x, y: x**2 / 4 + y**2 - 1)
```

40.1.6. CoordinateSpace.contour

MÉTODO `contour(function: Callable[[float, float], float], levels: Sequence[float], *, resolution: tuple[int, int] = (96, 64)) -> Drawable`

Curvas de nivel de `f(x, y)` para cada valor de `levels`.

Ejemplo:

```
plane = scene.viz.cartesian_2d(Axis.linear(-3, 3), Axis.linear(-2, 2))
rings = plane.contour(lambda x, y: x * x + y * y, levels=[0.5, 1, 2, 3])
```

40.1.7. CoordinateSpace.layer

MÉTODO `layer(name: Literal["grid", "major_grid", "minor_grid", "axis", "axes", "ticks", "numbers", "labels"]) -> Drawable`

Devuelve una capa semántica como `Drawable` para darle estilo o animarla por separado: `grid`, `major_grid`, `minor_grid`, `axis`, `axes`, `ticks`, `numbers` o `labels`.

Ejemplo:

```
plane = scene.viz.cartesian_2d(Axis.linear(-4, 4).ticks(1), Axis.linear(-2, 2).ticks(1))
plane.layer("grid").opacity(0.3)
scene.play([plane.layer("axes").animate.create(), plane.layer("numbers").animate.fade_in()])
```

40.1.8. CoordinateSpace.coord / CoordinateRef.place

MÉTODO

Coordenada simbólica `(x, y)` del espacio. Pásala a `Drawable.at_coordinate` (o a `CoordinateRef.place`) para colocar un objeto como hijo del espacio, que seguirá sus cambios de vista.

Ejemplo:

```
plane = scene.viz.cartesian_2d(Axis.linear(-4, 4), Axis.linear(-2, 2))
peak = scene.geometry.dot(0.1).fill(GOLD).at_coordinate(plane.coord(1, 1.5))
label = plane.coord(1, 1.8).place(scene.text("máximo", role="label"))
```

40.1.9. CoordinateSpace.data_to_scene

MÉTODO `data_to_scene(x: float, y: float) -> PointRef`

Devuelve un `PointRef` de escena en el dato `(x, y)` para objetos que no son hijos del espacio (anotaciones, flechas, conectores). Se resuelve en cada fotograma con la ventana de `view_to` vigente y con la posición, escala y rotación del plano. Datos no finitos o fuera de escala lanzan `ValueError`.

Ejemplo:

```
peak = plane.data_to_scene(2, 2)
note = scene.text("máximo").follow(peak, offset=(0.6, 0.4))
arrow = scene.geometry.connector(note, peak)
scene.play([plane.animate.view_to((0, 4), (0, 3)).duration(1.2)])
```

40.1.10. CoordinateSpace.data_to_local / local_to_data

MÉTODO `data_to_local(x, y) -> tuple[float, float]` · `local_to_data(x, y) -> tuple[float, float]`

Convierten entre datos y coordenadas locales del espacio con la vista vigente en el cursor.

40.1.11. CoordinateSpace.view_to

MÉTODO `view_to(x_domain: tuple[float, float], y_domain: tuple[float, float]) -> CoordinateSpace`

Cambia la ventana de datos en el cursor; `plane.animate.view_to(...)` la anima. El área de trazado no se mueve: ejes, rejillas, marcas y datos se recortan a ella, los títulos quedan fijos y los números y los marcadores de `scatter_data` conservan su tamaño. Los trazos mantienen su grosor. Los ejes con marcas automáticas regeneran marcas y rejilla con un fundido cruzado; los de paso fijo (`.ticks(1)`) lo conservan. Aplica los estilos de `layer(...)` antes del primer cambio de vista. Solo ejes lineales o temporales; otros lanzan `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`x_domain / y_domain`	`(float, float)`	obligatorio	Nueva ventana de datos; finita y creciente.

Ejemplo:

```
import math
from gaanim import Axis, GOLD, Scene
scene = Scene(frame=(16, 9), background="#0f172a")
plane = scene.viz.cartesian_2d(Axis.linear(-6, 6), Axis.linear(-3, 3))
curve = plane.plot(lambda x: math.sin(2 * x)).stroke(GOLD, 0.05)
scene.play([plane.animate.create(), curve.animate.create()], duration=0.6)
scene.play([plane.animate.view_to((-1.5, 1.5), (-1.2, 1.2)).duration(1.2)])
scene.render()
```

Las marcas de datos de un espacio cartesiano (`plot`, `parametric`, `scatter_data`, campos…) se recortan siempre al área de trazado, haya o no `view_to`. `mark.no_clip()` deja que una marca concreta sobresalga. No se recortan los objetos colocados con `at_coordinate` ni las marcas de `scene.viz.chart`.

40.1.12. CoordinateSpace.animate / drawable / move_to / scale_to / rotate_to / CoordinateSpaceAnimation.create / write / fade_in / fade_out / move_to / scale_to / rotate_to / view_to

PROPIEDAD `space.animate -> CoordinateSpaceAnimation · space.drawable() -> Drawable · space.move_to(x, y) · space.scale_to(factor) · space.rotate_to(radians) -> CoordinateSpace`

Proxy `CoordinateSpaceAnimation`: `create()` traza las capas con trazos independientes de la resolución, `write()` construye a la vez ejes, rejillas, marcas, números y títulos, y también ofrece `fade_in`, `fade_out`, `move_to`, `scale_to`, `rotate_to` y `view_to`. `move_to`, `scale_to` y `rotate_to` del propio espacio devuelven el espacio para encadenar, y `drawable()` da el objeto raíz para layout y estilo.

40.1.13. Ejes

`Axis` describe dominio, escala, marcas, números, título, cruce y estilo de un eje. Es inmutable: cada método devuelve una copia.

40.1.14. Axis.linear / log / symlog / power / time / category

MÉTODO `linear(min, max) · log(min, max, base=10) · symlog(min, max, *, base=10, threshold=1) · power(min, max, exponent) · time(min_ts, max_ts) · category(values)`

Constructores estáticos. Elige según el significado: `linear` para diferencias uniformes, `log` para órdenes de magnitud positivos, `symlog` para datos con signo alrededor de cero, `power` para potencias, `time` para marcas de tiempo en segundos y `category` para grupos discretos.

Ejemplo:

```
frequency = Axis.log(0.1, 1000, base=10).ticks(10).label("frecuencia")
signed = Axis.symlog(-100, 100, threshold=1)
groups = Axis.category(["A", "B", "C"])
```

40.1.15. Axis.ticks

MÉTODO `ticks(step: float) -> Axis`

Marcas principales cada `step` unidades de datos. Un paso fijo se conserva en cualquier vista.

40.1.16. Axis.auto_ticks

MÉTODO `auto_ticks() -> Axis`

Vuelve al paso automático, que se regenera con cada `view_to`.

40.1.17. Axis.minor_ticks

MÉTODO `minor_ticks(subdivisions: int) -> Axis`

Divide cada intervalo principal en `subdivisions` partes con marcas menores.

40.1.18. Axis.numbers

MÉTODO `numbers(format: Literal["auto", "fixed", "scientific", "percent", "fraction", "pi", "datetime"] = "auto", precision: int = 2, denominator: int = 4, pattern: Optional[str] = None) -> Axis`

Formato de los números: `auto`, `fixed`, `scientific`, `percent`, `fraction`, `pi` o `datetime`. `precision` se aplica a `fixed`, `scientific` y `percent`; `denominator`, a `fraction` y `pi`; `pattern`, a `datetime`. `fraction` y `pi` redondean al múltiplo de `1/denominator` más cercano y simplifican la fracción: con `denominator=2`, `pi` rotula π/2, π, 3π/2 y 2π, y `fraction` rotula `1/2`, `1` y `3/2`.

Ejemplo:

```
angle = Axis.linear(0, 2 * math.pi).ticks(math.pi / 2).numbers("pi", denominator=2)
share = Axis.linear(0, 1).ticks(0.25).numbers("percent", precision=0)
```

40.1.19. Axis.label

MÉTODO `label(text: str, *, position: Literal["start", "center", "middle", "mid", "end", "top", "bottom"] = "end") -> Axis`

Título del eje, separado de las marcas. `position="end"` (predeterminado) lo pone tras el extremo positivo y `"start"` tras el negativo; `"center"` (o `"middle"`, `"mid"`) centra x bajo sus números e y a la izquierda, girado 90°. En un eje vertical, `"top"` y `"bottom"` son alias de los extremos.

Ejemplo:

```
y = Axis.linear(0, 1).label("valor relativo", position="center")
```

40.1.20. Axis.crossing

MÉTODO `crossing(value: Literal["auto", "zero", "min", "max", "minimum", "maximum"] | float) -> Axis`

Dónde cruza este eje al otro: `"auto"`, `"zero"`, `"min"`, `"max"` o un valor de datos.

40.1.21. Axis.style

MÉTODO `style(*, color: Optional[ColorLike] = None, width: Optional[float] = None, tick_length: Optional[float] = None, tick_width: Optional[float] = None, tick_color: Optional[ColorLike] = None, number_color: Optional[ColorLike] = None, label_color: Optional[ColorLike] = None) -> Axis`

Sustituye solo las propiedades indicadas (`color`, `width`, `tick_length`, `tick_width`, `tick_color`, `number_color`, `label_color`); el resto se hereda del tema. Así, cambiar solo `width` conserva el color del tema.

Ejemplo:

```
x = Axis.linear(-3, 3).ticks(1).style(color=WHITE, width=0.03)
```

40.1.22. Axis.domain

PROPIEDAD `domain: tuple[float, float]`

Tupla `(mínimo, máximo)` del eje.

40.1.23. Construcciones de cálculo

Tangentes, secantes, áreas y sumas de Riemann sobre funciones del espacio.

40.1.24. CoordinateSpace.tangent

MÉTODO `tangent(function: Callable[[float], float], x: float, *, length: Optional[float] = None, dx: Optional[float] = None) -> Drawable`

Recta tangente a `function` en `x`. `length` fija la longitud visible y `dx` el paso de la derivada numérica.

Ejemplo:

```
from gaanim import Axis, BLUE, GOLD, Scene
scene = Scene(frame=(16, 9), background="#0f172a")
plane = scene.viz.cartesian_2d(Axis.linear(-1, 5).ticks(1), Axis.linear(-1, 3).ticks(1))
f = lambda x: 0.15 * x * x + 0.3
curve = plane.plot(f).stroke(BLUE, 0.04)
area = plane.riemann_sum(f, (0, 4), rectangles=8).fill(GOLD).opacity(0.6)
touch = plane.tangent(f, 3, length=3).stroke(GOLD, 0.04)
scene.play([plane.animate.create(), curve.animate.create()], duration=0.6)
scene.play([area.animate.fade_in(), touch.animate.create()], duration=0.8)
scene.render()
```

40.1.25. CoordinateSpace.normal

MÉTODO `normal(function: Callable[[float], float], x: float, *, length: Optional[float] = None, dx: Optional[float] = None) -> Drawable`

Recta normal a `function` en `x`, con las mismas opciones que `tangent`.

40.1.26. CoordinateSpace.secant

MÉTODO `secant(function: Callable[[float], float], x0: float, x1: float) -> Drawable`

Recta secante que une los puntos de `function` en `x0` y `x1`.

Ejemplo:

```
chord = plane.secant(lambda x: 0.15 * x * x, 1, 4)
```

40.1.27. CoordinateSpace.area_under

MÉTODO `area_under(function: Callable[[float], float], domain: tuple[float, float], *, samples: int = 160, baseline: float = 0.0) -> Drawable`

Región rellena entre `function` y `baseline` sobre `domain`.

Ejemplo:

```
area = plane.area_under(math.sin, (0, math.pi)).fill(BLUE).opacity(0.5)
```

40.1.28. CoordinateSpace.riemann_sum

MÉTODO `riemann_sum(function: Callable[[float], float], domain: tuple[float, float], *, rectangles: int = 12, method: Literal["left", "midpoint", "middle", "right"] = "midpoint", baseline: float = 0.0) -> Drawable`

Rectángulos de una suma de Riemann: `rectangles` de ancho igual y altura evaluada según `method` (`left`, `midpoint`, `middle` o `right`).

40.1.29. CoordinateSpace.projections

MÉTODO `projections(x: float, y: float) -> Drawable`

Líneas de proyección desde el punto `(x, y)` hasta los dos ejes.

Ejemplo:

```
guides = plane.projections(2.5, 1.8)
```

40.1.30. Series de datos

`plot_data` y `scatter_data` dibujan series `(xs, ys)` directamente en las coordenadas del espacio: si el plano se mueve o escala, la serie lo acompaña.

40.1.31. CoordinateSpace.plot_data

MÉTODO `plot_data(xs: Sequence[Optional[float]], ys: Sequence[Optional[float]], *, step: bool = False, baseline: Optional[float] = None, policy: Literal["gap", "drop", "error"] = "gap", color: Optional[ColorLike] = None, width: Optional[float] = None) -> Drawable`

Curva de datos medidos. Para recorrer muestras con el tiempo de la escena usa `Parameter.drive_from_samples`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`xs, ys`	`Sequence[float | None]`	obligatorio	Series de igual longitud; `None` es una muestra ausente.
`step`	`bool`	`False`	Gráfica escalonada.
`baseline`	`float | None`	`None`	Con un valor, rellena el área hasta esa línea base.
`policy`	`str`	`"gap"`	Muestras no finitas: `gap` corta la línea, `drop` las omite y `error` lanza un error.
`color / width`	`Color | None / float | None`	`None`	Sustituyen el trazo del tema.

Ejemplo:

```
from gaanim import CYAN, Axis, Scene

scene = Scene(frame=(16, 9), background="#0f172a")
plane = scene.viz.cartesian_2d(
  Axis.linear(0, 30).ticks(5).label("tiempo (s)"),
  Axis.linear(-0.4, 0.4).ticks(0.2).label("aceleración (g)"),
  width=12,
  height=4.75,
)
curve = plane.plot_data(times, accel, color=CYAN, width=0.05)
scene.play([plane.animate.create().duration(0.85), curve.animate.create().duration(2.2)])
```

40.1.32. CoordinateSpace.scatter_data

MÉTODO `scatter_data(xs: Sequence[Optional[float]], ys: Sequence[Optional[float]], *, radius: float = 0.06, policy: Literal["gap", "drop", "error"] = "gap", color: Optional[ColorLike] = None) -> Drawable`

Puntos de radio `radius` (unidades de escena) en las coordenadas de datos; `view_to` mueve los centros sin deformarlos. Úsalo para destacar muestras sobre una curva de `plot_data`.

Ejemplo:

```
peaks = plane.scatter_data(peak_times, peak_values, radius=0.07, color=GOLD)
scene.play([peaks.animate.fade_in()])
```

40.1.33. Valores reactivos

Magnitudes que conducen otras propiedades. Rust resuelve un snapshot numérico estable para cada instante y llama después a las funciones Python declaradas, así que el seek es exacto.

Elige el tipo según su papel:

- `Parameter`: magnitud invisible que conduce otras propiedades.
- `Variable`: magnitud que además se muestra como `etiqueta = valor`.
- `Readout`: muestra una cantidad derivada sin ser su fuente.
- `computed(...)`: escalar calculado a partir de otros.

Las funciones reciben primero las coordenadas y después los valores de `inputs=[...]`, en el orden declarado. Pueden usar `math`, funciones auxiliares y control de flujo, pero deben ser puras y devolver un escalar finito. Los setters absolutos (`move_to`, `rotate_to`, `scale_to`, `opacity` y sus variantes 3D) aceptan estas fuentes directamente.

40.1.34. Visualization.parameter

FÁBRICA `parameter(initial: float) -> Parameter`

Escalar invisible y animable. Conduce curvas reactivas, geometría (`point_on_curve`, `always_redraw_arc`...) y setters; cada objeto conducido necesita su propia animación de entrada. Valores no finitos lanzan `ValueError`.

Ejemplo:

```
import math
from gaanim import Axis, Scene

scene = Scene(frame=(16, 9), background="#0f172a")
amplitude = scene.viz.parameter(1.0)
axes = scene.viz.cartesian_2d(Axis.linear(-4, 4), Axis.linear(-2, 2))
curve = axes.plot(lambda x, a: a * math.sin(x), inputs=[amplitude])
scene.play([axes.animate.create(), curve.animate.write(),
amplitude.animate.set(2.0).duration(1.2)])
```

40.1.35. Parameter.set

MÉTODO `set(value: float) -> None`

Fija el valor al instante: el valor inicial al declarar o un corte reversible en el cursor después.

40.1.36. Parameter.animate

PROPIEDAD `animate: Anim`

Proxy puro: `parameter.animate.set(value)` devuelve un `Anim` con duración y easing configurables para `scene.play`.

40.1.37. Parameter.current

PROPIEDAD `current: float`

Valor en el cursor de autoría (el espejo de lo último que fijaste o animaste), no el valor en un instante reproducido.

40.1.38. Parameter.add_updater_fn

MÉTODO `add_updater_fn(callback: Callable[[float, float, float], float], *, reset: Callable[[], None] | None = None, fixed_dt: float | None = None) -> Parameter`

Conduce el escalar con `callback(actual, dt, transcurrido) -> valor`. Sin `reset` ni `fixed_dt`, calcula el valor a partir de `transcurrido`, como en el ejemplo: un valor que acumula `dt` vuelve al inicial al hacer seek o capturar fotogramas sueltos. Para simulaciones con estado, pasa `reset` y un `fixed_dt` positivo: Gaanim llama a `reset()` y repite los pasos fijos al hacer seek y al exportar. Las simulaciones de objetos de paso fijo se ejecutan antes, así que el callback ve su estado del mismo fotograma. Resultados no finitos lanzan `ValueError`.

Ejemplo:

```
phase = scene.viz.parameter(0.0)
phase.add_updater_fn(lambda value, dt, elapsed: 2.0 * elapsed)
hand = scene.geometry.line((0, 0), scene.geometry.polar_point((0, 0), 1.5, phase))
scene.wait(1)
```

40.1.39. Parameter.drive_from_samples

MÉTODO `drive_from_samples(times: Sequence[float], values: Sequence[float], *, interpolation: Literal["linear", "step"] = "linear", scale: float = 1.0, offset: float = 0.0) -> Parameter`

Recorre una serie `(times, values)` de forma nativa: el valor es `offset + scale * muestra` como función pura del tiempo, sin callbacks. `times` es relativo al cursor donde haces la llamada; `interpolation` es `"linear"` o `"step"`.

Ejemplo:

```
level = scene.viz.parameter(0.0)
level.drive_from_samples([0, 0.5, 1.0, 1.5], [0, 1, 0.4, 0.8], scale=2.0)
readout = scene.viz.readout(level, label="$h$", unit="m")
scene.wait(1.5)
```

40.1.40. Parameter.remove_updater

MÉTODO `remove_updater() -> None`

Quita el callback o la serie que conduce el parámetro.

40.1.41. computed

FUNCIÓN `computed(callback, *, inputs=()) -> Computed`

Escalar determinista calculado a partir de `inputs`, que pueden incluir otros `Computed`. Las dependencias compartidas reutilizan su caché para el mismo snapshot. Sigue siendo escalar: no devuelve colores, texto ni geometría. Envolver un parámetro de otra escena no permite usarlo aquí.

Ejemplo:

```
radius = scene.viz.parameter(1.0)
area = computed(lambda r: math.pi * r * r, inputs=[radius])
doubled = computed(lambda value: 2 * value, inputs=[area])
scene.viz.readout(doubled, label="2A")
```

40.1.42. Visualization.time

PROPIEDAD `time: TimeInput`

Tiempo de la línea de tiempo como entrada reactiva explícita; es la misma fuente que `scene.time`.

Ejemplo:

```
bob = scene.geometry.dot(0.15).fill(GOLD)
bob.move_to(0, computed(lambda t: math.sin(3 * t), inputs=[scene.viz.time]))
scene.wait(2)
```

40.1.43. Visualization.variable

FÁBRICA `variable(initial: float, *, label: str, format: str = ".2f", prefix: str = "", suffix: str = "", unit: Optional[str] = None, font_size: Optional[float] = None, color: Optional[ColorLike] = None, invalid: str = "invalid", decimal_separator: str = ".", spacing: Optional[float] = None, reserve: Optional[float] = None, font: Optional[str] = None, weight: Optional[int] = None) -> Variable`

Escalar animable que también se dibuja como `etiqueta = valor unidad`. Acepta las mismas operaciones que `Parameter`. Sus partes `label`, `equals`, `number` y `unit` son `Drawable` que se estilizan por separado; todos los términos usan `font_size` (0.48 por defecto) y comparten línea base. `color` pinta todo, también el número al actualizarse.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`initial`	`float`	obligatorio	Valor inicial.
`label`	`str`	obligatorio	Etiqueta antes del signo igual;

			admite matemática `$…$`.
`format`	`str`	`".2f"`	Especificación de formato de Python para un `float`, como en `format(valor, spec)`: relleno y alineación, signo, `z`, `#`, `0`, ancho, agrupación `,` o `_`, precisión y tipo `f`, `F`, `e`, `E`, `g`, `G` o `%`. `"05.2f"` muestra `00.07`. Los tipos de enteros o texto (`d`, `x`, `s`…) y las especificaciones mal formadas lanzan `ValueError`; para enteros usa `".0f"`. El relleno con espacios no se ve, porque el número se alinea a la derecha por su tinta: para reservar sitio usa `reserve`.
`prefix / suffix / unit`	`str / str / str | None`	`"" / "" / None`	Afijos y unidad visibles.
`decimal_separator`	`str`	`"."`	`","` muestra `3,14` y convierte la agrupación `,` en `.`.
`font / weight`	`str | None / int | None`	`None / None`	Familia y peso (1 a 1000) del número y de la etiqueta y la unidad en texto plano; una etiqueta `$…$`

			conserva la fuente matemática. Sin ellos, el número usa la fuente matemática del tema. Un peso fuera de rango lanza `ValueError`.

Ejemplo:

```
from gaanim import RED, Scene

scene = Scene(frame=(16, 9), background="#0f172a")
k = scene.viz.variable(10, label="$k$", format=".0f", color=RED)
scene.play([k.animate.create(), k.animate.set(100).duration(1.5)])
```

40.1.44. Visualization.readout

FÁBRICA `readout(source: _ReactiveScalar | Callable[..., float], *, inputs: Sequence[Parameter | Variable | Computed | TimeInput] = (), label: Optional[str] = None, format: str = ".2f", prefix: str = "", suffix: str = "", unit: Optional[str] = None, font_size: Optional[float] = None, color: Optional[ColorLike] = None, invalid: str = "invalid", decimal_separator: str = ".", spacing: Optional[float] = None, reserve: Optional[float] = None, font: Optional[str] = None, weight: Optional[int] = None) -> Readout`

Lectura numérica reactiva con términos equiespaciados y alineados en la línea base. El número solo se regenera si cambia el texto formateado. Sin `color` usa el color de texto del tema. Dentro de un layout, su celda mide lo que el texto más ancho que mostrará: el de los valores a los que se fijan o animan sus parámetros (el `percent` de una encuesta llega a 100) y el de `reserve`. El número queda alineado a la derecha en ese espacio, así que sus vecinos no se mueven mientras cuenta. `variable` también acepta `reserve`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`source`	`número | Parameter | Variable | Computed | callable`	obligatorio	Escalar o función pura cuyos argumentos son `inputs`.

`inputs`	`Sequence`	`()`	Dependencias en orden.
`invalid`	`str`	`"invalid"`	Texto para evaluaciones inválidas o no finitas.
`label / format / prefix / suffix / unit / decimal_separator / font / weight`	—	—	Igual que en `variable`; un separador que sea dígito, signo, espacio, `e` o `%` lanza `ValueError`.
`spacing`	`float | None`	`None`	Espacio entre etiqueta, `=`, número y unidad, en unidades de escena. Por defecto, un espacio fino que crece con `font_size`

			(0.1 con el tamaño por defecto). `prefix` y `suffix` van pegados al número: `suffix="m"` conserva su espacio.
`reserve`	`float | None`	`None`	Valor cuyo texto cabe en la celda de un layout, además de los valores que toman sus parámetros. Úsalo con conteos en vivo sin máximo conocido, como `votes` (`reserve=999`).

		Un valor no finito lanza `ValueError`.

Ejemplo:

```
import math
from gaanim import Scene

scene = Scene(frame=(16, 9), background="#0f172a")
radius = scene.viz.parameter(1.0)
area = scene.viz.readout(lambda r: math.pi * r**2, inputs=[radius], label="$A$", format=".2f",
unit="m²")
scene.play([area.animate.create(), radius.animate.set(3.0).duration(1.5)])
```

40.1.45. Readout.label / equals / number / unit / Variable.label / equals / number / unit / set / current / animate / add_updater_fn / remove_updater

PROPIEDAD `label · equals · number · unit -> Drawable | None`

Partes de `Readout` y `Variable` que se estilizan por separado (`number` siempre existe; las demás valen `None` si no se pidieron). `Variable` añade `current`, `set`, `animate`, `add_updater_fn` y `remove_updater`, con el mismo comportamiento que en `Parameter`.

Ejemplo:

```
speed = scene.viz.readout(3.2, label="$v$", unit="m/s")
speed.number.fill(GOLD)
```

40.1.46. Readout.move_to / Variable.move_to

MÉTODO `move_to(x, y=None, anchor: Anchor | TextAnchor | None = None) -> Readout`

Coloca la lectura como cualquier objeto: sin `anchor`, su centro va a `(x, y)`; con un `Anchor`, ese punto de su caja. Con un `TextAnchor` la coloca por su línea base, alineada con un `Text`: `TextAnchor.BASELINE_LEFT` deja fijo su comienzo aunque el número cambie de ancho. Con un `TextAnchor` hacen falta `x` e `y`, que pueden ser escalares reactivos.

Ejemplo:

```
scene.text("Carga").move_to(-4, 0, TextAnchor.BASELINE_LEFT)
scene.viz.readout(12.5, label="$P$", unit="kN").move_to(-2, 0, TextAnchor.BASELINE_LEFT)
```

40.1.47. Contador rodante

Cifras que giran como un cuentakilómetros, para dinero, marcadores y cuentas regresivas.

40.1.48. Visualization.rolling_number

FÁBRICA `rolling_number(value: float = 0.0, *, decimals: int = 0, min_digits: int = 1, pad: Literal["0", " "] = "0", group_separator: str = "", decimal_separator: str = ".", prefix: str = "", suffix: str = "", show_plus: bool = False, font_family: Optional[str] = None, weight: Optional[int] = None, font_size: float = 0.75, digit_spacing: float = 0.02, line_height: float = 1.25, mode: str = "odometer", direction: str = "up", color: Optional[ColorLike] = None) -> RollingNumber`

Las cifras son contornos vectoriales de ancho fijo recortados en su celda. La geometría depende solo del valor actual, así que reproducción, seeks y exportación coinciden. El borde derecho queda anclado; `min_digits` reserva posiciones para que no crezca durante un acarreo. Con `pad=" "` esas posiciones quedan vacías: el contador ocupa siempre lo mismo, el prefijo no se desplaza y una frase con el valor final queda centrada para todos los valores. En modo `continuous`, las ruedas se asientan durante el primer y el último 15 % de cada animación del valor. Opciones inválidas lanzan `ValueError`; si un driver sale del rango, se muestra una raya.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`value`	`float`	`0.0`	Valor inicial finito; su magnitud por `10**decimals` debe ser menor que `1e15`.
`decimals / min_digits`	`int`	`0 / 1`	De 0 a 6 decimales y de 1 a 15 posiciones enteras reservadas; la suma

			no supera 15.
`pad`	`str`	`"0"`	Relleno de las posiciones reservadas sin usar: `"0"` dibuja ceros iniciales y `" "` deja celdas vacías del mismo ancho, sin dibujar sus separadores de miles.
`group_separator / decimal_separator`	`str`	`"" / "."`	Cero o un carácter, y un carácter distinto.
`prefix / suffix / show_plus`	`str / str / bool`	`"" / "" / False`	Afijos de una línea (256 bytes UTF-8 en total)

			y signo positivo visible.
`font_family / weight`	`str | None / int | None`	`None`	Familia y peso (1 a 1000), resueltos como en `scene.text`; sin familia hereda la de cuerpo.
`font_size / digit_spacing / line_height`	`float`	`0.75 / 0.02 / 1.25`	Tamaño, espacio entre celdas y altura de la ventana como múltiplo de la altura de las cifras (al menos 1).
`mode`	`str`	`"odometer"`	`odometer` arrastra las ruedas superiores durante

			la última unidad antes del acarreo; `continuous` gira cada rueda a la velocidad de su posición.
`direction`	`str`	`"up"`	`up` o `down` para magnitudes crecientes; al disminuir se invierte.
`color`	`Color | None`	`None`	Color de cifras, signo y afijos; sin él, el del tema.

Ejemplo:

```
from gaanim import Easing, Scene

scene = Scene(frame=(16, 9), background="#0f172a")
counter = scene.viz.rolling_number(
    98, min_digits=4, prefix="$ ", group_separator=",",
).move_to(0, 0)
scene.play([counter.count_to(1250, duration=3).easing(Easing.SMOOTH)])
scene.play([counter.animate.set(500).duration(2)])
designs = scene.viz.rolling_number(0, min_digits=8, pad=" ", group_separator=" ").move_to(0, -1.5)
```

```
scene.play([designs.count_to(14_000_605, duration=3)])
counter.set(0)  # Corte reversible en el cursor actual.
scene.wait(0.5)
scene.render()
```

40.1.49. RollingNumber.count_to

MÉTODO `count_to(value: float, *, duration: float = 1.0, snap: bool = False) -> Anim`

Animación de cuenta para `scene.play`, compatible con easing y retardo. El destino se usa tal cual: una fracción de la unidad mínima deja la rueda entre dos cifras. `snap=True` redondea al valor representable más cercano, así que termina asentado y `current` coincide con lo visible.

Ejemplo:

```
mean = scene.viz.rolling_number(0, decimals=1)
scene.play([mean.count_to(61.7956, snap=True)])  # Termina en 61.8.
```

40.1.50. RollingNumber.set

MÉTODO `set(value: float, *, snap: bool = False) -> RollingNumber`

Fija el valor al instante (un corte reversible tras la declaración) y devuelve el contador. Acepta `snap` como `count_to`.

40.1.51. RollingNumber.move_to

MÉTODO `move_to(x: Any, y: Any = None, anchor: AnchorName | Anchor | TextAnchor | None = None) -> RollingNumber`

Coloca el contador. Con `TextAnchor.BASELINE_LEFT`, `BASELINE_CENTER` o `BASELINE_RIGHT`, la línea base de las cifras queda en `y`, para alinearlo con texto; un `Anchor` usa la ventana completa, con el margen de `line_height`.

Ejemplo:

```
number = scene.viz.rolling_number(2, min_digits=2, font_size=1)
number.move_to(-5, 1.12, TextAnchor.BASELINE_LEFT)
scene.text("Objetivos", size=1).move_to(-3.5, 1.12, TextAnchor.BASELINE_LEFT)
```

40.1.52. RollingNumber.visual / parameter / current / animate / fill / opacity

PROPIEDAD `visual: Drawable · parameter: Parameter · current: float · animate`

`visual` es el objeto dibujado: usa `counter.visual.animate` para posición, opacidad o entradas (sin entrada, el contador es visible desde su declaración). `parameter` es el escalar subyacente, reutilizable en `computed`, lecturas o `drive_from_samples`. `animate.set(value)` anima el valor

como en un `Parameter`. `move_to`, `fill`, `opacity` y `set` devuelven el contador; otros métodos heredados pueden devolver su `Drawable`.

40.1.53. Anillos de progreso y temporizadores

Un arco que se llena en el sentido del reloj desde las 12, con una pista tenue detrás y un contador rodante en el centro. Todo sigue a un único `Parameter`.

40.1.54. Visualization.progress_ring

FÁBRICA `progress_ring(value: float = 0.0, *, radius: float = 1.0, width: float = 0.12, color: Optional[ColorLike] = None, track: bool = True, track_color: Optional[ColorLike] = None, label: bool = True, decimals: int = 0, label_color: Optional[ColorLike] = None, font_size: float = 0.5) -> ProgressRing`

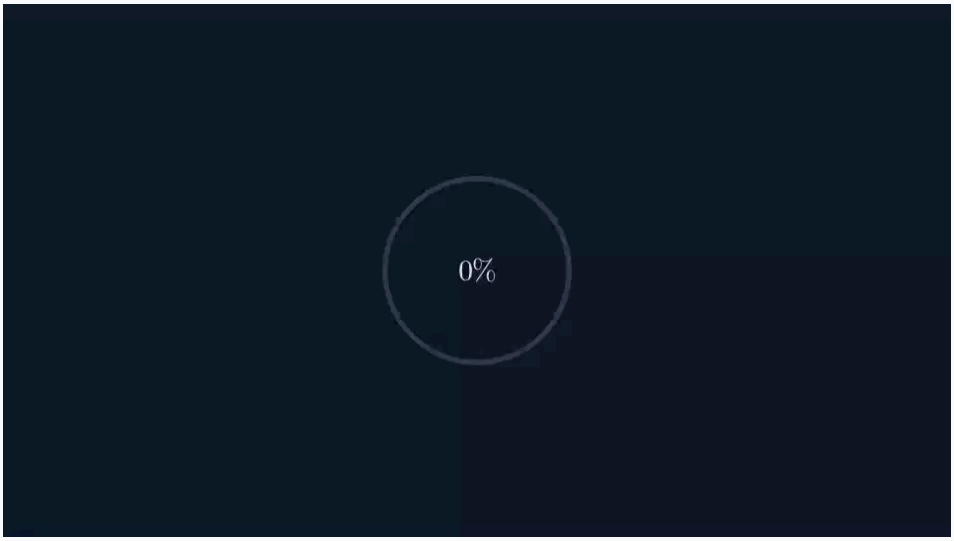
El arco tiene extremos redondeados y su caja es el anillo completo, así que la maquetación no sigue al barrido. Anima el valor con `animate.set(...)`; la geometría depende solo del valor actual, así que reproducción, seeks y exportación coinciden. `arc`, `track` y `label` dan acceso a cada parte. Tamaños no positivos o un valor no finito lanzan `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`value`	`float`	`0.0`	Progreso inicial; el anillo muestra `value` recortado a `[0, 1]`.
`radius / width`	`float`	`1.0 / 0.12`	Radio del anillo y grosor del trazo, en unidades de escena.
`color`	`Color | None`	`None`	Color del arco; sin él, el acento del tema.
`track / track_color`	`bool / Color | None`	`True / None`	Dibuja un círculo tenue completo detrás del arco, con otro color si se indica.
`label / decimals`	`bool / int`	`True / 0`	Muestra el porcentaje con un contador rodante de 0 a 6 decimales.
`label_color / font_size`	`Color | None / float`	`None / 0.5`	Color y tamaño del porcentaje; sin color, el del texto.

Ejemplo:

```python
from gaanim import Scene
scene = Scene(frame=(16, 9), background="#0f172a")
ring = scene.viz.progress_ring(0.0, radius=1.6, width=0.18)
scene.play([ring.animate.set(0.75).duration(1.5)])
scene.render()
```



40.1.55. Visualization.countdown

FÁBRICA `countdown(seconds: float, *, radius: float = 1.0, width: float = 0.12, color: Optional[ColorLike] = None, track: bool = True, track_color: Optional[ColorLike] = None, label: bool = True, label_color: Optional[ColorLike] = None, font_size: float = 0.6) -> ProgressRing`

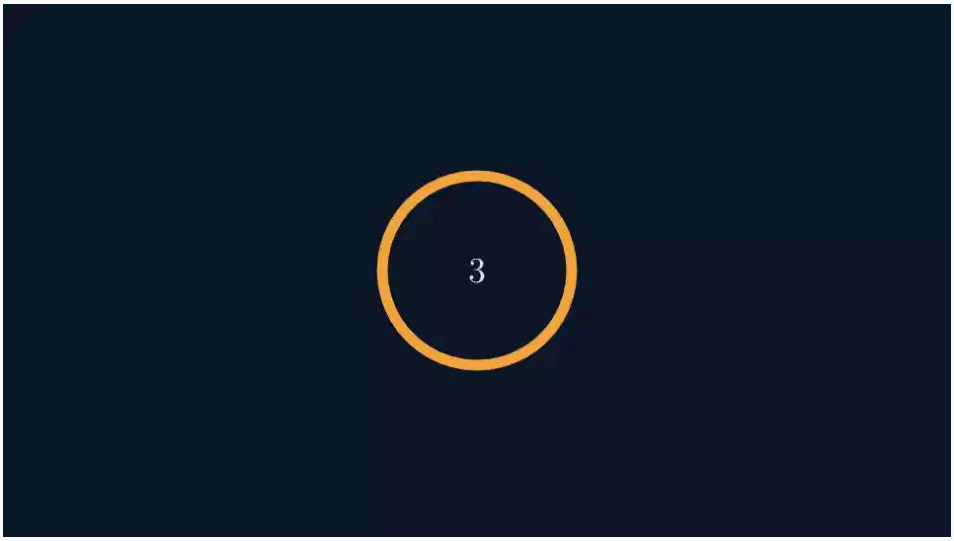
Un `ProgressRing` que empieza lleno, con los segundos restantes rodando en el centro. `count_down()` lo vacía a velocidad constante. El resto de opciones son las de `progress_ring`. Unos `seconds` no positivos lanzan `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`seconds`	`float`	obligatorio	Duración de la cuenta; también es el máximo del anillo.

Ejemplo:

```python
from gaanim import Scene
scene = Scene(frame=(16, 9), background="#0f172a")
timer = scene.viz.countdown(3, radius=1.6, width=0.18)
scene.play([timer.count_down()])
scene.render()
```



40.1.56. ProgressRing.count_down

MÉTODO `count_down(duration: Optional[float] = None) -> Anim`

Con `countdown(10)` dura diez segundos reales. Una duración negativa lanza `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`duration`	`float | None`	`None`	Segundos de la animación; por defecto, el valor actual.

Devuelve: `Anim` — El valor bajando hasta cero con easing lineal.

40.1.57. ProgressRing.parameter / visual / arc / track / label / maximum / current / set / animate

PROPIEDAD `parameter: Parameter · visual: Drawable · arc: Drawable · track: Drawable | None · label: Drawable | None · maximum: float · current: float · set(value) · animate`

`parameter` es el escalar que mueve el arco y la etiqueta, reutilizable en `computed` o lecturas. `maximum` es `1.0` en un anillo y los segundos en una cuenta atrás. `set` fija el valor al instante (un corte reversible tras la declaración) y `animate.set(value)` lo anima. `visual` es el grupo de las tres partes: usa `visual.animate` para moverlo o desvanecerlo. `move_to`, `shift_by` y `opacity` devuelven el anillo. La etiqueta se asienta en cifras enteras: redondea el porcentaje (la cuenta atrás redondea los segundos hacia arriba) y gira a la cifra siguiente cuando el valor la cruza, así que un valor como `1/3` muestra `33%` sin quedarse entre dos cifras.

40.1.58. Carrera de barras

Una *bar chart race*: barras ordenadas por valor que crecen, se adelantan unas a otras y salen del ranking. Todo es función pura de una única posición de fotograma clave, así que un seek muestra lo mismo que la reproducción continua.

40.1.59. Visualization.bar_race

FÁBRICA `bar_race(frames: Mapping[Any, Mapping[str, float]] | Sequence[tuple[Any, Mapping[str, float]]], *, top: int = 10, rank_smoothing: float = 0.3, value_format: str = "{:,.0f}", width: float = 10.0, height: float = 6.0, label_width: Optional[float] = None, bar_gap: float = 0.18, colors: Mapping[str, ColorLike] | Sequence[ColorLike] | None = None, label_color: Optional[ColorLike] = None, font_size: Optional[float] = None, ticker: bool = True, ticker_size: Optional[float] = None, ticker_color: Optional[ColorLike] = None) -> BarRace`

Los valores se interpolan linealmente entre fotogramas clave. Cuando dos barras se cruzan, intercambian sus puestos con un movimiento suavizado en vez de saltar; entre cruces, cada barra ocupa su propio puesto. Las barras que salen del `top` se desvanecen por debajo del último puesto y la barra más larga ocupa siempre todo el ancho disponible. Los nombres van en una columna a la izquierda y un contador rodante (`rolling_number`) sigue el extremo de cada barra. Si todas las etiquetas son números enteros, como años, el ticker rueda entre ellos; si no, cambia de texto con un fundido. Datos vacíos o no finitos, nombres repetidos, un `value_format` no admitido o tamaños no positivos lanzan `ValueError`; otra forma de `frames`, `TypeError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`frames`	`Mapping[Any, Mapping[str, float]] | Sequence[tuple[Any, Mapping[str, float]]]`	obligatorio	Cada etiqueta de fotograma clave (un año, una fecha…) con los `{nombre: valor}` de ese instante, en orden. Un nombre ausente vale cero en ese fotograma.

`top`	`int`	`10`	Puestos visibles; el líder arriba.
`rank_smoothing`	`float`	`0.3`	Fotogramas clave (de 0 a 2) que dura el intercambio de dos barras, centrado en el instante en que se cruzan y acortado para que termine antes de que esas dos barras vuelvan a cruzarse; con `0` cambian de puesto

			al instante.
`value_format`	`str`	`"{:,.0f}"`	Campo de formato de Python con texto opcional alrededor: `"{}"`, `"{:,.0f}"`, `"{:_.2f}"` o `"${:,.1f}M"`, hasta 6 decimales.
`width / height`	`float`	`10.0 / 6.0`	Tamaño de toda la carrera, centrada en su origen, en unidades de escena.
`label_width / bar_gap`	`float | None / float`	`None / 0.18`	Ancho de la columna de nombres (por defecto el 22% de `width`) y

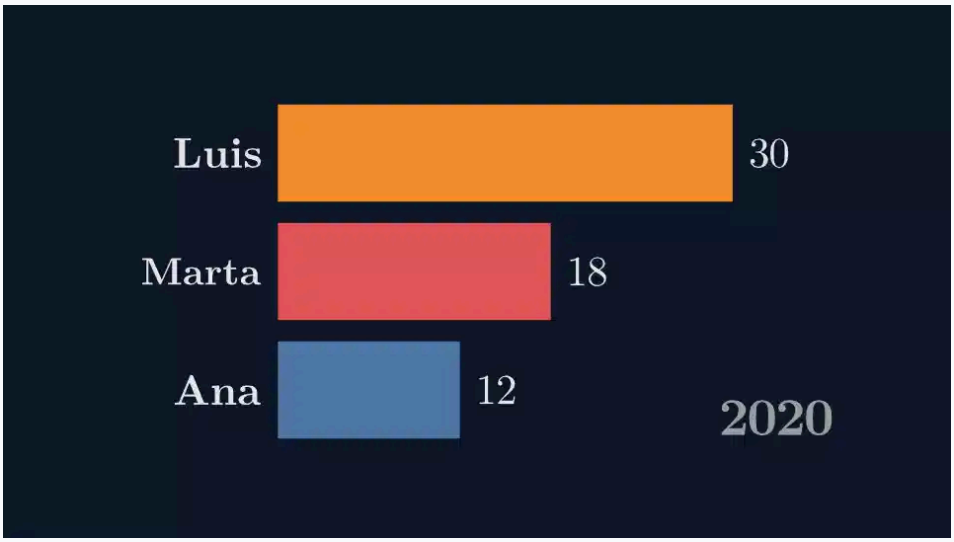
			fracción vacía de cada puesto. Un nombre más ancho que la columna se dibuja más pequeño, hasta que cabe.
`colors`	`Mapping[str, Color] | Sequence[Color] | None`	`None`	Colores por nombre o una lista que vuelve a empezar como una paleta; el resto toma la paleta del tema (`Theme(palette=`
`label_color / font_size`	`Color | None / float | None`	`None / None`	Color y tamaño de nombres y

			valores; por defecto el color de texto del tema y el 55% del grosor de la barra, sin pasar del 12% de `height` para que las carreras de pocos puestos sigan legibles.
`ticker / ticker_size / ticker_color`	`bool / float | None / Color | None`	`True / None / None`	Etiqueta del fotograma clave abajo a la derecha, por defecto al 15% de `height`

y con el color atenuado del tema.

Ejemplo:

```
from gaanim import Scene
scene = Scene(frame=(16, 9), background="#0f172a")
frames = {
    2020: {"Ana": 12, "Luis": 30, "Marta": 18, "Noé": 5},
    2021: {"Ana": 35, "Luis": 34, "Marta": 22, "Noé": 20},
    2022: {"Ana": 48, "Luis": 40, "Marta": 51, "Noé": 38},
    2023: {"Ana": 60, "Luis": 44, "Marta": 58, "Noé": 71},
}
race = scene.viz.bar_race(frames, top=3, width=12, height=6)
scene.play([race.animate.play().duration(4)])
scene.render()
```



40.1.60. BarRaceAnimation.play / to

MÉTODO `play() -> Anim · to(position: float) -> Anim`

`race.animate.play()` avanza desde la posición actual hasta el último fotograma clave a ritmo constante, un segundo por fotograma pendiente salvo que encadenes `.duration(segundos)`. `to(position)` va a una posición concreta: `0` es el primer fotograma clave y las posiciones fraccionarias quedan entre dos. Una posición fuera de `[0, frame_count - 1]` lanza `ValueError`.

40.1.61. BarRace.parameter / position / frame_count / names / ticker / bar / set / visual

PROPIEDAD `parameter: Parameter · position: float · frame_count: int · names: list[str] · ticker: Drawable | None · bar(name) -> Drawable · set(position) · visual: Drawable`

`parameter` es la posición de fotograma clave, reutilizable en `computed` o lecturas. `bar(name)` devuelve la fila de una barra (barra, nombre y valor), que se mueve entre puestos y se desvanece al salir del top; su propia opacidad queda libre, así que `race.bar(name).animate.opacity(0.3)` la atenúa además de ese desvanecimiento. Un nombre desconocido lanza `KeyError`. `set` salta a una posición (un corte reversible tras la declaración). `visual` es el grupo de todas las filas y el ticker; `move_to`, `shift_by` y `opacity` devuelven la carrera.

40.1.62. Recta numérica, plano polar y plano complejo

Espacios tipados de una dimensión, polares y complejos.

40.1.63. Visualization.number_line

FÁBRICA `number_line(axis: Axis, *, length: Optional[float] = None, axis_visible: bool = True, ticks: bool = True, numbers: bool = True, labels: bool = True) -> NumberLine`

Recta numérica tipada. `axis_visible`, `ticks`, `numbers` y `labels` controlan cada componente; los componentes ocultos siguen disponibles como capas vacías y la conversión de datos sigue activa.

Ejemplo:

```
line = scene.viz.number_line(Axis.linear(0, 10).ticks(1), length=10)
```

40.1.64. NumberLine.point_ref

MÉTODO `point_ref(value: _ReactiveScalar, *, normal_offset: Optional[_ReactiveScalar] = None) -> PointRef`

`PointRef` reactivo que sigue a la recta cuando se mueve, rota o escala. Las escalas categóricas rechazan valores reactivos con `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`value`	`float | Parameter | Variable | Computed`	obligatorio	Valor convertido con la escala de la recta.
`normal_offset`	`float | fuente reactiva | None`	`None`	Desplazamiento perpendicular

			en unidades locales.

40.1.65. NumberLine.function

MÉTODO `function(function: Callable[..., float], domain: Optional[tuple[float, float]] = None, *, normal_scale: float = 1.2, reveal: Optional[_ReactiveScalar] = None, samples: Optional[int] = None, tolerance: float = 0.0075, inputs: Sequence[Parameter | Variable | Computed | TimeInput] = ()) -> Drawable`

Grafica una función perpendicular a la recta, con muestreo y actualización en Rust. Un `reveal` reactivo puede compartir el mismo escalar que un punto móvil sin desfases de longitud de arco.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`function`	`Callable[..., float]`	obligatorio	Recibe la coordenada y después `inputs`.
`normal_scale`	`float`	`1.2`	Distancia local para una salida igual a 1.
`reveal`	`float | fuente reactiva | None`	`None`	Extremo visible exacto, en coordenadas de datos.

Ejemplo:

```
import math
from gaanim import Axis, Scene, computed

scene = Scene(frame=(16, 9), background="#0f172a")
theta = scene.viz.parameter(0.0)
line = scene.viz.number_line(
  Axis.linear(0, 3 * math.pi).ticks(math.pi).numbers("pi", denominator=1),
  length=9.5,
)
curve = line.function(lambda t: math.sin(t), normal_scale=1.2, reveal=theta)
point = scene.geometry.dot(0.1).follow(
  line.point_ref(theta, normal_offset=computed(lambda t: 1.2 * math.sin(t), inputs=[theta]))
)
scene.play([line.animate.create(), curve.animate.fade_in().duration(0.01),
point.animate.fade_in()])
scene.play([theta.animate.set(3 * math.pi).duration(4)])
```

40.1.66. NumberLine.coord / data_to_local / layer / drawable / animate

MÉTODO `coord(value) -> CoordinateRef · data_to_local(value) -> float · layer(name) -> Drawable · drawable() -> Drawable · animate`

Coordenada simbólica, conversión a coordenada local y capas `axis`, `ticks`, `numbers` y `labels`.

40.1.67. Visualization.polar

FÁBRICA `polar(radial: Axis, *, radius: float = 2.2, angle_divisions: int = 12, grid: bool = True, axes: bool = True, numbers: bool = True, labels: bool = True, rings: Optional[bool] = None, spokes: Optional[bool] = None) -> PolarSpace`

Espacio polar de radio `radius` con `angle_divisions` radios. `rings` y `spokes` heredan `grid` si se omiten; `numbers` afecta a los valores radiales y `labels`, al título radial.

Ejemplo:

```
polar = scene.viz.polar(Axis.linear(0, 2).ticks(0.5).label("r"), radius=3)
rose = polar.plot(lambda angle: 2 * abs(math.cos(3 * angle))).stroke(GOLD, 0.04)
```

40.1.68. PolarSpace.plot

MÉTODO `plot(function: Callable[[float], float], domain: tuple[float, float] = (0.0, 6.283185307179586), *, samples: int = 360) -> Drawable`

Curva polar `r = f(θ)` sobre `domain` (una vuelta completa por defecto).

40.1.69. PolarSpace.coord / layer / drawable / animate

MÉTODO `coord(radius, angle) -> CoordinateRef · layer(name) -> Drawable · drawable() -> Drawable · animate`

Coordenada simbólica polar y capas `grid`, `axes`, `numbers` y `labels`.

40.1.70. Visualization.complex

FÁBRICA `complex(x: Optional[Axis] = None, y: Optional[Axis] = None, *, width: Optional[float] = None, height: Optional[float] = None, grid: bool = True, axes: bool = True, ticks: bool = True, numbers: bool = True, labels: bool = True, x_axis: Optional[bool] = None, y_axis: Optional[bool] = None, x_grid: Optional[bool] = None, y_grid: Optional[bool] = None, x_ticks: Optional[bool] = None, y_ticks: Optional[bool] = None, x_numbers: Optional[bool] = None, y_numbers: Optional[bool] = None, x_labels: Optional[bool] = None, y_labels: Optional[bool] = None) -> ComplexSpace`

Plano complejo cartesiano. Sin ejes, crea `Re` e `Im`; los interruptores y su precedencia son los de `cartesian_2d`, y devuelve el mismo tipo de espacio.

Ejemplo:

```
plane = scene.viz.complex(labels=False)
z = scene.geometry.dot(0.1).fill(GOLD).at_coordinate(plane.coord(1, 1))
```

40.1.71. Gráficos declarativos

`ChartSpec` es una receta inmutable: datos, marca, codificaciones, ejes y guías. `scene.viz.chart(spec)` la materializa en capas y `chart.animate.to(otra)` anima el paso de una receta a otra.

```
from gaanim import Axis, ChartSpec, Field, Guide, Scale, Scene, Value

scene = Scene(frame=(16, 9), background="#0f172a")
spec = (
  ChartSpec({
    "id": ["a", "b", "c", "d"],
    "x": [-2, -0.5, 1, 2.5],
    "y": [1, 3, 2, 3.5],
    "group": ["A", "B", "A", "B"],
  }, key="id")
  .mark("point")
  .encode(x="x", y="y", color=Field("group", scale=Scale.category()), size=Value(0.12))
  .axes(x=Axis.linear(-3, 3).ticks(1).label("x"), y=Axis.linear(0, 4).ticks(1).label("y"))
  .guides(color=Guide.legend(title="Grupo"))
)
chart = scene.viz.chart(spec)
scene.play([chart.layer("axes").animate.create().duration(0.8)])
scene.play([chart.layer("marks").animate.fade_in(), chart.layer("guides").animate.fade_in()],
duration=0.6)
scene.render()
```

40.1.72. ChartSpec

CLASE `ChartSpec(data, *, key=None)`

Receta de gráfico. Una mutación externa posterior de los datos no cambia el resultado: los seeks anteriores siguen siendo iguales. Cada método devuelve una receta nueva, así que puedes derivar varios estados de una base.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`data`	`mapping | dataframe | DataTable | DataSource`	obligatorio	Se copia al

			instante como instantánea inmutable propia.
key	`str | None`	`None`	Columna de identidad estable para las transiciones; nulos o duplicados lanzan un error.

Ejemplo:

```
base = ChartSpec({"category": ["a", "b", "c"], "time": [0, 1, 2], "value": [3, 1, 2]})
bars = base.mark("bar").encode(x="category", y="value")
points = base.mark("point").encode(x="time", y="value")
```

40.1.73. ChartSpec.mark

MÉTODO `mark(kind: ChartMark, **options: float | str | Color) -> ChartSpec`

Tipo de marca: `point`, `line`, `step`, `area`, `bar`, `histogram`, `box`, `violin`, `error_bar`, `heatmap`, `surface` o `arc`. Las opciones dependen de la marca: las barras aceptan `width` (fracción de la banda en ejes categóricos), `label_position` (`"outside"` o `"inside"`), `label_offset` y `label_color`; los puntos, `radius`. `arc` dibuja una tarta sin ejes, una porción por fila según su parte del total de `theta`, en el sentido del reloj desde las 12; acepta `inner_radius` (fracción del radio que queda vacía, en `[0, 1)`: un donut si es mayor que 0), `pad_angle` (radianes vacíos entre porciones), `start_angle` (radianes en sentido antihorario desde el eje x; `math.pi / 2` por defecto), `radius` (unidades de escena) y `label_color`. Las marcas rellenas (`bar`, `histogram`, `area`, `point`, `box`, `violin` y las celdas de `heatmap`) se dibujan sin borde; para trazarlo, usa `chart.layer("marks").stroke(...)`.

Ejemplo:

```
bars = (
  ChartSpec({"method": ["one", "two"], "elapsed": [40, -10], "kind": ["new", "old"]})
  .mark("bar", width=0.72, label_position="outside", label_offset=0.2)
  .encode(
    x="method", y="elapsed", label="elapsed",
    color=Field("kind", scale=Scale.category().colors([BLUE, GOLD])),
  )
)
```

40.1.74. ChartSpec.encode

MÉTODO `encode(*, x: Optional[EncodingLike] = None, y: Optional[EncodingLike] = None, z: Optional[EncodingLike] = None, color: Optional[EncodingLike] = None, size: Optional[EncodingLike] = None, opacity: Optional[EncodingLike] = None, label: Optional[EncodingLike] = None, theta: Optional[EncodingLike] = None, radius: Optional[EncodingLike] = None) -> ChartSpec`

Asigna canales (`x`, `y`, `z`, `color`, `size`, `opacity`, `label`, `theta`, `radius`). Cada canal acepta un nombre de columna, `Field(columna, scale=...)` o `Value(constante)`. En `point`, `size` es el radio en unidades de escena (0.06 por defecto); un `Field` numérico lo reparte entre 0.03 y 0.12 con área proporcional. `color` y `opacity` con `Field` valen por fila; en `line`, `step` y `area`, un `color` categórico dibuja una serie por categoría. `theta` y `radius` son de `arc`: `theta` (obligatorio, no negativo; un valor ausente no ocupa sitio) da a cada fila su parte del círculo, y un campo `radius` dibuja un gráfico de rosa, con cada porción desde el radio interior hacia el exterior en proporción a su valor.

40.1.75. ChartSpec.axes

MÉTODO `axes(*, x: Optional[Axis] = None, y: Optional[Axis] = None, z: Optional[Axis] = None) -> ChartSpec`

Ejes explícitos; los omitidos se infieren. Un dominio explícito se respeta tal cual. Al inferirlos en 2D, el dominio se amplía para que quepan las marcas: las barras incluyen la línea base y media barra de margen, los mapas de calor media celda, el área su línea base, las barras de error su rango más un 5 % y los puntos y líneas un 5 % del rango.

40.1.76. ChartSpec.guides

MÉTODO `guides(*, color: Optional[Guide] = None, size: Optional[Guide] = None, opacity: Optional[Guide] = None) -> ChartSpec`

Leyendas y barras de color para los canales `color`, `size` y `opacity`, derivadas de sus escalas.

40.1.77. ChartSpec.validate

MÉTODO `validate() -> None`

Comprueba canales obligatorios, columnas, clave, escalas y opciones, y lanza un error antes de materializar.

40.1.78. ChartSpec.key

PROPIEDAD `key: Optional[str]`

Columna de identidad de las transiciones, o `None`.

40.1.79. Scale.linear / log / symlog / power / time / category

MÉTODO `linear(domain=None, *, clamp=False) · log(domain=None, *, base=10, clamp=False) · symlog(domain=None, *, base=10, threshold=1, clamp=False) ·`

`power(domain=None, *, exponent=1, clamp=False) · time(domain=None, *, clamp=False) · category(values=None)`

Escala de un canal codificado con `Field`. La misma escala gobierna posiciones, colores, leyendas y barras de color. `category` sin valores los infiere en orden de aparición.

Ejemplo:

```
color = Field("temperature", scale=Scale.symlog((-100, 100), threshold=1))
kind = Field("kind", scale=Scale.category(["new", "old"]).colors(["#38bdf8", "#f97316"]))
```

40.1.80. Scale.colors

MÉTODO `colors(colors: PaletteLike) -> Scale`

Copia de la escala con un rango ordenado de colores: una `Palette`, el nombre de una paleta (`"okabe_ito"`) o colores (`Color`, cadenas CSS/hex o tuplas RGB(A), incluso mezclados). Colores inválidos, nombres desconocidos o una lista vacía lanzan `ValueError`. En un campo de texto, los colores se reparten entre las categorías por turno y vuelven a empezar como una paleta; sin ellos, las categorías toman la paleta del tema. En un campo numérico forman una rampa. En un `heatmap`, los colores y el dominio de la escala del canal `color` sustituyen a la paleta `heatmap` del tema solo en ese gráfico, y `bands=` cuantiza sobre ellos.

40.1.81. Field

CLASE `Field(column, *, scale=None)`

Codifica un canal con una columna, opcionalmente con su escala. Úsalo cuando el valor cambia por fila.

40.1.82. Value

CLASE `Value(value)`

Constante para todo un canal: un número finito, una cadena o un color. Por ejemplo, `Value(BLUE)` da un único color a toda la serie.

40.1.83. Guide.legend / colorbar / disabled

MÉTODO `legend(*, title=None) · colorbar(*, title=None) · disabled()`

`legend` lista cada categoría con una muestra de su color, en el orden de `Scale.category` o de aparición; `colorbar` es la guía continua; `disabled` oculta la guía de un canal.

40.1.84. DataSource.replace / append / version / DataTable.columns

CLASE `DataSource(data, *, key=None) · replace(data) · append(data) · version`

Tabla reemplazable. Tras `replace` o `append`, construye un `ChartSpec` nuevo para capturar la nueva versión inmutable; `version` cuenta los cambios. `DataTable(columns)` es la tabla inmutable equivalente, con `columns` y `len()`.

Ejemplo:

```
source = DataSource({"x": [0, 1], "y": [2, 3]})
before = ChartSpec(source).mark("line").encode(x="x", y="y")
source.append({"x": [2], "y": [5]})
after = ChartSpec(source).mark("line").encode(x="x", y="y")
```

40.1.85. Visualization.chart

FÁBRICA `chart(spec: ChartSpec) -> Chart`

Materializa la receta en un `Chart` con capas semánticas por lotes: las marcas se agrupan por estilo resuelto en lugar de crear una entidad por registro.

40.1.86. Chart.layer

MÉTODO `layer(name: Literal["marks", "axes", "grid", "guides", "labels"]) -> Drawable`

Capa estable como `Drawable`: `marks`, `axes`, `grid`, `guides` o `labels`. La capa `axes` reúne rejilla, ejes, marcas, números y títulos, así que `chart.layer("axes").animate.create()` la mantiene oculta hasta su entrada. Las tres últimas son opcionales: `grid` existe si los ejes dibujan rejilla, `guides` si la especificación tiene leyenda o título de guía (`ChartSpec.guides`) y `labels` en gráficos de barras o de arcos con el canal `label`. La capa `axes` de un gráfico de arcos está vacía. Pedir una que el gráfico no tiene lanza `ValueError` con las capas que sí tiene.

Ejemplo:

```
chart = scene.viz.chart(ChartSpec({"x": [0, 1, 2], "value": [18, 42,
31]}).mark("bar").encode(x="x", y="value"))
scene.play([chart.layer("axes").animate.create(), chart.layer("marks").animate.fade_in()])
```

40.1.87. Chart.drawable

MÉTODO `drawable() -> Drawable`

Objeto raíz del gráfico, para layout, estilo y animaciones genéricas. La opacidad se propaga por todas las capas, incluidas las mallas 3D.

40.1.88. ChartAnimation.grow

MÉTODO `grow() -> Composition`

Entrada de un gráfico de arcos, desde `chart.animate.grow()`: una manecilla gira en el sentido del reloj desde el ángulo inicial, acelerando y frenando, y cada porción se abre en su sitio cuando la manecilla la cruza. La etiqueta de una porción aparece desde que la manecilla pasa por su

centro hasta que la porción se completa; las guías aparecen a lo largo de toda la entrada. En otro tipo de gráfico lanza `ValueError`.

Devuelve: `Composition` — Las animaciones de la entrada, en paralelo.

Ejemplo:

```
sales = {"region": ["North", "South", "East"], "sales": [42, 28, 18]}
spec = ChartSpec(sales, key="region").mark("arc", inner_radius=0.55).encode(theta="sales",
color="region", label="region")
chart = scene.viz.chart(spec)
scene.play([chart.animate.grow()], duration=1.5)
later = ChartSpec({"region": ["North", "South", "West"], "sales": [30, 35, 20]}, key="region")
scene.play([chart.animate.to(later.mark("arc", inner_radius=0.55).encode(theta="sales",
color="region", label="region"))])
```

40.1.89. ChartAnimation.to

MÉTODO `to(target: ChartSpec, *, match_: Literal["key", "index"] = "key", fallback: Literal["error", "crossfade"] = "error") -> Anim`

Transición accesible desde `chart.animate.to(...)`. Entre gráficos 2D, las marcas, ejes y etiquetas que se corresponden se transforman y el resto se funde; al terminar, el gráfico de destino sustituye al original. `point`, `line` y `bar` pueden pasar entre 2D y 3D (con un fundido), y `heatmap` y `surface` comparten rejilla. Entre dos gráficos de arcos, cada porción barre sus ángulos y radios hasta la porción con su clave (su fila con `match_="index"`); una porción sin pareja se cierra sobre su ángulo medio o se abre desde él. Nunca mueve la cámara global.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`target`	`ChartSpec`	obligatorio	Receta de destino.
`match_`	`str`	`"key"`	`"key"` relaciona elementos por la columna de identidad (ambas recetas deben declararla); sin clave, pide `"index"`.
`fallback`	`str`	`"error"`	Con familias de marca incompatibles: `"error"` o `"crossfade"`.

Ejemplo:

```
target = spec.encode(z="height").axes(z=Axis.linear(-2, 2))
scene.play([
  chart.animate.to(target).duration(1.4),
  scene.camera.animate.look_at(eye=(8, 6, 8), target=(0, 0, 0)).duration(1.4),
])
```

40.1.90. Chart.inspect / inspection_enabled / move_to / move_to_3d / scale_to / animate

MÉTODO

Activa metadatos de inspección de los campos indicados en la vista previa. No aparece en capturas ni exportaciones; `inspection_enabled` indica si está activa. `move_to`, `move_to_3d` y `scale_to` devuelven el gráfico.

40.1.91. Campos vectoriales

Un `VectorField` separa la función de sus representaciones: el mismo campo produce flechas, líneas de corriente, advección y partículas.

40.1.92. CoordinateSpace.field / VectorField.dimensions / evaluation

MÉTODO

Evaluador reutilizable de una función que devuelve dos componentes (tres en 3D) y recibe las coordenadas y después `inputs`. No dibuja por sí solo. `dimensions` vale 2 o 3 y `evaluation`, `"python"` en esta ruta.

Ejemplo:

```
from gaanim import Axis, Scene

scene = Scene(frame=(16, 9), background="#0f172a")
plane = scene.viz.cartesian_2d(Axis.linear(-4, 4).ticks(1), Axis.linear(-3, 3).ticks(1))
field = plane.field(lambda x, y: (-y, x))
arrows = field.arrows(resolution=(18, 12), colormap="batlow")
streams = field.streamlines(seeds=(12, 8), max_time=3.5, colormap="vik")
scene.play([plane.animate.create(), arrows.animate.write()], duration=0.8)
scene.play([streams.animate.write().duration(0.8)])
scene.play(streams.flow(1.5, time_width=0.12))
scene.render()
```

40.1.93. VectorField.arrows

MÉTODO `arrows(*, resolution: Optional[tuple[int, int] | tuple[int, int, int]] = None, min_length: float = 0.0, max_length: Optional[float] = None, length_scale: float = 1.0, width: float = 0.02, tip_length: Optional[float] = None, tip_width: Optional[float] = None, color: Optional[ColorLike] = None,`

`colormap: Optional[ColorMapLike] = None, color_range: Optional[tuple[float, float]] = None) -> ArrowVectorField`

Devuelve un `ArrowVectorField` (grupo retenido de astas y puntas) con `drawable()` y `animate`. Se valida todo antes de crear geometría.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`resolution`	`(int, int) | (int, int, int) | None`	`None`	Muestras regulares por eje.
`min_length / max_length / length_scale`	`float`	`0 / 0.28 (0.24 en 3D) / 1`	Límites y escala de longitud en unidades locales.
`width / tip_length / tip_width`	`float`	`0.02 / auto / auto`	Grosor del asta y tamaño de la punta.
`color / colormap`	`ColorLike | ColorMapLike | None`	`None / viridis`	Excluyentes; el mapa usa la magnitud.
`color_range`	`(float, float) | None`	`None`	Rango explícito de magnitudes.

40.1.94. VectorField.streamlines

MÉTODO `streamlines(*, seeds: Optional[tuple[int, int] | tuple[int, int, int]] = None, direction: Literal["forward", "backward", "both"] = "both", tolerance: float = 1e-4, min_step: float = 1e-5, max_step: float = 0.1, max_time: float = 3.0, max_length: Optional[float] = None, max_steps: int = 10_000, stagnation: float = 1e-10, padding: float = 0.05, separation: float = 0.035, width: float =`

`0.02, opacity: float = 1.0, color: Optional[ColorLike] = None, colormap: Optional[ColorMapLike] = None, color_range: Optional[tuple[float, float]] = None) -> StreamLines`

Líneas de corriente deterministas coloreadas por velocidad. Devuelve `StreamLines` con `drawable()` y `animate`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`seeds`	`(int, int) | (int, int, int) | None`	`None`	Resolución determinista de semillas candidatas.
`direction`	`str`	`"both"`	`forward`, `backward` o `both`.
`tolerance / min_step / max_step`	`float`	`1e-4 / 1e-5 / 0.1`	Control adaptativo Dormand–Prince RK45.
`max_time / max_length / max_steps`	`float | int`	`3 / None / 10000`	Límites que hacen reproducible cada trayectoria.
`padding / separation`	`float`	`0.05 / 0.035`	Margen del dominio y distancia mínima entre líneas.

40.1.95. StreamLines.flow / drawable / animate / ArrowVectorField.drawable / animate / FlowParticles.flow / drawable / animate

MÉTODO `StreamLines.flow(duration=2.0, *, time_width=0.15) -> list[Anim] · FlowParticles.flow() -> list[Anim] · drawable() -> Drawable · animate`

`StreamLines.flow` devuelve animaciones que desplazan resaltados más claros por las líneas sin recortar la geometría base; `FlowParticles.flow` devuelve un clip de advección por partícula.

Son finitas y exactas al hacer seek: pasa la lista directamente a `scene.play`. Los tres grupos (`StreamLines`, `ArrowVectorField` y `FlowParticles`) ofrecen `drawable()` para layout y estilo y `animate` para sus entradas y salidas.

40.1.96. VectorField.advect

MÉTODO `advect(target: Drawable, seed: tuple[float, float] | tuple[float, float, float], *, duration: float = 3.0, direction: Literal["forward", "backward", "both"] = "forward", tolerance: float = 1e-4, min_step: float = 1e-5, max_step: float = 0.1, max_time: float = 3.0, max_length: Optional[float] = None, max_steps: int = 10_000, stagnation: float = 1e-10, padding: float = 0.05) -> Anim`

Mueve el centro de un objeto por la trayectoria que parte de `seed` (coordenadas de datos), con el mismo integrador. Devuelve un `Anim` finito por longitud de arco; no deforma la geometría del objeto.

Ejemplo:

```
boat = scene.geometry.dot(0.12).fill(GOLD)
scene.play([field.advect(boat, (2, 0), duration=2)])
```

40.1.97. VectorField.particles

MÉTODO `particles(count: int = 32, *, radius: Optional[float] = None, duration: float = 3.0, tolerance: float = 1e-4, min_step: float = 1e-5, max_step: float = 0.1, max_time: float = 3.0, max_length: Optional[float] = None, max_steps: int = 10_000, stagnation: float = 1e-10, padding: float = 0.05, color: Optional[ColorLike] = None, colormap: Optional[ColorMapLike] = None, color_range: Optional[tuple[float, float]] = None, opacity: float = 1.0) -> FlowParticles`

Crea `count` partículas con semillas de Halton deterministas (puntos en 2D, esferas en 3D). El `FlowParticles` resultante queda oculto hasta su primera entrada (`create`, `write`, `fade_in`, `grow_from_center`) y `flow()` devuelve un clip de advección por partícula.

Ejemplo:

```
particles = field.particles(24, duration=2)
scene.play([particles.animate.fade_in()])
scene.play(particles.flow())
```

Los colores y mapas explícitos de flechas y líneas tienen prioridad sobre las reglas `plot` del tema, que sigue controlando el fondo y los ejes.

40.1.98. Espacios 3D

EXPERIMENTAL

La API 3D es experimental: todavía tiene errores conocidos y puede cambiar entre versiones. Úsala para explorar, pero no la tomes como estable.

Ejes 3D con rejillas por plano, superficies y curvas paramétricas. Combínalos con la cámara en perspectiva de Escena.

40.1.99. Visualization.cartesian_3d

FÁBRICA `cartesian_3d(x: Axis, y: Axis, z: Axis, *, size: tuple[float, float, float] = (10.0, 8.0, 6.0), grid: bool = True, axes: bool = True, ticks: bool = True, numbers: bool = True, labels: bool = True, x_axis: Optional[bool] = None, y_axis: Optional[bool] = None, z_axis: Optional[bool] = None, xy_grid: Optional[bool] = None, xz_grid: Optional[bool] = None, yz_grid: Optional[bool] = None, x_ticks: Optional[bool] = None, y_ticks: Optional[bool] = None, z_ticks: Optional[bool] = None, x_numbers: Optional[bool] = None, y_numbers: Optional[bool] = None, z_numbers: Optional[bool] = None, x_labels: Optional[bool] = None, y_labels: Optional[bool] = None, z_labels: Optional[bool] = None) -> Cartesian3D`

Devuelve un `Cartesian3D` (un `CoordinateSpace3D`) con capas estables. Las rejillas ocultas no emiten segmentos ni duplican las aristas de los ejes; las etiquetas miran a la cámara.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`x / y / z`	`Axis`	obligatorio	Ejes lineales o temporales reutilizables.
`size`	`(float, float, float)`	`(10.0, 8.0, 6.0)`	Tamaño positivo en el mundo; elígelo según la distancia de la cámara.
`xy_grid / xz_grid / yz_grid`	`bool | None`	`None`	Visibilidad de cada plano; `None` hereda `grid`.
`x_axis / y_axis / z_axis, x_ticks / …, x_numbers / …, x_labels / …`	`bool | None`	`None`	Eje, marcas, números

y título de cada eje, con el prefijo `x_`, `y_` o `z_`; `None` hereda el interruptor global.

Ejemplo:

```
world = scene.viz.cartesian_3d(
    Axis.linear(-3, 3).ticks(1).label("x"),
    Axis.linear(-3, 3).ticks(1).label("y"),
    Axis.linear(-2, 2).ticks(1).label("z"),
    size=(6, 6, 4), xz_grid=False, yz_grid=False,
)
```

40.1.100. CoordinateSpace3D.surface

MÉTODO `surface(function: Callable[..., float], *, resolution: tuple[int, int] = (64, 48), inputs: Sequence[Parameter | Variable | Computed | TimeInput] = (), color_map: Optional[ColorMapLike] = None) -> Drawable`

Superficie `z = f(x, y)` muestreada en una rejilla de `resolution`, con `inputs` reactivas como `plot`. Los vértices se colorean por altura, de la más baja a la más alta, con `color_map`: un `ColorMap` o el nombre de un mapa integrado, como `"viridis"` o `"plasma"`. Sin él, usa una rampa de azul, verde y rojo.

Ejemplo:

```
saddle = world.surface(lambda x, y: 0.1 * (x * x - y * y), resolution=(32, 24))
```

40.1.101. CoordinateSpace3D.parametric

MÉTODO `parametric(function: Callable[..., tuple[float, float, float]], domain: tuple[float, float], *, samples: int = 320, inputs: Sequence[Parameter | Variable | Computed | TimeInput] = ()) -> Drawable`

Curva 3D `t → (x, y, z)` sobre `domain` con `samples` muestras e `inputs` reactivas.

40.1.102. CoordinateSpace3D.field

MÉTODO `field(function: Callable[..., tuple[float, float, float]], *, inputs: Sequence[Parameter | Variable | Computed | TimeInput] = ()) -> VectorField`

Campo vectorial 3D; sus flechas, líneas de corriente y partículas funcionan como en 2D, con color por vértice.

40.1.103. CoordinateSpace3D.layer / move_to_3d / scale_to / data_to_local / local_to_data / drawable / animate

MÉTODO

Capa 3D animable por separado: `grid`, `axes`, `ticks`, `numbers` o `labels`. `move_to_3d`, `scale_to`, `data_to_local` y `local_to_data` completan el espacio.

40.1.104. Límites

La inspección interactiva solo existe en la vista previa. La exportación interactiva, los dashboards, las facetas, los volúmenes y las isosuperficies aún no forman parte de esta API. La animación de cámara es global, explícita y componible.

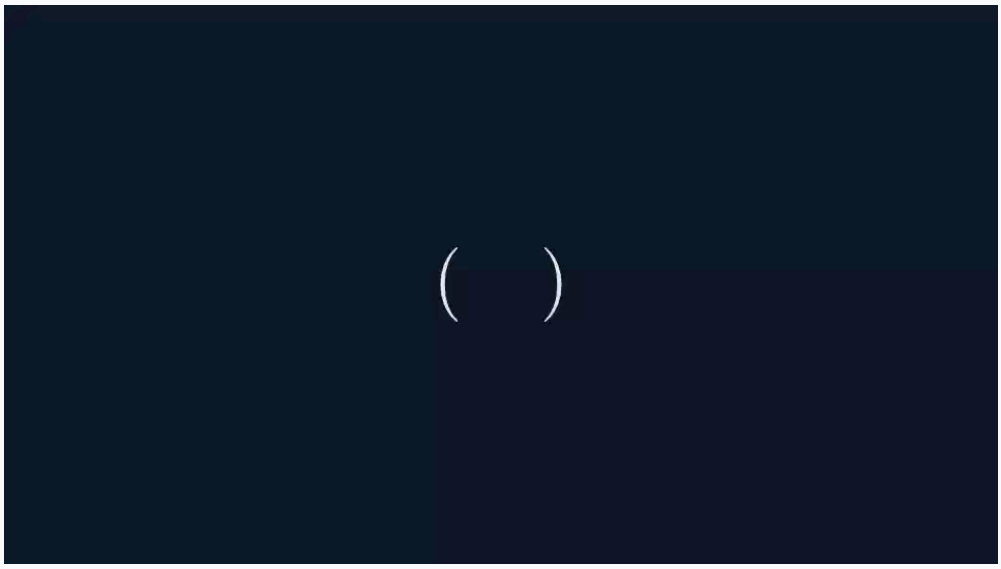
CAPÍTULO

41. Matrices

scene.viz.matrix: matrices seleccionables y mutables, con morph entre estados y álgebra simbólica opcional

`scene.viz.matrix(...)` crea una matriz cuyas entradas son objetos independientes. Una celda, fila, columna, diagonal o bloque puede recibir estilo y animación sin desarmar la composición, y la matriz puede cambiar de forma o transformarse en otra.

```
from gaanim import GOLD, Scene
scene = Scene(frame=(16, 9), background="#0f172a")
m = scene.viz.matrix([[1, 2, 3], [4, 5, 6]], delimiters="parentheses")
m[0, :].fill(GOLD)
scene.play(m.entries.animate.write(order="spiral_in", stagger=0.05).duration(0.4))
scene.render()
```



`Matrix` y sus tipos auxiliares (`MatrixSelection`, `MatrixEntry`…) se importan desde `gaanim` y se definen en su módulo Python, por eso las fichas de esta página los nombran por la variable (`matrix.row`) y no por la clase. `Matrix` es un `Drawable`: se coloca, escala y anima como cualquier objeto, y los métodos heredados, `place` incluido, devuelven la propia matriz. Al escalarla crecen también las entradas y los delimitadores.

41.1.1. Crear una matriz

La fábrica y los datos que acepta.

41.1.2. Visualization.matrix

FÁBRICA `matrix(data: Any, *, row_gap: float = 0.24, column_gap: float = 0.24, delimiter_gap: float = 0.12, delimiters: Literal["brackets", "parentheses", "braces", "bars", "double_bars", "none"] = "brackets", delimiter_size: float | None = None, delimiter_weight: int = 300, row_labels: Sequence[Any] | None = None, column_labels: Sequence[Any] | None = None, label_mode: Literal["math",`

```
"text"] = "math", cell_mode: Literal["math", "text"] = "math", entry_style: Any |
None = None, label_style: Any | None = None, cell_factory: Callable[[Any, int,
int], Drawable] | None = None, numeric_format: str = "g", fractions:
Literal["stacked", "inline"] = "stacked") -> Matrix
```

Los escalares se convierten en ecuaciones Typst; un `Drawable` se conserva tal cual. Filas desiguales, etiquetas incompatibles, modos, grosores o delimitadores inválidos lanzan `ValueError` o `TypeError` antes del render.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`data`	`Sequence[Sequence[valor]] | sympy.MatrixBase`	obligatorio	Datos rectangulares no vacíos.
`row_gap / column_gap`	`float`	`0.24 / 0.24`	Separación entre filas y columnas, en unidades de escena.
`delimiter_gap`	`float`	`0.12`	Espacio entre las entradas y los delimitadores.
`delimiters`	`str`	`"brackets"`	`brackets`, `parentheses`, `braces`, `bars`, `double_bars` o `none`.
`delimiter_size / delimiter_weight`	`float | None / int`	`None / 300`	Tamaño y grosor tipográfico, de 100 a 900.

			Sin tamaño, los delimitadores abarcan las filas: siguen el tamaño de `entry_style` y la separación entre filas.
`row_labels / column_labels`	`Sequence | None`	`None`	Etiquetas externas alineadas con filas o columnas.
`cell_mode / label_mode`	`"math" | "text"`	`"math"`	Entradas y etiquetas como matemática Typst o texto plano.
`entry_style / label_style`	`TextStyle | None`	`None`	Estilo de entradas y etiquetas.
`cell_factory`	`Callable[[valor, fila, columna], Drawable] | None`	`None`	Crea un objeto propio para cada celda.

`numeric_format`	`str`	`"g"`	Formato de Python para los números decimales.
`fractions`	`"stacked" | "inline"`	`"stacked"`	Los racionales exactos como fracciones apiladas o, con `"inline"`, en una línea (`3/2`), tan altas como las demás entradas.

Ejemplo:

```
costs = scene.viz.matrix(
    [[12.5, 8], [3, 40.25]],
    row_labels=["A", "B"], column_labels=["ida", "vuelta"],
    label_mode="text", numeric_format=".1f", delimiters="brackets",
)
```

41.1.3. MatrixEntry

CLASE `MatrixEntry(value, key=None, style=None)`

Valor de una celda con una clave estable opcional (`key`) para emparejarlo en `morph_to` y un estilo de texto local. Los predicados de `where` y `MatrixSelection.values` reciben entradas de este tipo; el número está en `.value`.

Ejemplo:

```
m = scene.viz.matrix([[MatrixEntry("a", key="a"), 0], [0, MatrixEntry("b", key="b")]])
```

41.1.4. matrix.shape / nrows / ncols

PROPIEDAD `shape: tuple[int, int] · nrows: int · ncols: int · entries: MatrixSelection · row_labels · column_labels · drawable`

Dimensiones, todas las entradas como selección, las etiquetas como tuplas de `Drawable` y el objeto raíz de la composición.

41.1.5. Seleccionar entradas

Grupos estables de celdas que se estilizan y animan juntos.

41.1.6. matrix[fila, columna]

MÉTODO `matrix[r, c] -> Drawable · matrix[filas, columnas] -> MatrixSelection`

Con dos enteros devuelve la celda; con *slices* o listas de índices, una `MatrixSelection`. Admite índices negativos.

Ejemplo:

```
m = scene.viz.matrix([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
m[1, 1].fill(RED)
m[:, 0].fill(GOLD)
```

41.1.7. matrix.row / column / block

MÉTODO `row(index) · column(index) · block(rows, columns) -> MatrixSelection`

Una fila, una columna o un bloque; `rows` y `columns` aceptan un índice, un *slice* o una lista.

Ejemplo:

```
m = scene.viz.matrix([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
m.block(slice(0, 2), [1, 2]).fill(BLUE)
scene.play(m.row(2).animate.indicate(order="row_major", stagger=0.1))
```

41.1.8. matrix.diagonal / anti_diagonal

MÉTODO `diagonal(offset=0) · anti_diagonal(offset=0) -> MatrixSelection`

La diagonal principal o la secundaria, desplazada `offset` columnas.

41.1.9. matrix.where

MÉTODO `where(mask_or_predicate) -> MatrixSelection`

Selecciona con una máscara rectangular de booleanos o con un predicado `f(entrada, fila, columna)`, donde `entrada` es un `MatrixEntry`.

Ejemplo:

```
m = scene.viz.matrix([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
m.where(lambda entry, row, column: entry.value % 2 == 0).fill(GOLD)
```

41.1.10. MatrixSelection

CLASE `fill(color) · opacity(value) · offset(dx, dy) · coordinates · values · animate`

Secuencia estable de celdas. `fill` y `opacity` cambian su estilo al instante, `offset` desplaza las celdas sin salir del layout, `coordinates` y `values` devuelven sus posiciones y entradas, y `animate` da un `MatrixSelectionAnimation`.

41.1.11. MatrixSelectionAnimation

CLASE `write · create · fade_in · fade_out · indicate · wiggle · fill · opacity · scale_by · rotate_by (order=…, stagger=…, seed=0) · duration · easing`

Animación de todas las celdas de una selección. `order` recorre las celdas en `simultaneous`, `row_major`, `column_major`, `main_diagonal`, `anti_diagonal`, `spiral_in`, `spiral_out` o `random` (reproducible con `seed`), o en el orden de una lista de coordenadas o de una función `(fila, columna) -> clave`. `stagger` es el retraso entre celdas; con `simultaneous` todas empiezan juntas. Sin `order`, empiezan juntas o, si das un `stagger`, fila a fila. Se pasa directamente a `scene.play`.

Ejemplo:

```
m = scene.viz.matrix([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
scene.play(m.entries.animate.fade_in(order="main_diagonal", stagger=0.08).duration(0.8))
scene.play(m.diagonal().animate.fill(GOLD).duration(0.4))
```

41.1.12. Cambiar la estructura

Cambios de valores, filas y columnas que recomponen la matriz en el cursor. La matriz conserva su lugar, su transformación y sus animaciones: cambian los hijos de su cuadrícula, no la cuadrícula.

41.1.13. matrix.set

MÉTODO `set(row, column, value) -> Drawable`

Sustituye el valor de una celda y devuelve su nuevo objeto.

Ejemplo:

```
m = scene.viz.matrix([[1, 2], [3, 4]])
scene.wait(0.5)
m.set(0, 1, 7).fill(GOLD)
```

41.1.14. matrix.insert_row / remove_row / insert_column / remove_column

MÉTODO `insert_row(index, values) · remove_row(index) · insert_column(index, values) · remove_column(index) -> Matrix`

Añade o quita filas y columnas; los valores nuevos deben tener la longitud correcta.

Ejemplo:

```
m = scene.viz.matrix([[1, 2], [3, 4]])
m.insert_row(1, [0, 0]).remove_column(0)
```

41.1.15. matrix.swap_rows / swap_columns / reorder_rows / reorder_columns

MÉTODO `swap_rows(first, second) · swap_columns(first, second) · reorder_rows(order) · reorder_columns(order) -> Matrix`

Intercambia o permuta filas y columnas; `order` es una permutación completa de índices.

Ejemplo:

```
m = scene.viz.matrix([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
m.swap_rows(0, 2)
m.reorder_columns([2, 0, 1])
```

41.1.16. matrix.become

MÉTODO `become(data) -> Matrix`

Sustituye todos los datos, incluso con otra forma. Un `delimiter_size` explícito crece o se encoge con el número de filas, como el automático.

41.1.17. matrix.morph_to

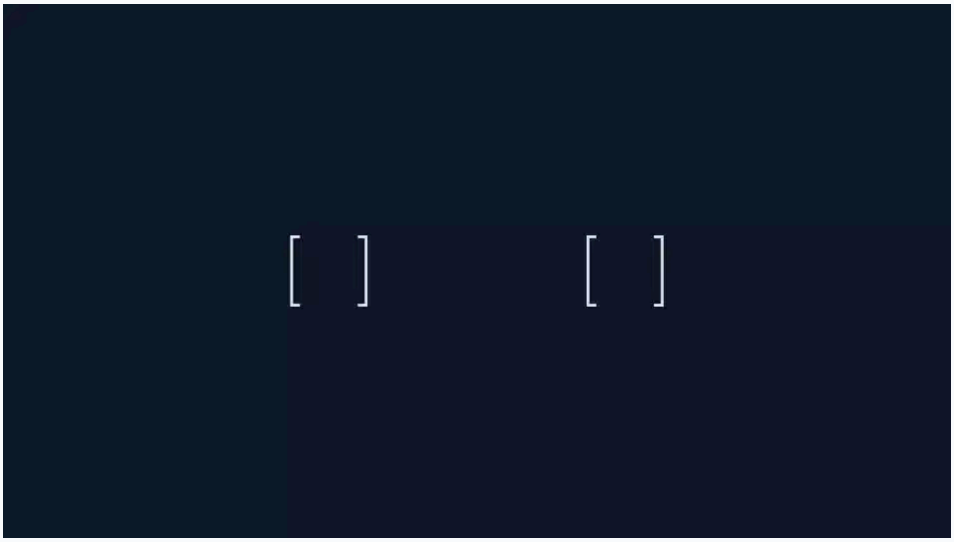
MÉTODO `morph_to(target: Matrix, *, match="auto", stagger=0.0) -> MatrixSelectionAnimation`

Transforma la matriz en `target`, que es la que queda al final. Con `match="auto"` empareja primero por las claves de `MatrixEntry`, después por valores y por último por posiciones. Cada entrada emparejada viaja hasta la suya y le cede el lugar; las que no tienen pareja se desvanecen, las del destino sin pareja aparecen y los delimitadores y etiquetas se transforman en los del destino. El destino se muestra al empezar, aunque estuviera oculto con `hidden()`. Configura el tiempo con `.duration(...)` y `.easing(...)`.

Ejemplo:

```
from gaanim import Scene
scene = Scene(frame=(16, 9), background="#0f172a")
a = scene.viz.matrix([[1, 2], [3, 4]]).move_to(-2.5, 0)
b = scene.viz.matrix([[1, 3], [2, 4]]).move_to(2.5, 0)
scene.play(a.entries.animate.write(stagger=0.05).duration(0.5))
```

```python
scene.play(a.morph_to(b).duration(1.0))
scene.render()
```



41.1.18. Álgebra simbólica

Operaciones exactas con SymPy que devuelven el resultado y los pasos animables.

Instala el extra `gaanim[algebra]` para usarlas. `add`, `subtract`, `matmul`, `hadamard`, `scalar_multiply`, `transpose`, `determinant`, `inverse`, `rank`, `trace`, `rref`, `lu`, `qr` y `eigen` aceptan `exact=True, precision=None` y devuelven una `MatrixDerivation[T]`: `T` es `Matrix` para resultados matriciales, `Drawable` para escalares y `tuple[Matrix, Matrix]` para descomposiciones. Los fallos exactos no se aproximan en silencio: pasa `exact=False, precision=N` de forma explícita. `to_sympy()` devuelve la matriz de SymPy. Una matriz resultado toma las opciones de la matriz de origen (separaciones, delimitadores, `entry_style`, `cell_factory`, `numeric_format`, `fractions`), no sus etiquetas; un escalar es una ecuación como `rank = 2`.

41.1.19. MatrixDerivation

CLASE `value · result · steps · animations(*, order="row_major", stagger=0.04)`

Resultado de una operación: `value` es el valor exacto de SymPy, `result` el objeto visual y `steps` los pasos (`MatrixStep`, con `kind`, `sources`, `target` y `expression`). `animations()` devuelve una `MatrixSelectionAnimation` que recorre los pasos. Un error de álgebra lanza `MatrixAlgebraError`, subclase de `ValueError`.

Ejemplo:

```python
from gaanim import Scene
scene = Scene(frame=(16, 9), background="#0f172a")
a = scene.viz.matrix([[1, 2], [3, 4]]).move_to(-3, 0)
b = scene.viz.matrix([[2, 0], [1, 2]])
product = a.matmul(b)
product.result.move_to(3, 0)
scene.play(product.animations().duration(0.7))
scene.render()
```

CAPÍTULO

42. Medios

Fábricas de scene.media: imágenes, SVG, video y Lottie

`scene.media` carga archivos locales como objetos de la escena: imágenes, SVG, video y animaciones Lottie. Todos son `Drawable`, así que se colocan, escalan y animan como cualquier forma. Una ruta relativa se resuelve desde la carpeta de recursos de la escena si se fijó con `scene.assets.load_project()` o `scene.assets.assets_dir(...)`; si no, desde el directorio de trabajo, la carpeta `assets/` del script y la carpeta del script, en ese orden (ver Recursos); el audio está en Audio.

```
from gaanim import Scene

scene = Scene(frame=(16, 9), background="#0f172a")
photo = scene.media.image("assets/cover.png").frame(6, 3.4, fit="cover").move_to(-3.5, 0)
robot = scene.media.svg("assets/robot.svg").scale_to(2).move_to(3.5, 0)
scene.play([photo.animate.fade_in(), robot.animate.write()], duration=1.2)
scene.render()
```

42.1.1. Imágenes

PNG, JPEG y WebP con marco, recorte y calidad de muestreo. La textura decodificada se comparte entre objetos que usan la misma ruta.

42.1.2. MediaLibrary.image

FÁBRICA `image(path: str, *, width: Optional[float] = None, height: Optional[float] = None, fit: str = "contain", crop: Optional[tuple[float, float, float, float]] = None, quality: Literal["low", "medium", "high"] = "medium") -> Image`

Crea una `Image` que conserva su tipo al encadenar setters. Ajustes, recortes, medidas o calidades inválidos lanzan `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`path`	`str`	obligatorio	Archivo PNG, JPEG o WebP.
`width / height`	`float | None`	`None`	Tamaño de destino en unidades de escena; sin ellos, la imagen cabe en el área segura.
`fit`	`str`	`"contain"`	Con `width` y `height`, como en `Image.frame`: `contain` conserva la imagen entera, `cover` llena el marco recortando el sobrante por igual a ambos lados y `stretch` la deforma.
`crop`	`(x, y, w, h) | None`	`None`	Recorte de la fuente en píxeles, con origen arriba a la izquierda.
`quality`	`str`	`"medium"`	Muestreo: `low` (vecino más cercano), `medium` (bilineal) o `high` (bicúbico).

Ejemplo:

```
logo = scene.media.image("assets/logo.webp", width=3)
banner = scene.media.image("foto.jpg", width=8, height=2, fit="cover")
```

42.1.3. Image.frame

MÉTODO `frame(width: float, height: float, fit: Literal["contain", "cover", "stretch"] = "contain") -> Self`

Fija un marco centrado en unidades de escena. `contain` conserva la proporción con bandas transparentes, `cover` llena el marco recortando el sobrante y `stretch` deforma. Después de congelarse la declaración, crea un corte reversible en el cursor.

Ejemplo:

```
foto = scene.media.image("foto.jpg").frame(8, 4.5, fit="cover")
```

42.1.4. Image.crop

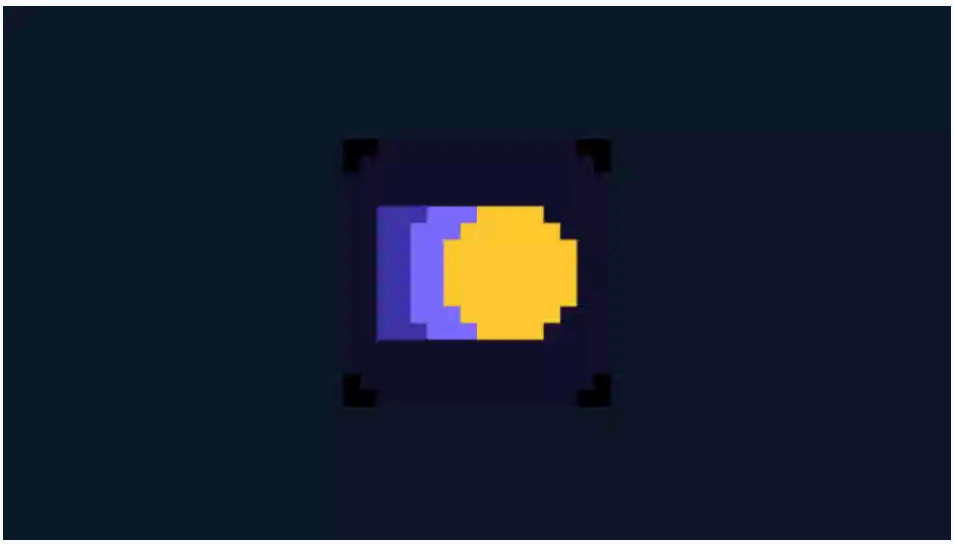
MÉTODO `crop(x: float, y: float, width: float, height: float, *, normalized: bool = False) -> Self`

Selecciona un rectángulo de la fuente sin mover el marco. El origen es la esquina superior izquierda, con Y hacia abajo; `normalized=True` usa fracciones del tamaño original.

`obj.animate.crop(...)` interpola el recorte para hacer paneos o zoom dentro del marco. Rectángulos fuera de la imagen lanzan `ValueError`.

Ejemplo:

```
from gaanim import Scene
scene = Scene(frame=(16, 9), background="#0f172a")
foto = scene.media.image("foto.jpg").frame(8, 4.5, fit="cover")
scene.play([
    foto.animate.crop(0.25, 0.25, 0.5, 0.5, normalized=True).duration(1.5)
])
scene.render()
```



42.1.5. Image.quality

MÉTODO `quality(value: Literal["low", "medium", "high"]) -> Self`

Cambia la calidad de muestreo (`"low"`, `"medium"` o `"high"`) en el cursor.

42.1.6. Image.source_width / source_height

PROPIEDAD `source_width: int · source_height: int`

Tamaño original en píxeles, independiente del recorte y las transformaciones.

Ejemplo:

```
foto = scene.media.image("foto.jpg")
aspect = foto.source_width / foto.source_height
```

42.1.7. SVG

Ilustraciones vectoriales importadas como una jerarquía real de caminos y grupos, con acceso a cada parte por su `id`.

42.1.8. MediaLibrary.svg

FÁBRICA `svg(path: str) -> Drawable`

Importa geometría, degradados, transformaciones, `clipPath` y `feGaussianBlur` como caminos vectoriales normales. El documento se importa a 100 píxeles SVG por unidad lógica, la misma equivalencia que usa el resto del motor (el trazo por defecto de 0.03 equivale a 3 px), con el centro del documento en el origen. `scale_to` y `rotate_to` usan como pivote el centro de lo que el SVG dibuja, no el del documento: si el dibujo no está centrado en su lienzo, tras escalarlo el centro del documento ya no queda en el origen; colócalo con `move_to`, que usa el centro visual, o fija el pivote con `with_pivot(0, 0)` antes de escalar. Las partes (`part(id)`) se colocan después de la raíz, así que moverlas no cambia dónde queda el SVG. El tamaño en píxeles es el `width`/`height` del documento, o el `viewBox` si faltan: un SVG de 200 × 200 px mide 2 × 2 unidades en el marco de 16 × 9. Trazos, degradados, `clipPath`, texto y filtros escalan con la geometría. `scale_to(factor)` cambia el tamaño de todo el SVG; los anchos de trazo fluidos aplicados a la raíz o a una parte (`part(id)`), fijos o animados, siguen en unidades lógicas de escena aunque el SVG se escale. Como todo trazo, ninguno sigue una animación de escala posterior. Los detalles de compatibilidad están en Recursos.

Ejemplo:

```
from gaanim import BLUE, Scene
scene = Scene(frame=(16, 9), background="#f8fafc")
# robot.svg mide 200 px, es decir 2 unidades: scale_to(2.5) lo deja en 5
robot = scene.media.svg("assets/robot.svg").scale_to(2.5).move_to(0, 0)
robot.part("left-arm").fill(BLUE)
scene.play([robot.animate.write().duration(0.8)])
scene.play([robot.part("head").animate.shift_by(0, 0.4).duration(0.6)])
scene.render()
```


42.1.9. Drawable.part

MÉTODO `part(id: str) -> Drawable`

Devuelve la parte con ese `id` de un SVG como un `Drawable` propio. Los nombres distinguen mayúsculas; un nombre desconocido lanza `KeyError` con la lista de nombres disponibles. El estilo de un grupo llega a todos sus caminos.

Ejemplo:

```
robot = scene.media.svg("assets/robot.svg").scale_to(2)
head = robot.part("head").fill(GOLD)
```

42.1.10. Drawable.parts

MÉTODO `parts() -> tuple[str, ...]`

Tupla con los nombres que acepta `part`.

Ejemplo:

```
print(scene.media.svg("assets/robot.svg").parts())

('body', 'elbow', 'forearm', 'head', 'left-arm', 'torso', 'upper-arm')
```

42.1.11. Video

Clips MP4 sincronizados con la línea de tiempo, con su audio. Requieren `ffmpeg` y `ffprobe`.

42.1.12. MediaLibrary.video

FÁBRICA `video(path: str, *, width: Optional[float] = None, height: Optional[float] = None, fit: Literal["contain", "cover", "stretch"] = "contain", crop: Optional[tuple[float, float, float, float]] = None, quality: Literal["low", "medium", "high"] = "medium", offset: float = 0.0, duration: Optional[float] = None, loop: bool = False, speed: float = 1.0, audio: bool = True, volume: float = 1.0) -> Video`

Declarar el video no inicia fotogramas ni audio: `scene.play([clip])` fija su inicio absoluto. Un video finito sin bucle aporta su duración al lote; con bucle no alarga la línea de tiempo. Rangos inválidos lanzan `ValueError` y los fallos de medios, `RuntimeError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`path`	`str`	obligatorio	Archivo MP4 local.
`width / height / fit / crop / quality`	—	—	Igual que en `image`.
`offset`	`float`	`0.0`	Inicio dentro de la fuente, en segundos.

`duration`	`float | None`	`None`	Duración seleccionada; `None` llega al final.
`loop`	`bool`	`False`	Repite el intervalo seleccionado.
`speed`	`float`	`1.0`	Velocidad positiva; conserva el tono del audio.
`audio`	`bool`	`True`	Reproduce y exporta la primera pista de audio.
`volume`	`float`	`1.0`	Ganancia no negativa.

Ejemplo:

```python
from gaanim import Scene
scene = Scene(frame=(16, 9), background="#0f172a")
clip = scene.media.video("clip.mp4", width=9, duration=1.5)
scene.play([clip.animate.fade_in().duration(0.3)])
scene.play([clip])
scene.render()
```

42.1.13. Video.segment

MÉTODO `segment(*, start: float, end: float, speed: Optional[float] = None, audio: Optional[bool] = None, volume: Optional[float] = None) -> VideoSegment`

Devuelve un `VideoSegment` inerte hasta pasarlo a `scene.play` o a una composición. Dura `(end - start) / speed` segundos de escena y no hereda `offset`, `duration` ni `loop`. Antes del primer fragmento se ve el póster; entre fragmentos y después del último se mantiene el último fotograma, sin audio. Cada fragmento se programa una sola vez: para repetirlo crea otro. Los solapamientos sobre el mismo video, estirar composiciones con medios y mezclar fragmentos con `scene.play([video])` lanzan `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`start / end`	`float`	obligatorio	Segundos absolutos de la fuente, con `0 <= start < end <= source_duration`.
`speed / audio / volume`	`float | bool | None`	`None`	`None` hereda el valor del video.

Ejemplo:

```
video = scene.media.video("clip.mp4").frame(8, 4.5)
scene.play([video.segment(start=2, end=5)])
scene.wait(1)
scene.play([video.segment(start=5, end=8, speed=2)])
```

42.1.14. Video.frame / crop / quality

MÉTODO `frame(width, height, fit="contain") · crop(x, y, width, height, *, normalized=False) · quality(value) -> Self`

Mismo comportamiento que en `Image`; los cambios de fotograma conservan la calidad.

42.1.15. Video.source_duration / frame_rate / source_width / source_height

PROPIEDAD `source_duration: float · frame_rate: float · source_width: int · source_height: int`

Duración completa de la fuente en segundos (antes de recortes o cambios de velocidad), fotogramas por segundo y tamaño original en píxeles.

Ejemplo:

```
clip = scene.media.video("clip.mp4")
print(clip.source_duration, clip.frame_rate, clip.source_width, clip.source_height)

10.0 10.0 160 90
```

42.1.16. Lottie y dotLottie

Animaciones vectoriales exportadas desde After Effects, Cavalry u otras herramientas, renderizadas con Velato dentro de la misma escena Vello.

42.1.17. MediaLibrary.lottie

FÁBRICA `lottie(path: str, *, animation_id: Optional[str] = None, theme_id: Optional[str] = None, state_machine_id: Optional[str] = None, width: Optional[float] = None, height: Optional[float] = None, fit: Literal["contain", "cover", "stretch"] = "contain", offset: float = 0.0, duration: Optional[float] = None, loop: bool = False, speed: float = 1.0) -> Lottie`

Declarar el clip muestra su primer fotograma; `scene.play([clip])` fija el inicio de la reproducción, que sigue el reloj de la escena también al saltar hacia atrás. `fade_in`, `write` y `create` pueden presentar el primer fotograma antes de reproducirlo. Una máquina de estados no alarga la escena: usa `scene.wait`. Opciones inválidas lanzan `ValueError`; fallos de archivo, JSON o imágenes, `RuntimeError`. Los detalles de compatibilidad están en Recursos.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`path`	`str`	obligatorio	Lottie JSON o paquete dotLottie v1/v2 local.
`animation_id / theme_id / state_machine_id`	`str | None`	`None`	Selectores del paquete. Animación y máquina de estados son excluyentes; un JSON no acepta selectores.
`width / height / fit`	—	—	Igual que en `image`.

`offset / duration / loop / speed`	—	`0 / hasta el final / False / 1`	Intervalo reproducido, como en `video`.

Ejemplo:

```
from gaanim import Scene, sequence
scene = Scene(frame=(16, 9), background="#0f172a")
clip = scene.media.lottie("assets/pulse.json", width=3)
scene.play(sequence(clip.animate.fade_in().duration(0.3), clip))
scene.render()
```

42.1.18. Lottie.set_input

MÉTODO `set_input(name: str, value: float | int | bool | str) -> Lottie`

Fija una entrada tipada de la máquina de estados en el cursor (o el estado inicial antes de `play`) y devuelve el clip sin avanzar el tiempo. Entradas desconocidas, tipos incompatibles o números no finitos lanzan `ValueError`; `bool` es distinto de los números.

Ejemplo:

```
clip = scene.media.lottie("assets/button.lottie", state_machine_id="main", width=4)
clip.set_input("active", False)
scene.play([clip])
scene.wait(1)
clip.set_input("active", True)
scene.wait(1)
```

42.1.19. Lottie.fire_event

MÉTODO `fire_event(name: str) -> Lottie`

Dispara un evento declarado en el cursor y devuelve el clip. Requiere que el clip esté activo. Las órdenes simultáneas conservan su orden de declaración y se reproducen igual al buscar y exportar.

Ejemplo:

```
clip.fire_event("reset")
scene.wait(1)
```

42.1.20. Lottie.set_theme

MÉTODO `set_theme(id: Optional[str] = None) -> Lottie`

Aplica un tema estático del paquete en el cursor; `None` restaura el diseño base. Antes de activar el clip fija el tema inicial.

Ejemplo:

```
clip.set_theme("gold")
scene.wait(1)
```

42.1.21. Lottie.animation_ids / theme_ids / state_machine_ids

PROPIEDAD `animation_ids: list[str] · theme_ids: list[str] · state_machine_ids: list[str]`

Identificadores disponibles en el paquete, en el orden del manifiesto; vacíos para un JSON.

Ejemplo:

```
package = scene.media.lottie("assets/button.lottie")
print(package.animation_ids, package.theme_ids, package.state_machine_ids)

['idle', 'active'] ['gold'] ['main']
```

42.1.22. Lottie.warnings

PROPIEDAD `warnings: list[str]`

Lista de características de la fuente que Velato omite o aproxima. Revísala si el clip no se ve como en la herramienta de origen. `source_width`, `source_height`, `frame_rate` y `source_duration` describen la composición original.

CAPÍTULO

43. Audio

Pistas sincronizadas, efectos de sonido anclados, narración grabada en el editor y mezcla en la exportación de video

Música, efectos y narración sincronizados con la línea de tiempo. Declaras una pista con `scene.media.audio(...)` y la activas con `scene.play([pista])`. La vista previa la reproduce siguiendo el reloj de la escena (pausa, seek y velocidad incluidos) y la exportación a MP4 o WebM la mezcla con el video.

43.1.1. Pistas de audio

Archivos de sonido que empiezan en el cursor donde los activas.

43.1.2. MediaLibrary.audio

FÁBRICA `audio(path: str, *, duration: Optional[float] = None, end: Optional[float] = None, volume: float = 1.0, fade_in: float = 0.0, fade_out: float = 0.0) -> Audio`

Declara una pista validada. La declaración no cambia la línea de tiempo: `scene.play([pista])` fija su inicio en el cursor absoluto de esa llamada. Rutas o tiempos inválidos lanzan `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`path`	`str`	obligatorio	Archivo de audio; una ruta relativa usa la carpeta de assets, igual que imágenes y SVG.
`duration`	`float | None`	`None`	Recorta la pista y la hace participar en la duración del lote. Sin ella, la pista suena de fondo sin alargar la línea de tiempo.
`end`	`float | None`	`None`	Corta la pista en ese segundo del archivo **sin** alargar el lote: música de fondo que suena bajo las escenas siguientes y termina donde quieres. No se combina con `duration`.
`volume`	`float`	`1.0`	Ganancia lineal.
`fade_in / fade_out`	`float`	`0.0`	Fundidos de entrada y salida en segundos; `fade_out` necesita `duration` o `end` y termina justo en el corte.

Ejemplo:

```python
from gaanim import Scene

scene = Scene()
scene.assets.assets_dir("assets")

music = scene.media.audio("music.ogg", volume=0.35)
pop = scene.media.audio("pop.wav", duration=0.4, volume=0.8, fade_in=0.02)
scene.play([music])
scene.wait(1.5)
scene.play([pop])

scene.render()
```

Una narración grabada con duración fija alarga el lote y termina con un fundido:

```python
narration = scene.media.audio(
    "narration.m4a",
    duration=7.5,
    volume=0.9,
    fade_in=0.15,
    fade_out=0.25,
)
scene.play([narration])
```

Música de fondo que dura 30 s y se funde al final, mientras la escena sigue programándose desde el mismo cursor:

```python
musica = scene.media.audio("music.ogg", end=30.0, fade_out=1.5, volume=0.5)
scene.play([musica])  # no mueve el cursor: lo que sigue suena sobre la música
```

43.1.3. Audio

CLASE `scene.media.audio(...) -> Audio`

Declaración de audio ligada a la escena que la creó. Solo se usa pasándola a `scene.play`.

43.1.4. Efectos de sonido anclados

Sonidos cortos que se colocan en la línea de tiempo sin `scene.play`. Usan el mismo mezclador que `scene.media.audio`: suenan en la vista previa (con pausa, seek y velocidad) y MP4/WebM los mezclan con el video. Un seek al segundo `t` oye lo mismo que la reproducción continua, porque cada efecto tiene un inicio absoluto fijado al componer la escena. Ninguno alarga la línea de tiempo ni mueve el cursor. Hay tres formas de anclarlos:

- `scene.media.sfx(...)`: en un segundo absoluto.
- `anim.sound(...)`: al inicio de una animación. Si la animación se mueve (un `delay`, su lugar en una `sequence` o un `stagger`), el sonido se mueve con ella.
- `Transition.*(..., sound=...)`: al inicio de la transición, que es el inicio del segmento al que entra.

43.1.5. MediaLibrary.sfx

MÉTODO `sfx(path: str, at: Optional[float] = None, *, volume: float = 1.0) -> None`

Coloca un efecto que suena una vez, entero. Un archivo inexistente o un `at` o `volume` negativos o no finitos lanzan `ValueError`. Para que el sonido siga a una animación que puede moverse, usa `Anim.sound`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`path`	`str`	obligatorio	Archivo de audio; una ruta relativa usa la carpeta de assets.
`at`	`float | None`	`None`	Segundo absoluto de inicio; sin él, el cursor actual (`scene.cursor`).
`volume`	`float`	`1.0`	Ganancia lineal.

Devuelve: `None` — El efecto queda programado de inmediato.

Ejemplo:

```
scene.play(title.animate.fade_in())
scene.media.sfx("whoosh.wav", at=scene.cursor)
scene.media.sfx("pop.wav", at=0.25, volume=0.6)
scene.wait(1)
```

43.1.6. Anim.sound

MÉTODO `sound(path: str, *, volume: float = 1.0, offset: float = 0.0) -> Anim`

El sonido empieza cuando empieza la animación resuelta, incluidos su `delay` y su posición en composiciones, y suena una vez, entero, aunque la animación use `repeat`. Una ruta vacía, un `volume` negativo o no finito o un `offset` no finito lanzan `ValueError` al llamarlo; un archivo inexistente o un sonido que empezaría antes de 0 s lanzan `ValueError` desde `scene.play`, que entonces no programa nada.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`path`	`str`	obligatorio	Archivo de audio; una ruta relativa usa la carpeta de assets.
`volume`	`float`	`1.0`	Ganancia lineal.
`offset`	`float`	`0.0`	Segundos desde el inicio de la animación hasta el sonido; negativo lo adelanta.

Devuelve: `Anim` — Una copia con el efecto anclado; una segunda llamada lo reemplaza.

Ejemplo:

```
title = scene.text("Efectos anclados", role="title")
dot = scene.geometry.circle(0.5).move_to(0, -2)
scene.play(title.animate.write().sound("typing.wav", volume=0.6))
```

```
# El pop suena 0.3 s después, con el círculo.
scene.play(dot.animate.grow_from_center().delay(0.3).sound("pop.wav"))
```

Una transición con sonido:

```
scene.segment("detalle", Transition.slide(0.5, "left", sound="whoosh.wav"))
scene.wait(1)
```

La vista previa reproduce varias pistas a la vez y aplica `volume`, `fade_in` y `fade_out`. MP4 usa AAC y WebM usa Opus. Las secuencias de imágenes, GIF y WebP animado rechazan las pistas porque esos formatos no transportan audio.

43.1.7. Audio de los videos

El audio embebido de un video sigue las mismas reglas que una pista.

`clip = scene.media.video(...)` solo declara el video y `scene.play([clip])` activa juntos sus fotogramas y su audio; ambos se pausan, recorren y repiten con la línea de tiempo. `audio=False` silencia el video y `volume` fija su ganancia. Con `video.segment(start=..., end=...)`, cada fragmento genera una pista finita: las pausas entre fragmentos son silenciosas aunque el último fotograma siga visible, y `speed` conserva el tono. Consulta Medios.

43.1.8. Audio que anima la escena

Los datos de una pista animan cualquier cosa. `level`, `band` y `pulse` devuelven un `Computed` del tiempo, entre 0 y 1, que se usa donde se acepta un número reactivo:

- `scale_to`, `opacity`, `move_to` y `rotate_to`;
- la cámara (`bind_2d`) y los uniforms de los shaders;
- `computed`, `scene.viz.readout` y `fill_level`;
- `Updater.rotate` y `Updater.wiggle`;
- los colores, con `Falloff.source` y `drive("fill", ...)`.

El archivo se analiza una sola vez: los WAV directamente y los demás formatos con FFmpeg. Cada señal consulta ese análisis donde suena la pista, según su inicio, así que un salto en la línea de tiempo o una exportación leen lo mismo que la reproducción. Antes de que la pista suene y después de que termine, las señales valen 0.

```
musica = scene.media.audio("assets/ritmo.ogg")
bajo = musica.band(20, 150, smoothing=0.5)
golpe = musica.pulse(decay=0.12)

logo = scene.geometry.star(5, 1, 0.45).fill(GOLD)
logo.scale_to(computed(lambda b: 0.8 + 0.5 * b, inputs=[bajo]))
logo.add_updater(Updater.wiggle(position=computed(lambda p: 0.15 * p, inputs=[golpe])))
puntos = scene.geometry.group([scene.geometry.circle(0.2).move_to(x, -2) for x in range(-3, 4)])
puntos.drive("fill", Falloff.source(bajo).gradient(BLUE, GOLD))

scene.play(musica)
scene.wait(musica.duration)
scene.render()
```

`scene.play(musica)` sin `duration` no alarga el lote: `scene.wait(musica.duration)` mantiene la escena mientras suena. El ejemplo completo, `examples/audio/signals.py`, está en el repositorio de Gaanim; no viene con el paquete instalado.

43.1.9. Audio.level

MÉTODO `level(*, smoothing: float = 0.0, range_db: Optional[float] = None) -> Computed`

Envolvente de volumen (RMS), normalizada al percentil 99 del archivo para que unos pocos picos no aplasten el resto. En una pista masterizada las partes fuertes quedan todas cerca de 1; `range_db` las separa. Un `smoothing` fuera de rango o un `range_db` fuera de `(0, 120]` lanzan `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`smoothing`	`float`	`0.0`	En `[0, 1)`: cada cuadro conserva esa parte del anterior, así que cerca de 1 cambia despacio.
`range_db`	`float | None`	`None`	Mide en decibelios, como el oído: 0 a `range_db` dB por debajo de la parte fuerte y 1 en ella. Con 24, un drop se distingue de un verso más suave.

Devuelve: `Computed` — El volumen donde suena la pista: 0 en silencio, 1 en su parte fuerte.

43.1.10. Audio.band

MÉTODO `band(low: float, high: float, *, smoothing: float = 0.0) -> Computed`

El análisis llega hasta 11 kHz. Una banda con `low >= high`, negativa o por encima de ese rango lanza `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`low / high`	`float`	obligatorio	Banda de frecuencias en Hz: `band(20, 150)` sigue el bajo y `band(2000, 8000)`, los platillos y las sibilantes.
`smoothing`	`float`	`0.0`	Como en `level`.

Devuelve: `Computed` — La amplitud de la banda, de 0 a 1, cada banda respecto a su propia parte fuerte.

43.1.11. Audio.pulse

MÉTODO `pulse(decay: float = 0.15, *, low: Optional[float] = None, high: Optional[float] = None) -> Computed`

La forma más directa de reaccionar al ritmo: `scale_to(computed(lambda p: 1 + 0.2 * p, inputs=[musica.pulse()]))`. Un golpe es un **onset**: donde empieza un sonido (un bombo, una nota, una sílaba). Sin banda salta con cualquiera, y en música electrónica los platillos lo disparan

sin parar; `pulse(low=40, high=120)` sigue solo el bombo. Un `decay` no positivo, solo uno de `low`/`high` o una banda inválida lanzan `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`decay`	`float`	`0.15`	Segundos en que el pulso cae a un tercio.
`low / high`	`float | None`	`None`	Solo cuentan los golpes de esa banda, en Hz; los dos o ninguno.

Devuelve: `Computed` — 1 en cada golpe y bajando hasta 0.

43.1.12. Audio.beats

MÉTODO `beats(*, low: Optional[float] = None, high: Optional[float] = None) -> list[float]`

Son onsets, no una rejilla de tempo; para la rejilla usa `tempo`. Suma `musica.start` cuando la pista ya sonó para obtener segundos de la línea de tiempo, por ejemplo para colocar marcas o animaciones en cada golpe.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`low / high`	`float | None`	`None`	Solo los golpes de esa banda, en Hz, como en `pulse`.

Devuelve: `list[float]` — Segundos desde el inicio de la pista en que empieza un sonido, en orden.

Ejemplo:

```
scene.play(musica)
golpes = [musica.start + t for t in musica.beats()]
bombos = [musica.start + t for t in musica.beats(low=40, high=120)]
```

43.1.13. Audio.tempo

MÉTODO `tempo(*, min_bpm: float = 60.0, max_bpm: float = 200.0) -> AudioTempo`

Estima un tempo estable: el periodo en que más se repiten los golpes, dando peso a los graves (bombo y bajo, 40–150 Hz), y la fase que pone los pulsos sobre esos golpes. `confidence` mide cuánto se repiten los golpes en la rejilla de ese tempo, a uno, dos o cuatro pulsos: un bombo a medio tiempo, cada dos pulsos, también da un pulso claro. Si el rango no contiene el tempo de la música, se devuelve el mejor pico dentro del rango con una confianza baja, nunca uno de sus extremos. El tempo y el pulso que se siente pueden estar en proporción 4:3 o 2:1, así que conviene comparar con el tempo publicado del tema. Así `scene.tempo(t.bpm, offset=musica.start + t.offset)` alinea `wait_until(bar=...)` con la música. Sigue la `speed`

y el `offset` de la pista. Sirve para música con pulso estable; con `confidence` por debajo de 0.3 no hay un pulso claro. No detecta secciones (drops, estribillos): el volumen confunde la entrada del bombo con un drop, así que esas marcas se colocan a mano con `analysis`. Un rango fuera de `0 < min_bpm < max_bpm <= 400` lanza `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`min_bpm / max_bpm`	`float`	`60 / 200`	Rango en que se busca el tempo.

Devuelve: `AudioTempo` — `bpm`, `offset` (segundos desde el inicio de la pista hasta su primer pulso, menos de un pulso) y `confidence`, de 0 a 1.

Ejemplo:

```
scene.play(musica)
pulso = musica.tempo()
scene.tempo(pulso.bpm, offset=musica.start + pulso.offset)
scene.wait_until(bar=2)
```

43.1.14. Audio.analysis

MÉTODO `analysis() -> AudioData`

Para colocar cortes a mano con datos reales. `frame_rate` cuadros por segundo (unos 43) y `times`, el segundo del archivo de cada cuadro; `level`, el volumen de 0 a 1, y `level_db`, en decibelios por debajo de la parte fuerte (0 en ella, −120 en silencio); `spectrum`, una fila por cuadro con la amplitud de 96 bandas logarítmicas (`frequencies`, en Hz), de 0 a 1 contra la parte fuerte de todo el archivo; `band(low, high)`, la amplitud de una banda por cuadro, y `onsets(low=, high=)`. Los tiempos son del archivo, antes del `offset` y la `speed` de la pista. Son listas de Python, listas para numpy si lo usas.

Devuelve: `AudioData` — El análisis del archivo como números.

Ejemplo:

```
datos = musica.analysis()
bombo = datos.band(40, 120)
fuertes = [t for t, v in zip(datos.times, bombo) if v > 0.9]
```

43.1.15. Audio.start / duration

PROPIEDAD

`start` es el segundo de la línea de tiempo en que la pista empezó a sonar por última vez, o `None` antes de reproducirla. `duration` son los segundos que suena: su `duration`, o el archivo entero.

43.1.16. Falloff.source

FÁBRICA `source(value: Computed) -> Falloff`

Lleva las señales de audio a `drive`, que es como cambian los colores: `drive("fill", Falloff.source(bajo).gradient(BLUE, GOLD))`. Un `Computed` que lee un `Parameter` lanza `ValueError`. Ver Animaciones.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`value`	`Computed`	obligatorio	Un `Computed` que depende solo del tiempo: una señal de audio, `scene.noise` o `scene.time`.

Devuelve: `Falloff` — Su valor, el mismo para todos los miembros.

43.1.17. Visualizar el audio

Para dibujar el sonido, una pista da una señal por banda de frecuencia (`spectrum`) o por instante de su pasado reciente (`waveform`), y `scene.viz.equalizer` las dibuja. Cada parte se elige o se reemplaza: la forma de los elementos, su disposición y sus colores. Como son señales normales, también sirven para dibujar a mano o en un shader.

43.1.18. Audio.spectrum

MÉTODO `spectrum(bands: int = 32, *, low: float = 40.0, high: Optional[float] = None, smoothing: float = 0.0, normalize: Literal["band", "global"] = "band") -> list[Computed]`

Reparte el rango en escala logarítmica, como el oído percibe el tono. Con `"band"`, igual que `band`, los agudos también se mueven aunque suenen menos; con `"global"` las barras conservan la forma real del espectro (graves altos, agudos bajos). Un rango inválido u otro `normalize` lanzan `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`bands`	`int`	`32`	Bandas, de 1 a 256.
`low / high`	`float`	`40 / None`	Rango en Hz; `high` es por defecto la frecuencia más alta del análisis, unos 11 kHz.
`smoothing`	`float`	`0.0`	Suavizado de cada banda, en `[0, 1)`.
`normalize`	`str`	`"band"`	`"band"` mide cada banda contra su propio máximo; `"global"`, todas contra la más fuerte.

Devuelve: `list[Computed]` — Una señal de 0 a 1 por banda, de la más grave a la más aguda.

43.1.19. Audio.waveform

MÉTODO `waveform(points: int = 64, *, span: float = 2.0, low: Optional[float] = None, high: Optional[float] = None, smoothing: float = 0.0) -> list[Computed]`

El punto `i` lee la pista `span * (1 - i / (points - 1))` segundos atrás, así que los valores avanzan hacia el primer punto mientras suena, como el trazo de una grabadora. Valores fuera de rango lanzan `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`points`	`int`	`64`	Instantes, de 2 a 512.
`span`	`float`	`2.0`	Segundos que abarca, hacia atrás desde el presente.
`low / high`	`float | None`	`None`	Sigue esa banda en lugar del volumen; los dos o ninguno.
`smoothing`	`float`	`0.0`	Suavizado, en `[0, 1)`.

Devuelve: `list[Computed]` — El volumen en `points` instantes de los últimos `span` segundos, el más antiguo primero.

43.1.20. Audio.samples

MÉTODO `samples(points: int = 256, *, span: float = 0.05) -> list[Computed]`

La onda misma, con signo, contra el pico del archivo: un osciloscopio. `waveform` sigue el volumen, de 0 a 1; `samples` dibuja la forma de las notas con un `span` de unas centésimas de segundo. Valores fuera de rango lanzan `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`points`	`int`	`256`	Instantes, de 2 a 1024.
`span`	`float`	`0.05`	Segundos que abarca, hacia atrás desde el presente.

Devuelve: `list[Computed]` — La onda del sonido en esos instantes, de −1 a 1, la más antigua primero.

Ejemplo:

```
onda = musica.samples(200, span=0.03)
osciloscopio = scene.geometry.polyline(
    [(-6 + 12 * i / 199, computed(lambda v: 1.5 * v, inputs=[s])) for i, s in enumerate(onda)]
).stroke(GOLD, 0.03)
scene.play(musica)
scene.wait(3)
scene.render()
```

43.1.21. Visualization.equalizer

FÁBRICA `equalizer(values: Sequence[float | Parameter | Variable | Computed], *, shape: Literal["bar", "capsule", "dot", "line", "area"] | Callable[[Scene, int, int], Drawable] = "bar", layout: Literal["row", "mirror", "radial"] |`

```
Callable[[int, int], tuple[float, float, float]] = "row", center: tuple[float, float] = (0.0, 0.0), width: float = 8.0, height: float = 2.0, radius: float = 1.5, gap: float = 0.25, thickness: Optional[float] = None, min_length: float = 0.04, start_angle: float = 1.5707963267948966, fill: Optional[Paint | Sequence[Paint] | FalloffColor | Callable[[int, int], Paint]] = None, value_colors: Optional[Sequence[ColorLike]] = None, stretch: bool = True) -> Drawable
```

Una lista vacía, una forma o disposición desconocida o tamaños inválidos lanzan `ValueError`; una función que no devuelve lo esperado, `TypeError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`values`	`Sequence[ScalarSource]`	obligatorio	Señales de 0 a 1: `spectrum()`, `waveform()`, `Parameter`, `Computed` o números. Lo que sale de ese rango se recorta.
`shape`	`str | Callable`	`"bar"`	`"bar"`, `"capsule"` (extremos redondos), `"dot"` (en la punta), `"line"` (una curva por las puntas) o

			`"area"` (esa curva rellena hasta la base). O una función `shape(scene, index, count)` que devuelve cualquier objeto, que se centra en su lugar, se gira a lo largo y se estira a su longitud.
`layout`	`str | Callable`	`"row"`	`"row"` (crece hacia arriba desde la base de una caja de `width` ×

`height`), `"mirror"` (crece hacia los dos lados de una línea media) o `"radial"` (alrededor de un círculo de `radius`, desde `start_angle` en sentido horario, hacia fuera). O una función `layout(index, count)` que devuelve `(x, y, ángulo)`: dónde empieza cada elemento y en qué dirección crece, en radianes.

`fill`	`Paint | list | FalloffColor | Callable`	`None`	Una pintura, una lista que se repite, una rampa `Falloff` sobre el grupo (`Falloff.index(` o `fill(index, count)`.
`value_colors`	`Sequence[ColorLike] | None`	`None`	Colorea cada elemento según su propio valor a lo largo de estos colores. Requiere señales que solo dependen del tiempo, como las de audio.
`center / width / height / radius`	—	`(0, 0) / 8 / 2 / 1.5`	Dónde y de qué tamaño.

			Cada elemento mide de `min_length` (0.04) a `height`.
`gap / thickness`	`float`	`0.25 / None`	Fracción de cada hueco que queda vacía, o un grosor fijo.
`stretch`	`bool`	`True`	Con una forma propia, `False` la escala entera en lugar de estirarla, para iconos que deben conservar su proporción.

Devuelve: `Drawable` — Un grupo con un miembro por valor, en orden: objetos normales que aceptan cualquier estilo, efecto o animación.

Ejemplo:

```
import math
from gaanim import BLUE, CORAL, GOLD, Falloff, Scene
scene = Scene(frame=(16, 9), background="#0b1020")
```

```
music = scene.media.audio("assets/ritmo.ogg")
bars = scene.viz.equalizer(music.spectrum(32, smoothing=0.3), center=(-4, 1.5), width=6.5,
                           fill=Falloff.index().gradient(BLUE, GOLD))
ring = scene.viz.equalizer(music.spectrum(40), shape="dot", layout="radial", center=(4, 1.5),
                           radius=1.2, height=0.8, fill=[CORAL, GOLD])
trace = scene.viz.equalizer(music.waveform(96, span=2), shape="line", layout="mirror",
                            center=(0, -2.2), width=12, height=1.6)
scene.play(music)
scene.wait(music.duration)
scene.render()
```

Las mismas señales llegan a un shader como uniforms de `PostProcess.shader`. Un shader no indexa campos, así que el ejemplo genera una función que elige la banda:

```
from gaanim import PostProcess, Scene
scene = Scene(frame=(16, 9), background="#0b1020")
music = scene.media.audio("assets/ritmo.ogg")
bands = music.spectrum(16, smoothing=0.3)
pick = "\n".join(f"    if (i == {i}u) {{ return gaanim_uniforms.b{i}; }}" for i in range(16))
scene.canvas.post = PostProcess.shader(f"""
fn band(i: u32) -> f32 {{
{pick}
    return 0.0;
}}
fn gaanim_post(uv: vec2<f32>, resolution: vec2<f32>, time: f32) -> vec4<f32> {{
    let column = min(u32(uv.x * 16.0), 15u);
    if (1.0 - uv.y < 0.3 * band(column)) {{
        return vec4<f32>(0.2, 0.8, 0.95, 1.0);
    }}
    return gaanim_scene(uv);
}}
""", uniforms={f"b{i}": band for i, band in enumerate(bands)})
scene.play(music)
scene.wait(music.duration)
scene.render()
```

Para más detalle, `audio=` le da al shader el espectrograma entero de la pista sin gastar uniforms (usa uno, `gaanim_audio_time`, reservado). El shader llama a:

- `gaanim_audio_spectrum(x, ago)`: el espectro en `x`, de 0 (20 Hz) a 1 (unos 11 kHz) en escala logarítmica, cada frecuencia de 0 a 1 contra su propio máximo, tal como sonaba hace `ago` segundos;
- `gaanim_audio_level(ago)`: el volumen;
- `gaanim_audio_band(low, high, ago)`: la media entre dos frecuencias en Hz;
- `gaanim_audio_x(hz)`: dónde cae una frecuencia en la escala de `x`.

Leer varios `ago` dibuja la historia del espectro: un espectrograma que avanza mientras suena. Todas valen 0 mientras la pista no suena. Los paquetes `.gaanim` guardan los datos una sola vez. También sirve en `shader_effect`.

```
from gaanim import PostProcess, Scene
scene = Scene(frame=(16, 9), background="#0b1020")
music = scene.media.audio("assets/ritmo.ogg")
scene.canvas.post = PostProcess.shader("""
fn gaanim_post(uv: vec2<f32>, resolution: vec2<f32>, time: f32) -> vec4<f32> {
    // Izquierda: hace 3 s; derecha: ahora. Abajo los graves.
    let energy = gaanim_audio_spectrum(1.0 - uv.y, 3.0 * (1.0 - uv.x));
    let heat = vec3<f32>(energy, energy * energy, 0.3 + 0.7 * energy * energy * energy);
    return vec4<f32>(mix(gaanim_scene(uv).rgb, heat, energy), 1.0);
```

```
}
""", audio=music)
scene.play(music)
scene.wait(music.duration)
scene.render()
```

43.1.22. Narración

Gaanim graba tu voz sin salir del editor y la sincroniza con la escena, sin editor de video externo. Hay dos formas de trabajar:

- **Voz en off**: la voz marca el ritmo. Cada bloque `scene.voiceover(...)` tiene su propia toma y el script espera a marcas con nombre en lugar de llevar duraciones fijas. Puedes regrabar un bloque sin tocar los demás.
- **Toma en vivo**: presentas la escena hablando. Cada `scene.stop()` posterior a `scene.live_take(...)` se convierte en una pausa tan larga como la que hiciste al grabar.

Las tomas viven en `narration/` dentro de la carpeta de assets: la de `scene.assets.load_project(...)` o, sin esa llamada, la del proyecto al que pertenece el script (el `gaanim.toml` más cercano hacia arriba). Fuera de un proyecto, o si un proyecto anterior ya tiene `narration/` junto al script y no en sus assets, van junto al script: `narration/<clave>.wav` y, al lado, `narration/<clave>.json` con las marcas. Al grabar o editar esos archivos, el editor recarga la escena con los tiempos nuevos. `render()` y la exportación alargan el timeline para que ninguna toma quede cortada, y MP4/WebM mezclan las tomas como cualquier otra pista.

43.1.22.1. El guion

El texto que lees no tiene que vivir en el código. Escríbelo en `narration/script.md` con cualquier editor: cada encabezado `## clave` contiene el texto de `scene.voiceover("clave")`, y un encabezado con el nombre de una diapositiva, sus notas del orador (ver El guion de la charla). Gaanim lo carga automáticamente; para usar otro archivo, llama antes a `scene.narration_script("guion.md")`. Al guardar el guion, el editor recarga la escena con las duraciones estimadas nuevas.

```
# Mi video sobre derivadas

## intro
Hoy vemos qué es una derivada: la pendiente de una curva en un punto.

### nota para mí: sonreír, sin prisa

## tangente
Movemos el punto por la recta y leemos su pendiente, que no cambia.
<!-- aquí respiro antes del cierre -->
```

Un `# título` cierra el bloque anterior; los encabezados `###` son notas para ti que no se leen, los comentarios HTML se ignoran y una línea `[stop]` o `[stop nombre]` marca dónde avanzas, y la voz no la lee. Las líneas seguidas se unen y una línea en blanco separa párrafos. Un `##` vacío o repetido (sin distinguir mayúsculas) produce `ValueError`. El texto de un bloque se busca en este orden: el argumento `text`, su sección del guion y las notas del segmento; con un guion cargado, una clave sin sección emite un `UserWarning`. El icono del guion en la cabecera del panel crea el archivo o le añade las secciones de los bloques y las diapositivas que faltan, y las marcas de sus stops, y lo abre en el editor de `GAANIM_EDITOR` en lo primero que añadió. En una toma en vivo, el subtítulo del editor muestra el tramo del guion que toca decir, con ▸ y el nombre del stop donde toca avanzar (ver El prompter).

43.1.22.2. Grabar desde el editor

Abre el panel con el botón de micrófono de la barra de reproducción (o **More controls → Script & narration**). El panel se acopla a la derecha (ver El panel Script & narration); al final lista la toma en

vivo y cada bloque de voz en off con su estado: **Not recorded** (duración estimada del texto) o **Recorded**, su nivel y de dónde sale su texto. Mientras grabas, un subtítulo abajo de la vista previa muestra el texto que toca decir y, debajo, el reloj, el medidor de nivel y las teclas.

- **Record** posiciona la escena al inicio del bloque y abre el micrófono predeterminado durante una cuenta atrás de tres segundos que sirve de prueba de nivel (no se guarda). Después graba mientras la escena avanza, con el texto del bloque en el subtítulo y la siguiente marca resaltada. Pulsa **Espacio** al decir cada marca, **Enter** para guardar y **Esc** para descartar.
- **Record live take** recarga la escena con sus paradas originales y graba mientras presentas. La reproducción se detiene en cada parada; → o **Espacio** continúa, **Enter** termina y **Esc** descarta. Con **Also dictate the script with Scribe**, Scribe de Zoom transcribe a la vez lo que dices y lo escribe en el guion (ver Dictar el guion con Scribe).
- **Listen** reproduce el bloque desde su inicio.
- **Level** lleva una toma ya grabada al nivel configurado.

Durante la grabación el audio del preview se silencia para que la toma anterior no se cuele en el micrófono.

43.1.22.3. Nivel de la voz

Al grabar, el medidor del subtítulo muestra los picos del micrófono en dBFS. La zona verde, entre −18 y −6 dBFS, es la ideal. Entre −6 y −3 dBFS el medidor pasa a ámbar, y por encima de −3 dBFS el borde del subtítulo se pone rojo con el aviso «Too loud!»: una palabra más fuerte podría saturar, y la saturación no se corrige después. Si algún tramo satura, el aviso lo cuenta hasta el final de la toma; si durante cuatro segundos nada llega a −30 dBFS, avisa de que la voz está muy baja.

Cada toma nueva se nivela al guardarla a −16 LUFS integrados, el nivel recomendado para voz en video online (YouTube reproduce a −14 y la televisión usa −23). FFmpeg mide la toma y le aplica una ganancia constante, con un pico real máximo de −1.5 dBTP y un filtro paso alto a 80 Hz. La grabación original se conserva en `narration/.originals/`. En **Settings → Recording** (`Ctrl+,`) puedes cambiar el nivel final, y avisa si eliges más de −14 LUFS o menos de −20, o desactivar la nivelación. Las tomas sin nivelar aparecen marcadas en el panel.

43.1.22.4. Whisper (opcional)

En lugar de pulsar Espacio, whisper.cpp puede detectar cuándo dices cada palabra. Instala `whisper-cli` y un modelo `ggml-*.bin`, e indícalos en **Settings → Transcription** (vacío busca `whisper-cli` en el `PATH`). **Transcribe** guarda en el JSON de la toma el tiempo de cada palabra; también puede hacerse automáticamente al guardar cada toma. Requiere FFmpeg. Para que una marca se resuelva por transcripción, su nombre debe ser la palabra o frase que dices; se ignoran mayúsculas, tildes y puntuación.

43.1.22.5. Cómo se resuelven las marcas

Cada marca se busca, en este orden: la marca pulsada al grabar (o escrita a mano en el JSON), la transcripción de Whisper y, por último, su posición proporcional dentro del texto del bloque. Las marcas se buscan después de la anterior, así que una palabra repetida corresponde a su siguiente aparición; pedir otra vez el mismo nombre devuelve el mismo tiempo. Una marca que no se encuentra emite un `UserWarning` y no espera.

Sin toma grabada, la duración se estima del texto a unas 150 palabras por minuto, de modo que puedes componer y revisar la escena antes de grabar.

El JSON es editable: mover un número retima la escena sin tocar el código.

```
{
  "version": 1,
  "markers": { "derivada": 2.4, "pendiente": 5.1 },
  "words": [{ "text": "Hoy", "start": 0.12, "end": 0.3 }],
  "holds": [],
```

```
  "loudness": -16.0
}
```

`markers` guarda segundos desde el inicio de la toma, `words` la transcripción, `holds` los segundos de cada pausa de una toma en vivo y `loudness` el nivel en LUFS al que se niveló.

43.1.22.6. API de narración

Los bloques de voz en off, sus marcas y la toma en vivo.

43.1.23. Scene.voiceover

MÉTODO `voiceover(key: str, *, text: Optional[str] = None, volume: float = 1.0) -> Voiceover`

Empieza un bloque en el cursor y devuelve un `Voiceover`, usable con `with`. Una toma grabada (`.wav`, `.flac`, `.mp3`, `.m4a`, `.aac`, `.ogg` u `.opus`) suena desde aquí y fija la duración del bloque; si no existe, la duración se estima del texto. Al salir del `with` se espera el resto de la toma. Claves inválidas o repetidas, volumen negativo o tomas ilegibles lanzan `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`key`	`str`	obligatorio	Nombre de la toma: letras, dígitos, `-`, `_` o `.`. Es el nombre de archivo dentro de `narration/`.
`text`	`str | None`	`None`	Texto del bloque: teleprompter y estimación de tiempos. Sin él se usa la sección `## key` del guion y, si no existe, las `notes` del segmento activo.
`volume`	`float`	`1.0`	Ganancia lineal de la toma.

Ejemplo:

```python
from gaanim import Scene

scene = Scene()
scene.segment("derivada", notes="Hoy vemos la derivada y su pendiente")
titulo = scene.text("La derivada", role="title").move_to(0, 2.5)
curva = scene.geometry.circle(1.5)
with scene.voiceover("intro") as vo:
    scene.play([titulo.animate.write()])
    vo.wait_until("derivada")
    scene.play([curva.animate.create()], duration=vo.until("pendiente"))
print(f"grabada: {vo.recorded} · {vo.duration:.1f} s · termina en {vo.end:.1f} s")
scene.render()
```

```
grabada: False · 3.2 s · termina en 3.2 s
```

43.1.24. Voiceover.wait_until

MÉTODO `wait_until(marker: str) -> None`

Avanza el cursor hasta la marca. Si la marca ya quedó atrás o no existe, no espera. Usarlo en un bloque cerrado lanza `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`marker`	`str`	obligatorio	Nombre de la marca: palabra o frase del guion, o el nombre pulsado al grabar.

43.1.25. Voiceover.until

MÉTODO `until(marker: str) -> float`

Segundos desde el cursor hasta la marca, nunca negativos. Úsalo como `duration` para que una animación termine justo en la marca.

Ejemplo:

```
vo = scene.voiceover("cierre", text="Y eso es todo por hoy")
scene.play([logo.animate.fade_in()], duration=vo.until("todo"))
vo.finish()
```

43.1.26. Voiceover.finish

MÉTODO `finish() -> None`

Espera el resto de la toma y cierra el bloque; repetirlo no hace nada. Es lo que hace el `with` al salir.

43.1.27. Voiceover.key / text / start / duration / end / remaining / recorded

PROPIEDAD `key: str · text: str | None · start: float · duration: float · end: float · remaining: float · recorded: bool`

Nombre de la toma, texto del bloque, segundos absolutos de inicio y fin, duración (medida o estimada), segundos que faltan desde el cursor y si hay un archivo grabado (`False` mientras la duración sea estimada).

43.1.28. Scene.narration_script

MÉTODO `narration_script(path) -> None`

Sin esta llamada se carga `narration/script.md` si existe. Las secciones con el nombre de una diapositiva son sus notas del orador y sustituyen a `notes=`. Llámala antes de los bloques que la usan; las diapositivas la leen la llames donde la llames. Un archivo ilegible o un `##` vacío o repetido producen `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`path`	`str`	obligatorio	Archivo Markdown; una ruta relativa se busca en la carpeta de assets, o junto al script si la escena no tiene assets.

Devuelve: `None` — Los bloques que empiezan después y todas las diapositivas leen su texto de este guion.

Ejemplo:

```
scene.narration_script("guion.md")
with scene.voiceover("intro") as vo:
    vo.wait_until("derivada")
```

43.1.29. Scene.live_take

MÉTODO `live_take(key="live", *, volume=1.0) -> None`

Empieza la toma en el cursor. Con la toma grabada, suena desde aquí y cada `stop()` posterior espera lo que duró tu pausa, así que el timeline se reproduce de corrido en sincronía con tu voz. Sin toma, las paradas siguen siendo interactivas. Una segunda toma en vivo, una clave inválida o usada por una voz en off, o una toma ilegible producen `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`key`	`str`	`"live"`	Nombre de la toma dentro de `narration/`.
`volume`	`float`	`1.0`	Ganancia lineal de la toma.

Devuelve: `None` — Declara la única toma en vivo de la escena.

Ejemplo:

```
scene.live_take("clase")
scene.segment("intro", notes="Presentamos el problema")
scene.play([titulo.animate.write()])
scene.stop()
scene.segment("idea", notes="La idea clave es…")
scene.play([diagrama.animate.fade_in()])
scene.stop()
scene.render()
```

CAPÍTULO

44. Recursos

Carpeta de recursos, precarga y recarga, y qué admiten los importadores de SVG y Lottie

Imágenes, SVG, vídeo, audio, Lottie, WGSL y archivos Typst se cargan desde rutas locales. Las fábricas que los crean están en Medios y Audio esta página explica cómo se resuelven las rutas, cómo precargar y recargar archivos, y qué partes de cada formato se importan.

```
from gaanim import Scene

scene = Scene(frame=(16, 9), background="#0f172a")
scene.assets.load_project()          # usa assets_dir del gaanim.toml del proyecto

logo = scene.media.svg("logo.svg")
cover = scene.media.image("cover.png")
pulse = scene.media.lottie("pulse.json")
```

Una ruta relativa se resuelve desde la carpeta de recursos de la escena. Sin ella, se busca en el directorio de trabajo del proceso, después en la carpeta `assets/` junto al script que creó la escena y por último en la carpeta del script, así que `scene.media.svg("logo.svg")` encuentra `assets/logo.svg` sin configurar nada. Una ruta absoluta se usa tal cual. Todas las fábricas usan archivos locales: no descargan URLs ni aceptan bytes o arrays.

44.1.1. Carpeta de recursos

44.1.2. AssetManager.load_project

MÉTODO `load_project(path: str | None = None) -> None`

Lee la clave `assets_dir` del manifiesto (`"assets"` si falta, como en la CLI) y la resuelve respecto de la carpeta del manifiesto, así que funciona igual se lance Gaanim desde donde se lance. Una ruta explícita relativa, en cambio, se resuelve desde el directorio de trabajo del proceso; sin argumento se busca el `gaanim.toml` en la carpeta del script que llama y después en sus carpetas superiores, así que un módulo como `capitulo4/estilo.py` usa el manifiesto del proyecto. Si no hay ninguno, lanza `RuntimeError` con la carpeta de partida. Es la forma recomendada en un proyecto. Un manifiesto ilegible lanza `RuntimeError`; un `assets_dir` que no es una cadena entre comillas o que apunta a una carpeta inexistente, `ValueError`. Consulta `gaanim.toml`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`path`	`str | None`	`None`	Ruta del manifiesto; sin ella, el `gaanim.toml` más cercano, buscando desde la carpeta del script que llama hacia arriba.

Devuelve: `None` — Fija la carpeta de recursos.

Ejemplo:

```python
scene = Scene(frame=(16, 9), background="#0f172a")
scene.assets.load_project()
scene.assets.load_project("gaanim.toml")   # ruta explícita
```

44.1.3. AssetManager.assets_dir

MÉTODO `assets_dir(path: str) -> None`

Una ruta relativa se resuelve desde el directorio de trabajo del proceso, no desde el script. Para que el script no dependa de dónde se lanza, usa `load_project()` o construye la ruta desde `__file__`. Una carpeta inexistente lanza `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`path`	`str`	obligatorio	Carpeta existente.

Devuelve: `None` — Fija la carpeta de recursos.

Ejemplo:

```python
from pathlib import Path

scene = Scene(frame=(16, 9), background="#0f172a")
scene.assets.assets_dir(str(Path(__file__).parent / "assets"))
hero = scene.media.image("cover.png", width=8)
```

44.1.4. AssetManager.preload

MÉTODO `preload(paths: Sequence[str]) -> None`

Comprueba archivos ráster, SVG, Lottie JSON y `.lottie` antes de reproducir la escena. Las imágenes se decodifican en la misma caché que usa `scene.media.image`, y las composiciones Lottie en la de `scene.media.lottie`; de un paquete dotLottie se precarga su animación o máquina de estados inicial. Un archivo que no se puede resolver o decodificar lanza `RuntimeError` con su ruta.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`paths`	`Sequence[str]`	obligatorio	Rutas que se validan, relativas a la carpeta de recursos.

Devuelve: `None` — Valida y guarda en caché los archivos.

Ejemplo:

```python
scene.assets.preload(["logo.svg", "cover.png", "pulse.json", "button.lottie"])
```

44.1.5. AssetManager.reload_assets

MÉTODO `reload_assets() -> None`

Vacía las cachés de imágenes ráster, Lottie JSON y paquetes dotLottie. Los objetos ya creados conservan sus recursos; las cargas siguientes leen los archivos del disco. En el editor no hace falta: al guardar un archivo del proyecto, el editor olvida lo que las cachés guardan de ese archivo y recarga la escena, recompilando desde el primer segmento que lo usa. Los SVG se vuelven a leer cada vez que `scene.media.svg(...)` crea un objeto.

Devuelve: `None` — Vacía las cachés de archivos.

Ejemplo:

```
scene.assets.reload_assets()
cover = scene.media.image("cover.png")
```

Para crear un proyecto con esta estructura, usa `gaanim init`.

44.1.6. Lottie JSON y dotLottie

`scene.media.lottie` acepta archivos Lottie JSON y paquetes `.lottie` v1 y v2, y los dibuja como vectores con Velato dentro de la misma escena. La reproducción sigue el tiempo absoluto de la escena, también al saltar hacia atrás. Las opciones de la fábrica y los métodos `set_theme`, `set_input`, `fire_event`, `warnings` y los identificadores del paquete están en Medios.

44.1.7. Lottie.source_width

PROPIEDAD `source_width: int`

Devuelve: `int` — Ancho de la composición original, en sus unidades.

44.1.8. Lottie.source_height

PROPIEDAD `source_height: int`

Devuelve: `int` — Alto de la composición original.

44.1.9. Lottie.frame_rate

PROPIEDAD `frame_rate: float`

Devuelve: `float` — Fotogramas por segundo de la composición.

44.1.10. Lottie.source_duration

PROPIEDAD `source_duration: float`

Devuelve: `float` — Duración de la composición, en segundos.

Ejemplo:

```python
clip = scene.media.lottie("assets/pulse.json")
print(clip.source_width, clip.source_height, clip.frame_rate, clip.source_duration)

200 200 30.0 2.0
```

Qué se importa. Formas vectoriales, transformaciones, rellenos, trazos y máscaras. Las capas sólidas se dibujan con su color y tamaño, también dentro de precomposiciones. Las imágenes externas (como las que exporta Cavalry) se resuelven respecto del archivo JSON, incluida su carpeta `images/`, y conservan el orden de capa, las transformaciones, la opacidad, el tiempo local de la precomposición y las máscaras simples. Los gradientes lineales y radiales conservan sus paradas de color y de opacidad, estáticas o animadas. Una transformación que omite posición o escala usa traslación cero y escala del 100 %, y las posiciones X/Y separadas se reproducen. El texto, las imágenes incrustadas y los efectos pueden omitirse o aproximarse: consulta `clip.warnings`. Si Velato encuentra algo que no puede convertir con seguridad, la carga lanza un error en lugar de cerrar el proceso.

Paquetes `.lottie`. Se leen en memoria, incluidas sus imágenes PNG, JPEG y WebP; no se extraen al disco ni se buscan recursos fuera del ZIP. La caché comparte recursos entre clips, pero cada clip tiene sus propias entradas y órdenes. `animation_id` y `state_machine_id` son excluyentes; sin ninguno se usa el contenido inicial del manifiesto o, si no hay, la primera animación. Los controles de reproducción del manifiesto v1 no se aplican: se usan los de Gaanim.

Máquinas de estados. Activar una máquina no alarga la escena: usa `scene.wait` u otras animaciones. El estado controla velocidad, dirección, bucles, segmento por marcador y fondo, así que `offset`, `duration`, `loop` y `speed` deben quedarse en sus valores predeterminados. Un estado final detiene las transiciones y un marcador ausente usa la animación completa. Se admiten `PlaybackState`, `GlobalState`, transiciones inmediatas y condiciones numéricas, booleanas, de cadena y de evento; los temas admiten valores estáticos de color, escalar, posición, vector y gradiente. No se admiten acciones, interacciones de ratón, entradas automáticas reservadas, transiciones suavizadas, temas con keyframes o expresiones, ni sustituciones temáticas de imágenes o texto: seleccionar un tema o una máquina que los use lanza `ValueError`, y el resto del paquete sigue disponible. Los ciclos de transiciones instantáneas también se rechazan.

```python
clip = scene.media.lottie("assets/button.lottie", state_machine_id="main", width=4)
clip.set_input("active", False)
scene.play([clip])
scene.wait(1)
clip.set_input("active", True).set_theme("gold")
scene.wait(1)
clip.fire_event("reset").set_theme(None)
scene.wait(1)
```

`write` y `create` presentan el primer fotograma trazando los contornos en paralelo y fundiendo después los rellenos y las imágenes; no inician la animación interna. Encadena el clip después para reproducirlo:

```python
from gaanim import Scene, sequence

scene = Scene(frame=(16, 9), background="#0f172a")
clip = scene.media.lottie("assets/pulse.json")
scene.play(sequence(clip.animate.create().duration(1.0), clip))
scene.render()
```

44.1.11. SVG

`scene.media.svg(path)` conserva el documento como geometría vectorial: una jerarquía real de caminos y grupos a la que se llega con `part(id)`. Por eso un SVG sirve también como máscara o como operando de operaciones booleanas. Se importa a 100 píxeles del documento por unidad lógica, centrado en el origen: un SVG de 300 px de ancho mide 3 unidades, y `scale_to(factor)` lo agranda o lo reduce (consulta `MediaLibrary.svg`).

Qué se importa. Grupos anidados, CSS, transformaciones, `viewBox` y `<use>`; rellenos y trazos sólidos, lineales y radiales con sus modos de extensión; geometría `clipPath`, sin rasterizar el documento; texto convertido en contornos con las fuentes instaladas; y los filtros `feGaussianBlur` y `feDropShadow`, como efectos vectoriales de Gaanim.

Qué no. Patrones de pintura, máscaras `<mask>` de alfa o luminancia, imágenes ráster incrustadas y grafos de filtros arbitrarios. Para una máscara, usa una silueta vectorial explícita. Para que el texto salga igual en cualquier equipo, instala la fuente en cada máquina que renderiza o convierte el texto en caminos en el SVG de origen.

Los identificadores de las partes distinguen mayúsculas; un identificador repetido en el archivo falla al importar y uno desconocido lanza `KeyError` con los nombres disponibles. El grosor aplicado al objeto raíz con `stroke(...)` está en unidades lógicas de escena y no cambia al escalar el SVG con `scale_to`; los trazos propios del archivo sí escalan con él.

```
diagram = scene.media.svg("assets/architecture.svg")
scene.play(diagram.part("database").animate.indicate().duration(0.6))
```

CAPÍTULO

45. Diapositivas

Componentes editoriales de scene.slides: insignias, tarjetas, rótulos, listas, tablas e identidad de presentación

`scene.slides` reúne componentes editoriales listos para presentaciones y videos explicativos: insignias, tarjetas, rótulos, listas, tablas y la identidad visual de cada segmento. Cada componente se mide a partir de su texto y devuelve un grupo `Drawable` que se coloca y anima como cualquier objeto.

```
from gaanim import Scene

scene = Scene(frame=(16, 9), theme="presentation")
heading = scene.slides.section_header("Resultados", kicker="03").move_to(0, 2.6)
metric = scene.slides.stat_card("98%", "Precisión", delta="+4.2%", variant="success").move_to(-2.2,
-0.4)
note = scene.slides.card("Conclusión", "El solver converge en 12 ms.", variant="accent").move_to(2.4,
-0.4)
scene.play([heading.animate.fade_in(), metric.animate.grow_from_center(), note.animate.fade_in()],
duration=1.0)
scene.render()
```

Los componentes comparten opciones de estilo:

- `variant`: tono semántico del tema (`"neutral"`, `"accent"`, `"success"`, `"warning"` o `"danger"`).
- `appearance`: `"soft"` (fondo suave), `"solid"` (fondo lleno) u `"outline"` (solo borde).
- `color`, `background` y `border`: colores explícitos que sustituyen los del tema.
- Las medidas (`width`, `padding`, `gap`, `radius`…) están en unidades de escena.

Texto vacío, variantes desconocidas o medidas inválidas lanzan `ValueError`.

45.1.1. Insignias y chips

Etiquetas cortas de estado, categoría o metadatos que se ajustan a su texto.

45.1.2. SlideKit.badge

FÁBRICA `badge(text: str, *, variant: Literal["neutral", "accent", "success", "warning", "danger"] = "neutral", appearance: Literal["soft", "solid", "outline"] = "soft", padding: tuple[float, float] = (0.18, 0.10), radius: Optional[float] = None, font_size: Optional[float] = None, min_width: Optional[float] = None, color: Optional[ColorLike] = None, background: Optional[ColorLike] = None, border: Optional[ColorLike] = None, font: Optional[str] = None, weight: Optional[int] = None, style: Optional[TextStyle] = None, markup: Optional[bool] = None) -> Drawable`

Insignia en el origen cuyo panel se mide con el mismo texto que se dibuja. Colócala con `.move_to(...)`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`text`	`str`	obligatorio	Etiqueta no vacía.
`radius`	`float | None`	`None`	`None` da forma de píldora a partir de la altura medida.
`min_width`	`float | None`	`None`	Ancho mínimo del panel.
`font / weight`	`str | None / int | None`	`None`	Familia y peso de la etiqueta; sustituyen a `style`.
`style`	`TextStyle | None`	`None`	Tipografía sobre el rol `label` del tema; su color, si lo tiene, sustituye el de la variante.
`markup`	`bool | None`	`None`	`False` deja * y _ literales, como en `scene.text`; `None` sigue `text_markup` del tema.

Ejemplo:

```
tag = scene.slides.badge("LISTO", variant="success", appearance="solid").move_to(-2.4, 1.2)
code = scene.slides.badge("_vel_max", font="DejaVu Sans Mono", weight=600, markup=False)
scene.play([tag.animate.grow_from_center()])
```

45.1.3. SlideKit.chip

FÁBRICA `chip(text: str, *, dot: bool = True, variant: Literal["neutral", "accent", "success", "warning", "danger"] = "neutral", appearance: Literal["soft", "solid", "outline"] = "soft", padding: tuple[float, float] = (0.14, 0.08), radius: Optional[float] = None, font_size: Optional[float] = None, color: Optional[ColorLike] = None, background: Optional[ColorLike] = None, border: Optional[ColorLike] = None, font: Optional[str] = None, weight:`

`Optional[int] = None, style: Optional[TextStyle] = None, markup: Optional[bool] = None) -> Drawable`

Insignia compacta con un punto opcional del color de la variante (`dot=True`), para filtros, estados y metadatos. Acepta las mismas opciones tipográficas que `badge`.

Ejemplo:

```
live = scene.slides.chip("En vivo", variant="danger", appearance="outline")
```

45.1.4. Tarjetas

Paneles de altura automática para contenido, métricas y citas.

45.1.5. SlideKit.card

FÁBRICA `card(title: str, body: Optional[str] = None, footer: Optional[str] = None, *, width: float = 4.2, min_height: float = 1.8, padding: tuple[float, float] = (0.28, 0.24), gap: float = 0.14, radius: float = 0.18, variant: Literal["neutral", "accent", "success", "warning", "danger"] = "neutral", appearance: Literal["soft", "solid", "outline"] = "soft", color: Optional[ColorLike] = None, background: Optional[ColorLike] = None, border: Optional[ColorLike] = None) -> Drawable`

Tarjeta con título, cuerpo y pie. Cada hueco usa un rol de texto del tema y se mide al construirla.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`title`	`str`	obligatorio	Encabezado.
`body / footer`	`str | None`	`None`	Cuerpo con ajuste de línea y pie opcional.
`width / min_height`	`float`	`4.2 / 1.8`	Ancho fijo y altura mínima.

Ejemplo:

```
result = scene.slides.card("Resultado", "El solver converge.", "12 ms", variant="accent")
```

45.1.6. SlideKit.stat_card

FÁBRICA `stat_card(value: str, label: str, *, delta: Optional[str] = None, width: float = 2.8, min_height: float = 1.7, padding: tuple[float, float] = (0.24, 0.20), gap: float = 0.08, radius: float = 0.18, variant: Literal["neutral", "accent", "success", "warning", "danger"] = "neutral", appearance: Literal["soft", "solid", "outline"] = "soft", color: Optional[ColorLike] = None, background: Optional[ColorLike] = None, border: Optional[ColorLike] = None) -> Drawable`

Tarjeta de métrica con valor, etiqueta y variación opcional (`delta`). El valor y la variación usan el tono de la variante; no se infiere signo ni formato numérico.

Ejemplo:

```
metric = scene.slides.stat_card("98%", "Precisión", delta="+4.2%", variant="success")
```

45.1.7. SlideKit.quote_card

FÁBRICA `quote_card(quote: str, attribution: Optional[str] = None, *, width: float = 6.2, padding: tuple[float, float] = (0.32, 0.28), gap: float = 0.16, radius: float = 0.18, variant: Literal["neutral", "accent", "success", "warning", "danger"] = "neutral", appearance: Literal["soft", "solid", "outline"] = "soft", color: Optional[ColorLike] = None, background: Optional[ColorLike] = None, border: Optional[ColorLike] = None) -> Drawable`

Cita con comillas tipográficas y atribución opcional alineada a la derecha.

Ejemplo:

```
quote = scene.slides.quote_card("La claridad importa.", "Gaanim", appearance="outline")
```

45.1.8. Rótulos y encabezados

Portadas, cabeceras de sección y rótulos anclados a los bordes seguros.

45.1.9. SlideKit.title_card

FÁBRICA `title_card(title: str, subtitle: Optional[str] = None, *, width: float = 7.6, height: float = 3.2, panel: bool = False, background: Optional[ColorLike] = None, color: Optional[ColorLike] = None, accent: Optional[ColorLike] = None) -> Drawable`

Portada centrada con título, regla y subtítulo opcional, animable como un solo grupo. `panel=True` la enmarca; `accent` colorea la regla.

Ejemplo:

```
from gaanim import Scene
scene = Scene(frame=(16, 9), background="#0f172a")
opening = scene.slides.title_card("Movimiento vectorial", "Una explicación técnica", panel=True)
scene.play([opening.animate.fade_in().duration(0.7)])
scene.render()
```

45.1.10. SlideKit.section_header

FÁBRICA `section_header(title: str, *, kicker: Optional[str] = None, subtitle: Optional[str] = None, width: float = 7.2, align: Literal["left", "center", "right"] = "left", rule: bool = False, padding: tuple[float, float] = (0.24, 0.18), gap: float = 0.10, radius: float = 0.12, variant: Literal["neutral", "accent", "success", "warning", "danger"] = "neutral", appearance: Literal["soft", "solid", "outline"] = "soft", color: Optional[ColorLike] = None, background: Optional[ColorLike] = None, border: Optional[ColorLike] = None) -> Drawable`

Encabezado de sección con antetítulo (`kicker`) y subtítulo opcionales. `align` (`left`, `center` o `right`) alinea todos los huecos; `rule=True` añade una regla de acento horizontal.

Ejemplo:

```
heading = scene.slides.section_header("Método", kicker="02", align="center")
```

45.1.11. SlideKit.banner

FÁBRICA `banner(title: str, subtitle: Optional[str] = None, *, position: Literal["top", "bottom"] = "top", width: Optional[float] = None, margin: float = 0.32, padding: tuple[float, float] = (0.28, 0.18), gap: float = 0.08, radius: float = 0.14, variant: Literal["neutral", "accent", "success", "warning", "danger"] = "neutral", appearance: Literal["soft", "solid", "outline"] = "soft", color: Optional[ColorLike] = None, background: Optional[ColorLike] = None, border: Optional[ColorLike] = None) -> Drawable`

Franja anclada al borde seguro superior o inferior (`position`). Con `width=None` ocupa el ancho seguro menos `margin`; su altura sigue al título y al subtítulo.

Ejemplo:

```python
notice = scene.slides.banner("Simulación completa", position="bottom", variant="success")
```

45.1.12. SlideKit.lower_third

FÁBRICA `lower_third(title: str, subtitle: Optional[str] = None, *, kicker: Optional[str] = None, side: Literal["left", "right"] = "left", width: float = 5.2, margin: float = 0.32, padding: tuple[float, float] = (0.28, 0.20), gap: float = 0.08, radius: float = 0.16, variant: Literal["neutral", "accent", "success", "warning", "danger"] = "neutral", appearance: Literal["soft", "solid", "outline"] = "soft", color: Optional[ColorLike] = None, background: Optional[ColorLike] = None, border: Optional[ColorLike] = None) -> Drawable`

Rótulo inferior anclado a una esquina segura (`side="left"` o `"right"`), con antetítulo, título y subtítulo que usan los roles del tema y se ajustan al ancho.

Ejemplo:

```python
speaker = scene.slides.lower_third("Ada Lovelace", "Matemática", kicker="PONENTE")
```

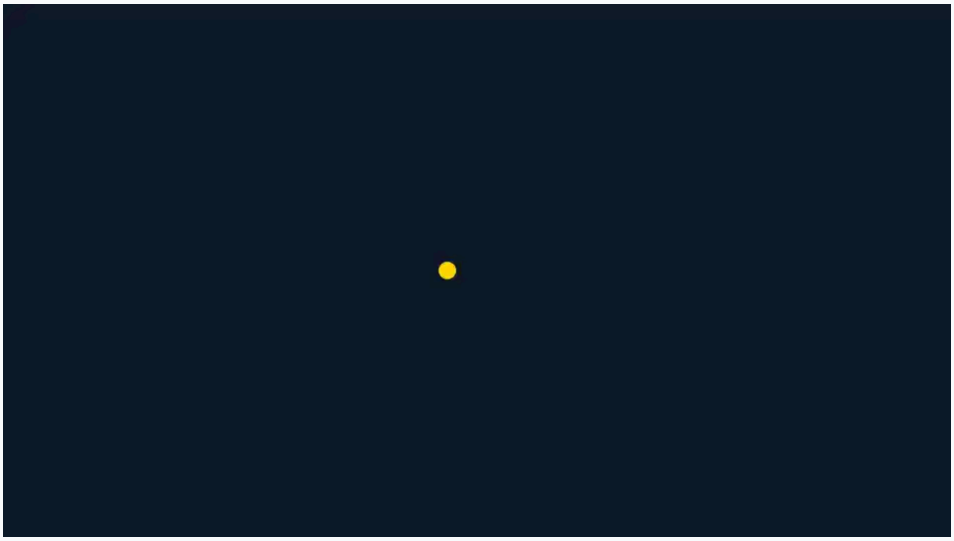
45.1.13. SlideKit.callout

FÁBRICA `callout(text: str, target: Drawable, *, offset: tuple[float, float] = (1.6, 0.96), width: float = 2.4, height: float = 0.72, background: Optional[ColorLike] = None, color: Optional[ColorLike] = None) -> Drawable`

Llamada con tarjeta, texto y conector que apunta a `target` desde `offset` y lo sigue de forma nativa, sin callbacks de Python por fotograma.

Ejemplo:

```python
from gaanim import GOLD, Scene
scene = Scene(frame=(16, 9), background="#0f172a")
mass = scene.geometry.dot(0.15).fill(GOLD).move_to(-0.5, 0)
note = scene.slides.callout("Masa en movimiento", mass, offset=(1.625, 0.875))
scene.play([mass.animate.shift_by(1, 0).duration(1.0), note.animate.fade_in().duration(0.4)])
scene.render()
```



45.1.14. Listas y tablas

Agendas con viñetas y tablas técnicas compactas.

45.1.15. SlideKit.bullets

FÁBRICA `bullets(items: Sequence[str], *, width: float = 7.2, gap: float = 0.68, bullet_radius: float = 0.08, bullet_color: Optional[ColorLike] = None, color: Optional[ColorLike] = None) -> Drawable`

Lista con viñetas como un solo objeto. Cada punto se alinea a la izquierda, justo después de su viñeta, y se ajusta al resto de `width`. Ajusta `width`, `gap`, `bullet_radius` y `bullet_color`.

Ejemplo:

```
from gaanim import GOLD, Scene
scene = Scene(frame=(16, 9), background="#0f172a")
agenda = scene.slides.bullets(["Preparar", "Animar", "Exportar"], gap=0.6,
bullet_color=GOLD).move_to(0, 0.5)
scene.play([agenda.animate.fade_in().duration(0.6)])
scene.render()
```

45.1.16. SlideKit.table

FÁBRICA `table(headers: Sequence[str], rows: Sequence[Sequence[str]], *, width: float = 7.6, row_height: float = 0.58, header_background: Optional[ColorLike] = None, rule_color: Optional[ColorLike] = None, color: Optional[ColorLike] = None) -> Drawable`

Tabla con cabecera y reglas. Cada fila debe tener tantas celdas como cabeceras. Para tablas con celdas combinadas usa `scene.text.typst(...)` (ver Texto).

Ejemplo:

```
from gaanim import Scene
scene = Scene(frame=(16, 9), background="#0f172a")
tbl = scene.slides.table(["Método", "Error", "Tiempo"], [["Base", "0.18", "48 ms"], ["GPU",
"0.04", "15 ms"]]).move_to(0, 0)
scene.play([tbl.animate.fade_in().duration(0.6)])
scene.render()
```

45.1.17. Identidad de la presentación

Logo, pie, regla y número de diapositiva comunes a todos los segmentos.

45.1.18. SlideKit.brand

MÉTODO `brand(*, logo: Optional[str] = None, footer: Optional[str] = None, slide_numbers: bool = True, number_anchor: Optional[AnchorName | Anchor] = None, rule: bool = True, show_on_cover: bool = False, logo_scale: float = 1.0) -> None`

Configura la identidad que se dibuja en cada segmento, por encima del contenido. Llámalo una vez, al principio. Consulta Presentaciones.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`logo`	`str | None`	`None`	SVG o imagen; se ajusta a 0.6 unidades de alto en la esquina segura superior derecha.
`footer`	`str | None`	`None`	Texto del pie. Empieza en el borde izquierdo del área segura y, si es muy largo, se reduce para caber.
`slide_numbers / rule`	`bool`	`True / True`	Número de diapositiva y regla del pie; la regla ocupa el ancho del área segura.
`number_anchor`	`Anchor | None`	`None`	Sin él, el número sigue al pie, abajo a la izquierda. Con un `Anchor`, va solo en ese punto del área segura: `Anchor.BOTTOM_RIGHT` lo deja libre de una barra de progreso a lo largo del borde inferior.
`show_on_cover`	`bool`	`False`	Muestra la identidad también en la portada: el primer segmento y los que usan una plantilla de portada.
`logo_scale`	`float`	`1.0`	Multiplica el tamaño del logo.

Ejemplo:

```
scene.slides.brand(logo="assets/logo.svg", footer="LAB · 2026")
scene.segment("Portada")
scene.wait(0.5)
scene.segment("Contenido")
scene.wait(0.5)
```

45.1.19. SlideKit.target

MÉTODO `target(seconds: Optional[float]) -> None`

Fija cuánto debe durar la presentación. Presenter View la cuenta hacia atrás en `LEFT`; la que fijes en la propia vista o con `gaanim --present --duration` la sustituye. `scene.segment(..., target=)` fija la de cada diapositiva. Un valor que no sea un número positivo lanza `ValueError`. Consulta El tiempo de la charla.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`seconds`	`float | None`	obligatorio	Duración prevista de la charla entera; `None` la quita.

Devuelve: `None` — No añade nada a la escena.

Ejemplo:

```
scene.slides.target(20 * 60)
scene.segment("Portada", target=60)
scene.wait(0.5)
```

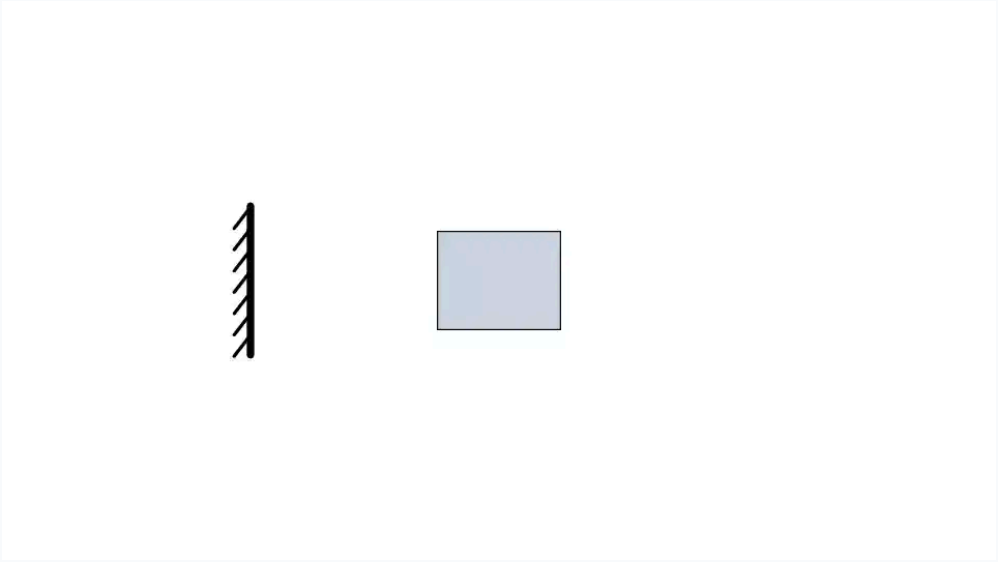
CAPÍTULO

46. Mecánica

Dibujo técnico y mecanismos con scene.mechanics: cotas, barras, muelles, fuerzas, apoyos y engranajes reactivos

`scene.mechanics` dibuja la notación de la mecánica y el dibujo técnico: cotas, ángulos, barras, muelles, vectores de fuerza, apoyos y engranajes. Casi todo es reactivo: los extremos aceptan objetos, `AnchorPoint` y `PointRef` (ver Geometría), y la geometría se regenera en el mismo fotograma cuando se mueven. Las magnitudes aceptan números o valores reactivos (`Parameter`, `Variable`, `Computed`).

```
from gaanim import Anchor, BLACK, Direction, WHITE, Scene

scene = Scene(frame=(16, 9), background=WHITE)
wall = scene.mechanics.fixed_support((-4, 0), direction=Direction.RIGHT, size=1.2, ground_length=2.4)
block = scene.geometry.rect(2, 1.6).fill("#cbd5e1").stroke(BLACK, 0.04).move_to(0, 0)
spring = scene.mechanics.spring_between((-4, 0), block.anchor_point(Anchor.LEFT), coils=7,
amplitude=0.3).no_fill().stroke(BLACK, 0.05)
gap = scene.mechanics.dimension_between((-4, -0.8), block.anchor_point(Anchor.BOTTOM_LEFT), 0.6,
side="below", show_value=True, unit="m", color=BLACK)
scene.play([spring.animate.fade_in(), gap.animate.fade_in()], duration=0.3)
scene.play([block.animate.shift_by(2, 0).duration(1.2)])
scene.render()
```



Los textos de las anotaciones (etiqueta, valor y unidad) miden 0.48 unidades por defecto, legibles en 1080p; `font_size` lo cambia. `color` pinta la geometría y la anotación completa, incluido el número cuando cambia o al hacer seek.

46.1.1. Cotas y ángulos

Medidas técnicas estáticas o que siguen a sus extremos.

46.1.2. Mechanics.dimension

FÁBRICA `dimension(x1: float, y1: float, x2: float, y2: float, offset: float) -> Drawable`

Cota estática entre `(x1, y1)` y `(x2, y2)`, desplazada `offset` en perpendicular, con líneas de extensión de 0.02 unidades y las puntas de `double_arrow`. Para una cota que siga a objetos móviles, usa `dimension_between`.

Ejemplo:

```
dim = scene.mechanics.dimension(-3, 0, 3, 0, 0.6).fill(WHITE)
```

46.1.3. Mechanics.dimension_between

FÁBRICA `dimension_between(from_: Endpoint, to: Endpoint, offset: float, *, label: Optional[str] = None, show_value: bool = False, value: Optional[_ReactiveScalar] = None, format: str = ".2f", unit: Optional[str] = None, scale: float = 1.0, label_gap: float = 0.10, label_orientation: Literal["upright", "aligned"] = "upright", font_size: Optional[float] = None, color: Optional[ColorLike] = None, line_width: float = 0.03, extension_style: Literal["solid", "dashed"] = "solid", dash_length: float = 0.12, gap_length: float = 0.08, side: Optional[Literal["left", "right", "above", "below"]] = None, font: Optional[str] = None, weight: Optional[int] = None, label_style: Optional[TextStyle] = None) -> Dimension`

Cota reactiva con líneas de extensión, puntas sólidas y anotación sincronizadas con los extremos. Como cualquier anotación, se ve desde su declaración; con una entrada (`fade_in`, `create`) espera oculta hasta que esta empieza. `value` solo controla el número: cambiarlo nunca cambia la longitud. Tipos, métricas, estilos u orientaciones inválidos lanzan `TypeError` o `ValueError`; un resultado reactivo no finito muestra el marcador de valor inválido.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`from_ / to`	`Endpoint`	obligatorio	Extremos medidos.
`offset`	`float`	obligatorio	Desplazamiento perpendicular. Sin `side`, positivo es a la izquierda de `from_` →

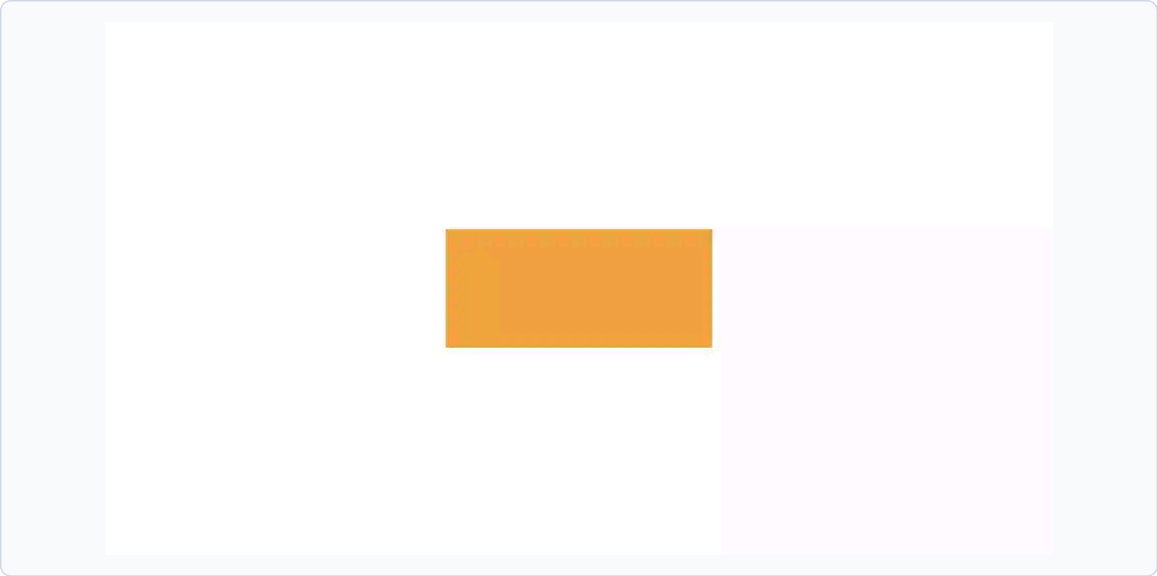
			`to`; con `side`, solo cuenta su magnitud.
`side`	`str | None`	`None`	`left`, `right`, `above` o `below`: mantiene la cota en ese lado de la escena aunque los extremos se crucen.
`label`	`str | None`	`None`	Texto simbólico o matemática en línea.
`show_value`	`bool`	`False`	Muestra la distancia actual multiplicada por `scale`.
`value`	`float | Parameter | Variable | Computed | None`	`None`	Lectura numérica propia; implica `show_value` y sustituye

			la distancia medida y `scale`.
`format / unit / scale`	`str / str | None / float`	`".2f" / None / 1`	Formato, unidad y factor de escena a unidades mostradas.
`label_gap / label_orientation`	`float / str`	`0.1 / "upright"`	Separación entre la línea y el borde más cercano de la anotación, y orientación `upright` o `aligned` (nunca boca abajo).
`line_width`	`float`	`0.03`	Ancho de la línea; las puntas miden seis anchos, limitadas en

			cotas cortas.
`extension_style / dash_length / gap_length`	`str / float / float`	`"solid" / 0.12 / 0.08`	Líneas de extensión `solid` o `dashed`.
`font / weight / label_style`	`str / int / TextStyle | None`	`None`	Tipografía de la anotación; `font`, `weight` y `font_size` sustituyen a `label_style`.

Ejemplo:

```
from gaanim import Anchor, BLACK, WHITE, Scene
scene = Scene(frame=(16, 9), background=WHITE)
frame = scene.geometry.rect(4.5, 2).move_to(0, 0)
physical_width = scene.viz.parameter(2.5)
dim = scene.mechanics.dimension_between(
  frame.anchor_point(Anchor.TOP_LEFT),
  frame.anchor_point(Anchor.TOP_RIGHT),
  0.6, label="$W_f$", value=physical_width, unit="m", color=BLACK,
  extension_style="dashed", line_width=0.03, dash_length=0.12, gap_length=0.08,
)
height = scene.mechanics.dimension_between(
  frame.anchor_point(Anchor.TOP_RIGHT),
  frame.anchor_point(Anchor.BOTTOM_RIGHT),
  0.6, side="right", show_value=True, unit="m", color=BLACK,
  font="DejaVu Sans Mono", weight=700,
)
scene.play([
  dim.animate.fade_in().duration(0.3),
  height.animate.fade_in().duration(0.3),
  physical_width.animate.set(4.0).duration(0.9),
])
scene.render()
```



46.1.4. Dimension.line / extensions / label / number / unit

PROPIEDAD `line · extensions · label · number · unit`

Tipo que devuelve `dimension_between`. Sus partes son `Drawable` que se estilizan por separado; `label`, `number` y `unit` valen `None` si no se pidieron.

Ejemplo:

```
dim = scene.mechanics.dimension_between((-2, 0), (2, 0), 0.5, label="$L$", show_value=True)
dim.extensions.opacity(0.4)
dim.number.fill(GOLD)
```

46.1.5. Mechanics.angle_between

FÁBRICA `angle_between(vertex: Endpoint, from_: AngleRay, to: AngleRay, *, radius: float = 0.64, label: Optional[str] = None, show_value: bool = False, format: str = ".1f", unit: Literal["deg", "rad"] = "deg", sweep: Literal["minor", "major", "cw", "ccw"] = "minor", arrowheads: Literal["none", "start", "end", "both"] = "both", label_gap: float = 0.12, label_orientation: Literal["upright", "aligned"] = "upright", show_extensions: bool = True, font_size: Optional[float] = None, color: Optional[ColorLike] = None) -> AngleDimension`

Cota angular reactiva. Un rayo de longitud cero oculta la geometría afectada en lugar de emitir caminos inválidos.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`vertex`	`Endpoint`	obligatorio	Vértice del ángulo.
`from_ / to`	`Direction | Endpoint`	obligatorio	Rayos como direcciones fijas o extremos.

`radius`	`float`	`0.64`	Radio del arco.
`sweep`	`str`	`"minor"`	`minor`, `major`, `cw` o `ccw`.
`arrowheads`	`str`	`"both"`	`none`, `start`, `end` o `both`.
`unit`	`str`	`"deg"`	`deg` o `rad` para el valor mostrado.
`show_extensions`	`bool`	`True`	Dibuja los rayos.

Ejemplo:

```
from gaanim import Direction, GOLD, Scene
scene = Scene(frame=(16, 9))
bob = scene.geometry.dot(0.125).move_to(1, -1.125)
theta = scene.mechanics.angle_between((0,0), Direction.DOWN, bob, label="$theta$",
show_value=True, color=GOLD)
scene.play([theta.animate.fade_in(), bob.animate.shift_by(1.125, 0.44).duration(1.2)])
scene.render()
```

46.1.6. AngleDimension.arc / arrows / extensions / label / number / unit

PROPIEDAD `arc · arrows · extensions · label · number · unit`

Tipo que devuelve `angle_between`, con partes que se estilizan por separado.

46.1.7. Barras y muelles

Elementos que unen dos extremos y cambian de longitud con ellos.

46.1.8. Mechanics.bar_between

FÁBRICA `bar_between(from_: Endpoint, to: Endpoint, *, width: float = 0.08) -> Drawable`

Barra de extremos redondeados con longitud y ángulo reactivos; `width` en unidades de escena. Dibuja las articulaciones aparte si las necesitas.

Ejemplo:

```python
from gaanim import Anchor, BLACK, Scene
scene = Scene(frame=(16, 9))
body = scene.geometry.rect(1.875, 0.875).move_to(0.5, -0.25)
bar = scene.mechanics.bar_between((-1.875, 1.25), body.anchor_point(Anchor.TOP_LEFT),
width=0.11).stroke(BLACK, 0.11)
scene.play([bar.animate.fade_in(), body.animate.shift_by(1, 0).duration(1.0)])
scene.render()
```

46.1.9. Mechanics.spring_between

FÁBRICA `spring_between(from_: Endpoint, to: Endpoint, coils: int = 8, amplitude: float = 0.12, crossing: float = 0.0, start_straight: float = 0.12, end_straight: float = 0.12) -> Drawable`

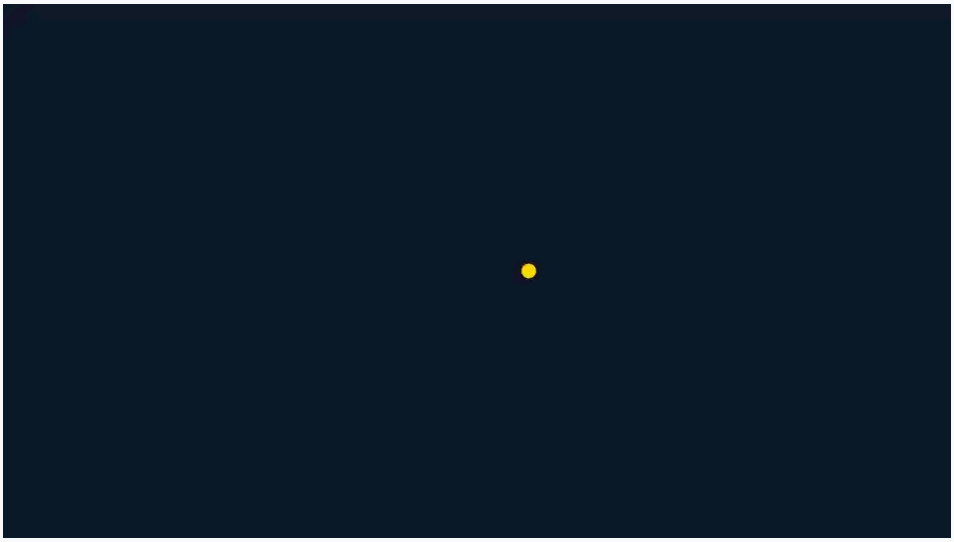
Muelle helicoidal oculto que conserva su radio mientras su paso se deforma con la distancia. Revélalo con una animación de entrada. Tramos negativos o no finitos lanzan `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`from_ / to`	`Endpoint`	obligatorio	Extremos.
`coils`	`int`	`8`	Número de vueltas.
`amplitude`	`float`	`0.12`	Radio de la hélice, perpendicular al eje.
`crossing`	`float`	`0`	De 0 a 1: pliega cada vuelta para crear cruces en forma de «e».
`start_straight / end_straight`	`float`	`0.12 / 0.12`	Tramos rectos no negativos en cada extremo; se acortan si los extremos están muy cerca.

Ejemplo:

```
from gaanim import GOLD, WHITE, Scene
scene = Scene(frame=(16, 9), background="#0f172a")
mass = scene.geometry.dot(0.125).fill(GOLD).move_to(0.875, 0)
spring = scene.mechanics.spring_between(( -0.875, 0), mass, coils=6, amplitude=0.175,
crossing=1.0, start_straight=0.225, end_straight=0.225).no_fill().stroke(WHITE, 0.04)
scene.play([spring.animate.fade_in().duration(0.3), mass.animate.shift_by(0.5, 0).duration(1.0)])
scene.render()
```



46.1.10. Fuerzas y momentos

Vectores con punta sólida y lectura opcional, en unidades físicas.

46.1.11. Mechanics.vector_between

FÁBRICA `vector_between(from_: Endpoint, to: Endpoint, *, label: Optional[str] = None, show_value: bool = False, format: str = ".1f", unit: Optional[str] = None, scale: float = 1.0, label_gap: float = 0.14, font_size: Optional[float] = None, color: Optional[ColorLike] = None) -> ForceVector`

Vector reactivo entre dos extremos con asta, punta sólida y lectura opcional de su magnitud (`show_value`, `format`, `unit`, `scale`).

Ejemplo:

```
force = scene.mechanics.vector_between((-1.25, 0), (0.875, 0.56), label="$F$", color=GOLD)
```

46.1.12. Mechanics.force_at

FÁBRICA `force_at(origin: Endpoint, magnitude: _ReactiveScalar, *, direction: _ReactiveScalar = 0.0, visual_scale: float = 1.0, label: Optional[str] = None, show_value: bool = False, format: str = ".1f", unit: str = "N", label_gap: float = 0.14, font_size: Optional[float] = None, color: Optional[ColorLike] = None) -> ForceVector`

Fuerza a partir de magnitud y dirección. La lectura opcional informa la magnitud física, no la longitud dibujada.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`origin`	`Endpoint`	obligatorio	Punto de aplicación.
`magnitude`	`float | fuente reactiva`	obligatorio	Magnitud física.
`direction`	`float | fuente reactiva`	`0.0`	Dirección en radianes.
`visual_scale`	`float`	`1.0`	Unidades de escena por unidad física; positiva.
`unit`	`str`	`"N"`	Unidad de la lectura.

Ejemplo:

```
from gaanim import GREEN, Scene
scene = Scene(frame=(16, 9))
body = scene.geometry.circle(0.3)
magnitude = scene.viz.parameter(30)
force = scene.mechanics.force_at(
  body.anchor_point(), magnitude, direction=0.5, visual_scale=0.025,
  label="$F$", show_value=True, unit="N", color=GREEN,
)
scene.play([body.animate.fade_in(), force.animate.fade_in(),
magnitude.animate.set(80).duration(1.5)])
scene.render()
```

46.1.13. Mechanics.force_from_components

FÁBRICA `force_from_components(origin: Endpoint, fx: _ReactiveScalar, fy: _ReactiveScalar, *, visual_scale: float = 1.0, label: Optional[str] = None, show_value: bool = False, format: str = ".1f", unit: str = "N", label_gap: float = 0.14, font_size: Optional[float] = None, color: Optional[ColorLike] = None) -> ForceVector`

Fuerza a partir de sus componentes físicas X e Y respecto a un origen móvil. Combínala con `Parameter.add_updater_fn` para simulaciones: las simulaciones de paso fijo se reconstruyen antes, así que una fuerza derivada del cuerpo simulado ve el estado del mismo fotograma.

Ejemplo:

```
body = scene.geometry.circle(0.3)
fx = scene.viz.parameter(20)
weight = scene.mechanics.force_from_components(body, fx, -40, visual_scale=0.02, label="$W$")
```

46.1.14. ForceVector.shaft / head / label / number / unit

PROPIEDAD `shaft · head · label · number · unit`

Tipo que devuelven `vector_between`, `force_at` y `force_from_components`, con partes que se estilizan por separado.

46.1.15. Mechanics.moment_about

FÁBRICA `moment_about(center: Endpoint, radius: float, *, direction: Literal["cw", "ccw"] = "ccw", label: Optional[str] = None, color: Optional[ColorLike] = None) -> Drawable`

Flecha curva de momento que sigue a un centro reactivo; `direction` es `"cw"` o `"ccw"`.

Ejemplo:

```
moment = scene.mechanics.moment_about((1.5, -0.5), radius=0.525, label="$M$")
```

46.1.16. Mechanics.coordinate_frame_at

FÁBRICA `coordinate_frame_at(origin: Endpoint, x_direction: Direction, *, length: float = 0.70, labels: Optional[tuple[str, str]] = None, color: Optional[ColorLike] = None) -> Drawable`

Sistema de ejes 2D ortogonal y reactivo en un extremo, con el eje X en `x_direction` y etiquetas opcionales.

Ejemplo:

```
from gaanim import Direction, GOLD, Scene
scene = Scene(frame=(16, 9))
force = scene.mechanics.vector_between((-1.25, 0), (0.875, 0.56), label="$F$", color=GOLD)
moment = scene.mechanics.moment_about((1.5, -0.5), radius=0.525, label="$M$")
frame = scene.mechanics.coordinate_frame_at((0, -1), Direction.RIGHT, labels=("$e_1$", "$e_2$"))
scene.play([force.animate.fade_in(), moment.animate.fade_in(), frame.animate.fade_in()])
scene.render()
```

46.1.17. Mechanics.contact_on_curve

FÁBRICA `contact_on_curve(curve: Drawable, tracker: _ReactiveScalar, *, tangent_length: float = 0.8, normal_length: float = 0.8) -> Drawable`

Agrupa un punto de contacto, su tangente y su normal sobre una curva muestreada, conducidos por un `Parameter` o `Variable` de 0 a 1.

Ejemplo:

```
t = scene.viz.parameter(0.1)
track = scene.geometry.polyline([(x / 10, 0.5 * math.sin(x / 10)) for x in range(-40,
41)]).no_fill()
contact = scene.mechanics.contact_on_curve(track, t, tangent_length=1.0, normal_length=0.6)
scene.play([contact.animate.fade_in(), t.animate.set(0.9).duration(1.5)])
```

46.1.18. Apoyos y articulaciones

Símbolos de apoyo con base rayada y juntas, que siguen a su punto de conexión.

46.1.19. Mechanics.support_at

FÁBRICA `support_at(point: Endpoint, *, kind: Literal["fixed", "pin", "roller", "simple", "guided", "prismatic", "cable", "spring"] = "pin", direction: Optional[Direction] = None, size: float = 0.48, ground_length: float = 0.70, color: Optional[ColorLike] = None) -> Support`

Apoyo vectorial que respeta el tema. Los atajos `fixed_support`, `pin_support`, `roller_support` y `guided_support` fijan `kind`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`point`	`Endpoint`	obligatorio	Punto de conexión.

`kind`	`str`	`"pin"`	`fixed`, `pin`, `roller`, `simple`, `guided`, `prismatic`, `cable` o `spring`.
`direction`	`Direction | None`	`None`	De la base hacia la conexión: `UP` pone el suelo debajo y `DOWN` crea un apoyo de techo.
`size / ground_length`	`float`	`0.48 / 0.70`	Medidas en unidades de escena.

Ejemplo:

```
anchor = scene.mechanics.support_at((0, 2), kind="cable", direction=Direction.DOWN)
```

46.1.20. Mechanics.pin_support

FÁBRICA `pin_support(point: Endpoint, *, direction: Optional[Direction] = None, size: float = 0.48, ground_length: float = 0.70, color: Optional[ColorLike] = None) -> Support`

Apoyo triangular articulado con junta circular.

Ejemplo:

```
from gaanim import Direction, Scene
scene = Scene(frame=(16, 9))
pin = scene.mechanics.pin_support((-1.25, 0), direction=Direction.UP)
roller = scene.mechanics.roller_support((1.25, 0), direction=Direction.UP)
scene.play([pin.animate.fade_in(), roller.animate.fade_in()])
scene.render()
```

46.1.21. Mechanics.roller_support

FÁBRICA `roller_support(point: Endpoint, *, direction: Optional[Direction] = None, size: float = 0.48, ground_length: float = 0.70, color: Optional[ColorLike] = None) -> Support`

Apoyo triangular sobre dos rodillos alineados.

46.1.22. Mechanics.fixed_support

FÁBRICA `fixed_support(point: Endpoint, *, direction: Optional[Direction] = None, size: float = 0.48, ground_length: float = 0.70, color: Optional[ColorLike] = None) -> Support`

Empotramiento (o apoyo de techo) con placa y rayado.

Ejemplo:

```
wall = scene.mechanics.fixed_support((-4, 0), direction=Direction.RIGHT)
```

46.1.23. Mechanics.guided_support

FÁBRICA `guided_support(point: Endpoint, *, direction: Optional[Direction] = None, size: float = 0.48, ground_length: float = 0.70, color: Optional[ColorLike] = None) -> Support`

Carro guiado alineado con `direction`.

46.1.24. Support.joint / body / ground / rollers / guides / hatching

PROPIEDAD `joint · body · ground · rollers · guides · hatching`

Tipo que devuelven los apoyos, con partes vectoriales que se estilizan por separado.

Ejemplo:

```
base = scene.mechanics.pin_support((0, 0))
base.hatching.opacity(0.5)
```

46.1.25. Mechanics.joint_at

FÁBRICA `joint_at(point: Endpoint, *, kind: Literal["revolute", "prismatic"] = "revolute", axis: Optional[Direction] = None, size: float = 0.36, color: Optional[ColorLike] = None) -> Drawable`

Símbolo independiente de junta de revolución (`kind="revolute"`) o prismática (`"prismatic"`, alineada con `axis`) que sigue a su punto.

Ejemplo:

```
hinge = scene.mechanics.joint_at((0, 0))
slider = scene.mechanics.joint_at((2, 0), kind="prismatic", axis=Direction.RIGHT)
```

46.1.26. Transmisiones

Engranajes, cremalleras y levas editoriales. Son relaciones visuales, no un solver cinemático ni de contacto; los dientes son esquemáticos.

46.1.27. Mechanics.gear

FÁBRICA `gear(radius: float, teeth: int, *, bore_radius: float = 0.08, color: Optional[ColorLike] = None) -> Drawable`

Silueta de engranaje de radio `radius` con `teeth` dientes y agujero central. Acóplalos con `Drawable.bind_rotation_from(source, ratio=..., phase=...)`.

Ejemplo:

```
from gaanim import Direction, Scene
scene = Scene(frame=(16, 9))
driver = scene.mechanics.gear(0.69, 20).move_to(-0.69, 0.25)
driven = scene.mechanics.gear(0.41, 12).move_to(0.41, 0.25).bind_rotation_from(driver, ratio=-5/3)
rack = scene.mechanics.rack(2.75, 18).move_to(0, -0.81).bind_translation_from_rotation(
  driver, axis=Direction.RIGHT, scale=0.69,
)
scene.play([driver.animate.fade_in(), driven.animate.fade_in(), rack.animate.fade_in(),
driver.animate.rotate_by(2.0)])
scene.render()
```

46.1.28. Mechanics.rack

FÁBRICA `rack(length: float, teeth: int, *, color: Optional[ColorLike] = None) -> Drawable`

Cremallera recta de `length` con `teeth` dientes equiespaciados. Muévela con `Drawable.bind_translation_from_rotation`.

46.1.29. Mechanics.cam_profile

FÁBRICA `cam_profile(samples: Sequence[tuple[float, float]], *, bore_radius: float = 0.08, color: Optional[ColorLike] = None) -> Drawable`

Leva radial cerrada a partir de muestras `(ángulo_en_radianes, radio)`.

Ejemplo:

```
samples = [(2 * math.pi * i / 64, 1 + 0.3 * math.cos(2 * math.pi * i / 64)) for i in range(64)]
cam = scene.mechanics.cam_profile(samples)
scene.play([cam.animate.rotate_by(math.tau).duration(2)])
```

CAPÍTULO

47. Público

Poll, Leaderboard, Teams, Audience, Character, LiveZone, Condition, gaanim.live y las preguntas

Los objetos que dan a la escena lo que hace el público: las encuestas y cuestionarios, la clasificación, los equipos, la sala, los personajes y las zonas vivas. Ninguno dibuja nada por sí mismo: dan `Parameter`, textos en vivo, barras y condiciones para que diseñes el juego. Fuera de una presentación en vivo siguen al ensayo (`scene.rehearsal`); al presentar, al relay.

Los métodos de `Scene` que los crean (`poll`, `quiz`, `question`, `leaderboard`, `teams`, `audience`, `rehearsal`, `character` y `live_zone`) están en Escena. Las guías están en Público en vivo.

47.1.1. Encuestas y cuestionarios

47.1.2. Poll

CLASE

Una encuesta de `scene.poll` o un cuestionario de `scene.quiz`, o lo que abre `scene.question`. Sus valores siguen los votos en vivo al presentar y el ensayo en todo lo demás. Al presentar un paquete `.gaanim`, las barras y las lecturas que muestran directamente uno de sus `Parameter` siguen los votos en vivo; lo que pasa por un `computed` muestra lo grabado.

Ejemplo:

```
poll = scene.poll("¿Té o café?", ["Té", "Café"], rehearse=[2, 3])
scene.viz.readout(poll.percent(1), format=".0f", suffix=" % café")
```

47.1.3. Poll.id / question / options

PROPIEDAD

El identificador estable de la pregunta en el relay, el texto de la pregunta y sus respuestas, en el orden dado.

47.1.4. Poll.code / url / qr

MÉTODO

`code` es el código de seis caracteres de la sesión y `url` la dirección que abre el QR. `qr(size=3.0)` devuelve el QR como un drawable de `size` unidades de lado, relleno de negro: ponlo sobre un fondo claro con margen. Un tamaño no positivo lanza `ValueError`.

Ejemplo:

```
scene.geometry.rounded_rect(3.4, 3.4, 0.2).fill(WHITE).no_stroke().move_to(-5, 0)
poll.qr(3.0).move_to(-5, 0)
scene.text(poll.code, size=0.5).move_to(-5, -2.2)
```

47.1.5. Poll.votes / share / percent / total

MÉTODO

`Parameter` que siguen los votos: `votes(i)` los de la respuesta `i` (0 para la primera), `share(i)` su fracción de 0 a 1, `percent(i)` lo mismo de 0 a 100 y `total()` todos los votos. En un cuestionario `total()` cuenta a los jugadores que respondieron; en una encuesta de selección múltiple, `votes(i)` cuenta cada respuesta elegida y `total()` y `share(i)` cuentan teléfonos. Sin votos, `share` y `percent` valen 0. Una respuesta que la pregunta no tiene lanza `ValueError`.

Ejemplo:

```
scene.viz.readout(poll.votes(0), format=".0f", prefix="Té: ")
scene.viz.readout(poll.total(), format=".0f", suffix=" votos").move_to(0, -1)
```

47.1.6. Poll.bar

MÉTODO `bar(answer: int, *, length: float = 6.0, thickness: float = 0.5, direction: Literal["right", "left", "up", "down"] = "right", scale: Literal["leader", "total"] = "leader", radius: float = 0.0) -> Drawable`

Una barra cuya longitud sigue los votos de `answer`. Sus límites son la caja completa de `length` por `thickness`, centrada en su posición, así que el layout no se mueve al llegar votos; crece desde el borde opuesto a `direction` (`"right"`, `"left"`, `"up"`, `"down"`). Con `scale="leader"` la respuesta que va ganando llena su barra; con `"total"` cada barra mide su parte del total. Se estiliza y anima como cualquier drawable (`create` la hace crecer) y sigue los votos en vivo también en un `.gaanim`. Una respuesta, dirección o escala desconocida, una longitud o grosor no positivos o un radio negativo lanzan `ValueError`.

Ejemplo:

```
poll.bar(0, length=5, thickness=0.4, radius=0.2).fill(RED).no_stroke().move_to(1, 1)
```

47.1.7. Poll.icon / color

MÉTODO

`icon(answer, size=0.6)` es la forma que la respuesta tiene en los teléfonos (triángulo, rombo, círculo, cuadrado, estrella o hexágono), de `size` unidades de alto, centrada y rellena de su color. `color(answer)` es ese color, `#rrggbb`. En un cuestionario los teléfonos ordenan y colorean las respuestas a su manera, así que solo en una encuesta la pantalla coincide con el teléfono. Una respuesta desconocida o un tamaño no positivo lanzan `ValueError`.

Ejemplo:

```
for i, answer in enumerate(poll.options):
    scene.geometry.rounded_rect(6, 1.4, 0.3).fill(poll.color(i)).no_stroke().move_to(0, 1 - 1.8 *
i)
    poll.icon(i, 0.6).fill(WHITE).move_to(-2.2, 1 - 1.8 * i)
```

47.1.8. Poll.answered / time_up

MÉTODO

Condiciones para `scene.stop(until=...)`. `answered(at_least=n)` se cumple con al menos `n` respuestas; `answered(share=f)` cuando respondió esa parte del público: los jugadores en un cuestionario y los teléfonos conectados en una encuesta; nunca se cumple sin nadie. `time_up()` se cumple cuando se acaba el tiempo del cuestionario en el reloj del relay. `answered` sin exactamente uno de sus argumentos o con una parte fuera de (0, 1], y `time_up` en una encuesta, lanzan `ValueError`.

Ejemplo:

```
quiz = scene.quiz("¿2 + 2?", ["3", "4"], correct=1)
scene.stop(until=quiz.answered(share=0.9) | quiz.time_up())
```

47.1.9. Poll.remaining

MÉTODO `remaining() -> Parameter`

Los segundos que quedan para responder un cuestionario, como `Parameter`. En la previsualización y la exportación baja con el ensayo y llega a 0 en la pausa donde la presentación espera las respuestas; al presentar sigue el reloj del relay. En una encuesta lanza `ValueError`.

Ejemplo:

```
clock = scene.viz.readout(quiz.remaining(), format=".0f", suffix=" s").move_to(6, 3.5)
```

47.1.10. Poll.reveal

MÉTODO `reveal() -> None`

Revela la respuesta de un cuestionario en el cursor: cuando la presentación pasa por este punto, cada teléfono muestra la respuesta correcta y al jugador si acertó, los puntos que ganó y su puesto; el cuestionario ya no acepta respuestas. Escrito justo después de un `scene.stop()`, espera mientras la presentación descansa en la pausa y revela al avanzar. En una encuesta, o una segunda vez, lanza `ValueError`.

Ejemplo:

```
quiz = scene.quiz("¿2 + 2?", ["3", "4"], correct=1)
scene.stop()
quiz.reveal()
```

47.1.11. Poll.revealed

MÉTODO `revealed() -> Parameter`

Vale 0 hasta que el cuestionario revela su respuesta y 1 desde entonces. Mostrar barras o porcentajes durante la pregunta influye en las respuestas: con este valor la escena decide qué hacer hasta el `reveal()`, ocultarlos, taparlos o mostrar «?». El cambio es un momento de la línea de tiempo, igual en la vista previa, en la exportación y en vivo, donde la presentación revela al llegar a él. Se pide antes del `reveal()`; después, o en una encuesta, lanza `ValueError`.

Devuelve: `Parameter` —

Ejemplo:

```
quiz = scene.quiz("¿2 + 2?", ["3", "4"], correct=1)
shown = quiz.revealed()
for i in range(2):
    scene.viz.readout(quiz.percent(i), format=".0f", suffix="%").move_to(4 * i - 2,
0).opacity(shown)
scene.wait(2)
scene.stop()
quiz.reveal()
```

47.1.12. Poll.close

MÉTODO `close() -> None`

Deja de recibir votos en el cursor, en vez de al final del segmento donde se abrió. Los valores conservan el último conteo. Si ya estaba cerrada lanza `ValueError`.

47.1.13. Poll.is_quiz / correct / multiple / time / has_image

PROPIEDAD

`is_quiz` dice si es un cuestionario; `correct` es el índice de la respuesta correcta, la lista de ellas en selección múltiple o `None` en una encuesta; `multiple` si se pueden elegir varias; `time` los segundos de un cuestionario (`None` en una encuesta); `has_image` si los teléfonos muestran una imagen sobre la pregunta.

47.1.14. Clasificación

47.1.15. Leaderboard

CLASE

La clasificación de `scene.leaderboard()`: los jugadores de los cuestionarios, de mejor a peor. Cada método recibe un puesto, 0 para el primero. Un puesto sin jugador muestra un apodo vacío, 0 puntos y una barra vacía. Todo sigue a los resultados en vivo, también en un `.gaanim`, que guarda los glifos de los apodos.

Ejemplo:

```
board = scene.leaderboard()
for rank in range(3):
    board.name(rank, size=0.5).move_to(-3, 1 - rank)
    scene.viz.readout(board.points(rank), format=".0f").move_to(3, 1 - rank)
```

47.1.16. Leaderboard.name

MÉTODO `name(rank: int, *, size: Optional[float] = None, weight: Optional[int] = None, font: Optional[str] = None, align: Literal["left", "center", "right"] = "left") -> Drawable`

El apodo del puesto `rank` como texto en vivo. `size` y `font` toman el texto del tema por defecto; `align` coloca el borde izquierdo, el centro o el borde derecho del texto en la posición del drawable, para que apodos de cualquier largo queden alineados. Se rellena con el color de primer plano del tema; cámbialo con `fill`. Un `align` desconocido o un tamaño no positivo lanzan `ValueError`.

Ejemplo:

```
board.name(0, size=0.6, align="left").fill(GOLD).move_to(-3, 1)
```

47.1.17. Leaderboard.points / players / correct / answered / streak

MÉTODO

`Parameter` del juego hasta ahora: `points(rank)` los puntos del puesto, `correct(rank)` sus aciertos, `answered(rank)` las preguntas que respondió, `streak(rank)` sus aciertos seguidos hasta la última revelada y `players()` cuántos jugadores hay.

47.1.18. Leaderboard.responded / chose / right / earned / answer_time

MÉTODO

`Parameter` de una pregunta para el jugador del puesto `rank`: `responded(rank, poll)` vale 1 cuando respondió; `chose(rank, poll, answer)` 1 si eligió esa respuesta (también en selección múltiple); `right(rank, quiz)` 1 si acertó; `earned(rank, quiz)` los puntos que ganó y `answer_time(rank, quiz)` los segundos que tardó, 0 sin respuesta. `right` en una encuesta lanza `ValueError`.

47.1.19. Leaderboard.bar

MÉTODO `bar(rank: int, *, length: float = 6.0, thickness: float = 0.5, direction: Literal["right", "left", "up", "down"] = "right", radius: float = 0.0) -> Drawable`

Una barra que mide los puntos del puesto `rank` frente a los del primero. Sus límites son la caja completa y crece desde el borde opuesto a `direction`, como `Poll.bar`, con los mismos errores.

47.1.20. Equipos

47.1.21. Teams

CLASE

Los equipos de `scene.teams(...)`. Cada método recibe el índice de un equipo, en el orden de los nombres. Fuera de una presentación siguen al ensayo, que reparte a sus jugadores igual que

el relay. En una zona viva cada jugador sabe `p.team`, `p.team_index`, `p.team_count`, `p.team_score` y `p.team_rank`.

Ejemplo:

```
teams = scene.teams(["Rojo", "Azul"], colors=["#ff4f8b", "#2fb8ff"])
for team, x in [(0, -4), (1, 4)]:
    scene.text(teams.names[team], size=0.6).fill(teams.colors[team]).move_to(x, 2)
    scene.viz.readout(teams.score(team), format=".0f", suffix=" pts").move_to(x, 1)
```

47.1.22. Teams.names / colors / choose

PROPIEDAD

Los nombres de los equipos, sus colores `#rrggbb` como los muestran los teléfonos, y si cada jugador elige su equipo en el teléfono. `len(teams)` es cuántos equipos hay.

47.1.23. Teams.score / players / average / leader

MÉTODO

`Parameter` de cada equipo: `score(team)` suma los puntos de sus jugadores, `players(team)` cuenta sus jugadores y `average(team)` da los puntos por jugador (0 sin jugadores), más justo cuando los equipos quedan desparejos. `leader()` es el índice del equipo que va ganando, el primero en un empate. Un equipo que el juego no tiene lanza `ValueError`.

47.1.24. Teams.bar

MÉTODO `bar(team: int, *, length: float = 6.0, thickness: float = 0.5, direction: Literal["right", "left", "up", "down"] = "right", radius: float = 0.0) -> Drawable`

Una barra que mide los puntos del equipo frente a los del que va ganando, como `Poll.bar`, con los mismos errores.

47.1.25. Sala

47.1.26. Audience

CLASE

El público de `scene.audience()`: los jugadores en el orden en que entraron, cada uno en un puesto, 0 para el primero. Un puesto vacío muestra un apodo vacío y `joined` y `age` en 0. Si quitas a un jugador, los siguientes suben un puesto. Una escena que lo usa pide el apodo en cuanto el teléfono abre la página.

Ejemplo:

```
audience = scene.audience()
for slot in range(8):
    audience.name(slot, size=0.4).move_to(-6 + slot * 1.7, -2)
scene.viz.readout(audience.count(), format=".0f", suffix=" en la sala")
```

47.1.27. Audience.code / url / qr

MÉTODO

El código de la sesión, la dirección para unirse y `qr(size=3.0)`, el QR de esa dirección, como en `Poll`.

47.1.28. Audience.name

MÉTODO `name(slot: int, *, size: Optional[float] = None, weight: Optional[int] = None, font: Optional[str] = None, align: Literal["left", "center", "right"] = "center") -> Drawable`

El apodo del puesto `slot` como texto en vivo, centrado por defecto; acepta `size`, `weight`, `font` y `align` como `Leaderboard.name`.

47.1.29. Audience.count / joined / age / at_least

MÉTODO

`count()` cuántos jugadores entraron; `joined(slot)` vale 1 cuando el puesto está ocupado; `age(slot)` cuenta los segundos desde que entró, hasta 60, para animar su llegada. `at_least(n)` es una condición para `scene.stop(until=...)` que se cumple con al menos `n` jugadores.

Ejemplo:

```
pop = computed(lambda a: min(a / 0.4, 1), inputs=[audience.age(0)])
audience.name(0, size=0.5).scale_to(pop)
scene.stop("sala", until=audience.at_least(5))
```

47.1.30. Audience.score / correct / answered / streak

MÉTODO

Los mismos datos del juego que `Leaderboard`, para el jugador de un puesto de la sala en vez de uno de la clasificación.

47.1.31. Audience.responded / chose / right / earned / answer_time

MÉTODO

Los mismos datos de una pregunta que `Leaderboard`, para el jugador de un puesto de la sala.

47.1.32. Personajes y zonas vivas

47.1.33. Character

CLASE

Un personaje como los que arma el público, de `scene.character(...)`: respira, parpadea y hace expresiones igual que en el teléfono. Se coloca, escala y anima como cualquier drawable.

Ejemplo:

```
hero = scene.character([1, 0, 4, 2, 1], name="Ana").move_to(-3, 0)
hero.express("winner", loop=True)
scene.wait(2)
```

47.1.34. Character.express / expressions / avatar

MÉTODO

`express(expression=None, loop=False)` hace una expresión desde el cursor: una vez, o con `loop=True` hasta la siguiente; `None` vuelve a su cara. Las expresiones, que da `Character.expressions()`, son `happy`, `sad`, `hurt`, `winner` y `surprised`; los lentes se quedan puestos. `avatar` son sus partes: cuerpo, color, ojos, boca y extra. Una expresión desconocida lanza `ValueError`.

47.1.35. LiveZone

CLASE

Una parte de la escena donde juega el público, de `scene.live_zone(...)`. Al presentar, cada jugador llega como su personaje y el comportamiento lo coloca en cada cuadro; la vista previa y la exportación reproducen el ensayo, siempre igual. El comportamiento se compila dentro de la escena, así que un `.gaanim` lo ejecuta sin Python. Los personajes se dibujan sobre el resto, recortados a los límites de la zona.

47.1.36. LiveZone.close / instructions

MÉTODO

`close()` cierra la zona en el cursor en vez de al final del segmento; si ya estaba cerrada lanza `ValueError`. `instructions` es cuántas instrucciones ejecuta el comportamiento compilado por jugador.

47.1.37. Condition

CLASE

Lo que el público debe hacer para que una pausa avance sola con `scene.stop(until=...)`. Se crea con `poll.answered(at_least=)` o `poll.answered(share=)`, `quiz.time_up()` y `audience.at_least(n)`, y se combina con `|` (cualquiera) y `&` (todas). Ver Avanzar por sí sola.

Ejemplo:

```
quiz = scene.quiz("¿2 + 2?", ["3", "4"], correct=1)
scene.stop(until=quiz.answered(share=0.8) | quiz.time_up())
```

47.1.38. gaanim.live

El módulo `gaanim.live` tiene lo que usan los comportamientos de una zona viva. Ver la guía Zonas vivas para el subconjunto de Python que se compila.

47.1.38.1. pose

`pose(x, y, *, rotation=0, scale=1, sx=1, sy=1, lean=0, look_x=None, look_y=None, show_name=True, flip=False, visible=True, express=None, since=None, loop=False)` dice dónde está un personaje y cómo se ve.

Argumento	Qué hace
`x`, `y`	Dónde están sus pies, en unidades de la escena.
`rotation`	Lo gira sobre su centro, en radianes, en sentido antihorario.
`scale`	Multiplica el tamaño de la zona.
`sx`, `sy`	Lo estiran a lo ancho y a lo largo desde los pies, para aplastar y estirar (`impact` y `anticipate` los dan).
`lean`	Lo inclina desde los pies.
`look_x`, `look_y`	Hacia dónde miran sus ojos, de −1 a 1 (derecha y arriba); sin ellos mira hacia donde se mueve.
`show_name`	`False` oculta el apodo en una zona que los dibuja.
`flip`	Lo refleja de izquierda a derecha.
`visible`	`False` lo oculta.
`express`	Una expresión: `"happy"`, `"sad"`, `"hurt"`, `"winner"` o `"surprised"`.
`loop`	Repite la expresión hasta otra.
`since`	El `p.t` en que empezó la expresión, para las que siguen a un momento que calcula el comportamiento, como aterrizar; sin él, empieza cuando aparece.

47.1.38.2. El jugador

El comportamiento recibe al jugador `p` con estos campos:

Campo	Qué es
`t`, `time`	Segundos desde que llegó a la zona, y desde que se abrió.
`joined`, `index`, `count`	Cuándo llegó, su orden de llegada desde 0 y cuántos hay.
`rank`, `score`, `leader`	Su puesto (0 el primero), sus puntos y los del primero.
`previous_rank`, `rank_since`	Su puesto anterior y los segundos desde que cambió.
`previous_score`, `score_since`	Sus puntos anteriores y los segundos desde que cambiaron.
`team`, `team_index`, `team_count`	Su equipo, su orden de llegada en él y cuántos son.
`team_score`, `team_rank`	Los puntos de su equipo y su puesto (0 el que va ganando).
`answer`, `answer_mask`	La respuesta que eligió en la pregunta abierta o la última (-1 si ninguna) y todas las que eligió, un bit cada una.
`answer_time`, `answer_points`	Lo que tardó en responder y lo que ganó.
`answers`, `correct`, `streak`	Preguntas respondidas, acertadas y aciertos seguidos en el juego.
`random(k)`	Un número entre 0 y 1 que depende solo del jugador y de `k`.
`state.<nombre>`	Los números que la zona guarda para él.

`gaanim.live.Player` crea un jugador de muestra con esos campos, para probar un comportamiento en Python: `fila(Player(index=2, t=0.5))`.

47.1.38.3. Estado y ayudas

- `state(**valores)`: los siguientes números guardados, como los devuelve una función `update`; los que no nombra conservan su valor.
- `STEP`: cada cuánto corre `update`, 1/60 s. `MAX_STATE`: cuántos números puede guardar una zona, 8.

- `clamp(x, low=0, high=1)`, `lerp(a, b, u)` y `progress(t, start=0, duration=1)`.
- Curvas de 0 a 1: `smoothstep(u)`, `ease_in(u)`, `ease_out(u)`, `ease_in_out(u)` y `ease_out_back(u, overshoot=1.7)`, que se pasa y vuelve.
- Movimiento: `spring(t, frequency=2, damping=0.3)`, un resorte de 0 a 1; `wobble(t, amount=1, frequency=3, decay=4)`, una sacudida que se apaga; `impact(t, amount=0.35, ...)` y `anticipate(t, at, amount=0.25, duration=0.3)`, que devuelven `(sx, sy)` para aplastar al aterrizar y agacharse antes de saltar; `ballistic(t, x, y, vx, vy, gravity=9.8)`, la posición de un tiro parabólico, y `landing(vy, drop, gravity=9.8)`, cuándo cae.
- `BehaviorError`: lo que lanza `scene.live_zone` cuando el comportamiento usa algo que no se compila, señalando la línea.

47.1.39. Preguntas en archivos

`gaanim.load_questions(path)` lee las preguntas de un `.md` (o `.txt`) o un `.csv`, con la ruta relativa al script que lo llama, y devuelve una lista de `Question`. Un archivo con una pregunta sin respuestas, más de 6 respuestas, una clave desconocida o un valor que no es un número lanza `QuestionError` con el archivo y la línea. El formato está en Preguntas en un archivo.

Campo de `Question`	Qué es
`text`, `options`	La pregunta y sus respuestas.
`correct`	Los índices de las correctas, desde 0; vacío en una encuesta.
`is_quiz`, `multiple`	Si es un cuestionario y si acepta varias respuestas.
`time`, `points`	Segundos para responder (20) y puntos máximos (1000).
`image`	La ruta absoluta de la imagen, o `None`.
`rehearse`	Cómo responde el ensayo: la parte que acierta o un peso por respuesta.
`notes`	Las notas del orador.

`scene.question(q)` abre la pregunta como cuestionario o encuesta según tenga respuestas correctas.

CAPÍTULO

48. Animaciones

Anim, easing, composición, updaters y transiciones entre segmentos

Los handles usan un solo vocabulario. Una llamada directa aplica un corte en el cursor actual sin avanzar el tiempo; la misma llamada bajo la propiedad `animate` describe un `Anim` que solo entra en la línea de tiempo al pasarlo a `scene.play(...)`.

```
dot.move_to(1.25, 1).fill(BLUE)
scene.play([dot.animate.move_to(5, 1).fill(RED)])
```

"Inmediato" no significa modificar un objeto ya compilado: registra un corte reversible en el cursor, y los seeks anteriores conservan el estado anterior. Construir, configurar o descartar un `Anim` no cambia la visibilidad, el estado ni el cursor.

Un `Anim` contiene un solo efecto (`fade_in`, `write`, `indicate`…) o una combinación de destinos de propiedad (`move_to`, `fill`, `opacity`…) que comparten duración, easing y retraso. Encadenar un efecto después de un destino de propiedad o de otro efecto, como `dot.animate.move_to(1, 0).fade_in()`, o un destino después de un efecto, como `dot.animate.fade_in().move_to(1, 0)`, lanza `ValueError`; combina animaciones separadas con `parallel()`. Los modificadores de tiempo y de trazo (`duration`, `easing`, `stroke_width`…) sí se encadenan con cualquier efecto. Cada `Anim` es de un solo uso. `pulse`, `wave`, `highlight`, `focus`, `cancel`, `reveal`, `brace` y `annotate` sobre una selección de texto (`text["parte"].animate.pulse()`) están en Texto.

48.1.1. Destinos de propiedad

Encadena varios destinos en el mismo `Anim`: se interpolan a la vez desde el estado que el objeto tenga al empezar el clip.

```
scene.play([
  circle.animate
    .move_to(2, 0.5)
    .scale_by(1.4)
    .fill(BLUE)
    .stroke(WHITE, 0.06)
    .opacity(0.8)
    .duration(1.5)
    .easing(Easing.SMOOTH)
])
```

Los destinos absolutos de posición, rotación, escala y opacidad aceptan un `ScalarSource`: un número, `Parameter`, `Variable`, `Computed` o `scene.time`. Bajo `.animate`, una fuente se evalúa al inicio efectivo del clip (incluidos retrasos y secuencias) y queda congelada durante la interpolación. En el setter directo, en cambio, la fuente enlaza el canal desde el cursor: otra fuente lo reemplaza y un número lo termina con un corte reversible. Mientras un canal está enlazado, anima la fuente o fija antes el canal con un número; las animaciones que lo escriben lanzan un error.

```
phase = scene.viz.parameter(0.0)
height = computed(lambda x: x * x, inputs=[phase])
dot.move_to(phase, height)                    # enlazado
```

```
scene.play(phase.animate.set(2).duration(2))
dot.move_to(2, 4)                              # fija el canal
scene.play(dot.animate.move_to(other).opacity(0.5))
```

48.1.2. Anim.move_to

MÉTODO `move_to(x: ScalarSource, y: ScalarSource, anchor: Anchor | None = None) -> Anim | move_to(reference: Drawable) -> Anim | move_to(point: AnchorPoint) -> Anim`

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`x, y`	`ScalarSource`	obligatorio	Posición de destino.
`anchor`	`Anchor | str | None`	`None`	Punto del objeto que llega a `(x, y)`; `None` usa el centro, salvo en un `Text`, que llega donde lo pondría `Text.move_to(x, y)`: su primera línea base centrada en el punto, o su centro visual si tiene varias líneas. Un `Anchor` o su nombre, como `"top_left"`.
`reference / point`	`Drawable | AnchorPoint`	obligatorio	Objeto o punto de anclaje de la misma escena, resuelto al inicio del clip; no lo sigue después.

Devuelve: `Anim` — Movimiento a una posición absoluta.

Ejemplo:

```
from gaanim import Easing, Anchor, BLUE, WHITE, Scene
scene = Scene(frame=(16, 9), background="#0f172a")
rect = scene.geometry.rect(1.25, 0.75).fill(BLUE).stroke(WHITE, 0.025).move_to(-1.5, 0)
scene.play([rect.animate.move_to(1, 0,
anchor=Anchor.TOP_RIGHT).duration(0.9).easing(Easing.SMOOTH)])
scene.render()
```

48.1.3. Anim.shift_by

MÉTODO `shift_by(dx: float, dy: float) -> Anim`

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
dx	`float`	obligatorio	Desplazamiento horizontal.
dy	`float`	obligatorio	Desplazamiento vertical.

Devuelve: `Anim` — Movimiento relativo.

Ejemplo:

```
from gaanim import Easing, BLUE, Scene
scene = Scene(frame=(16, 9), background="#0f172a")
circle = scene.geometry.circle(0.5).fill(BLUE).move_to(-1, 0)
scene.play([circle.animate.shift_by(2, 0).duration(1.0).easing(Easing.spring(stiffness=90,
damping=12))])
scene.render()
```

48.1.4. Anim.path_arc

MÉTODO `path_arc(angle: float) -> Anim`

Curva el `move_to` o `shift_by` de este `Anim` en un arco circular en lugar de la línea recta. Sin un destino de traslación lanza `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`angle`	`float`	obligatorio	Giro del arco en radianes; positivo es antihorario.

Devuelve: `Anim` — El mismo movimiento, por un arco.

Ejemplo:

```
scene.play(ball.animate.move_to(4, 0).path_arc(math.pi / 3))
```

48.1.5. Anim.move_along

MÉTODO `move_along(target: Drawable, *, orient: bool = False, rotate_offset: float = 0.0, start: float = 0.0, end: float = 1.0) -> Anim`

Muestrea el contorno por longitud de arco real. Con `.easing(Easing.LINEAR)` la velocidad es uniforme. Con `orient=True` la rotación sigue la tangente, así un avión apunta hacia donde vuela; sin él, rotación y escala no cambian.

Parámetros:

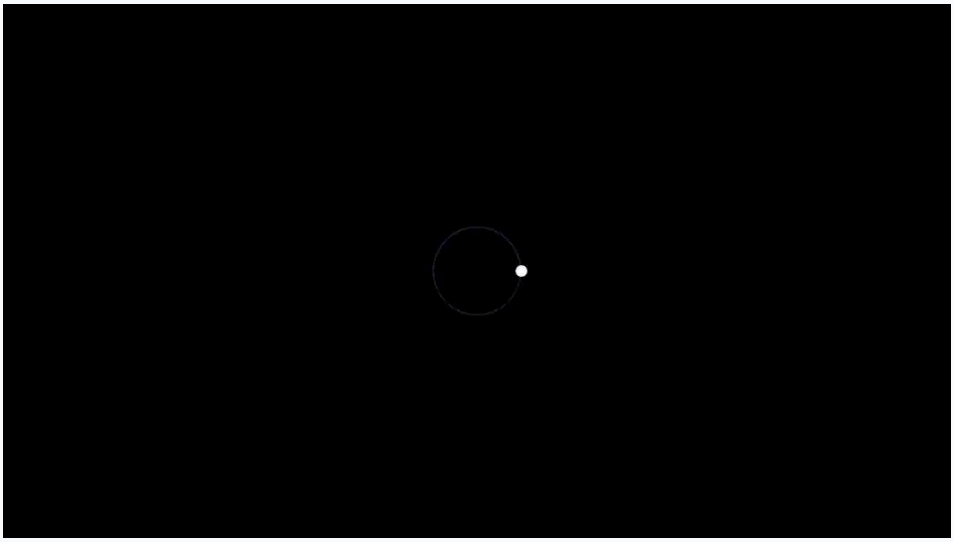
Nombre	Tipo	Predeterminado	Descripción

`target`	`Drawable`	obligatorio	Objeto cuyo contorno se recorre: círculo, rectángulo, curva, polilínea… Se muestrea su geometría de mundo. Una `arrow` o `curved_arrow` sólida se recorre por su eje, de la cola a la punta.
`orient`	`bool`	`False`	Gira el objeto con la tangente del camino.
`rotate_offset`	`float`	`0.0`	Radianes que se suman a la tangente al orientar.
`start, end`	`float`	`0.0, 1.0`	Tramo recorrido, en fracciones de longitud de arco en `[0, 1]`. `start > end` lo recorre al revés; valores iguales o fuera de rango lanzan `ValueError`.

Devuelve: `Anim` — Traslación a lo largo del camino.

Ejemplo:

```
from gaanim import Easing, BLACK, BLUE, WHITE, Scene
scene = Scene(frame=(16, 9), background=BLACK)
circle = scene.geometry.circle(0.75).stroke(BLUE, 0.04).no_fill().move_to(0, 0)
dot = scene.geometry.dot(0.1).fill(WHITE).move_to(0.75, 0)
scene.play([dot.animate.move_along(circle).duration(2.0).easing(Easing.LINEAR)])
scene.render()
```



48.1.6. Anim.scale_to

MÉTODO `scale_to(factor: ScalarSource) -> Anim`

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`factor`	`ScalarSource`	obligatorio	Escala uniforme absoluta.

Devuelve: `Anim` — Cambio de escala.

Ejemplo:

```
scene.play(badge.animate.scale_to(1.5).duration(0.4))
```

48.1.7. Anim.scale_by

MÉTODO `scale_by(factor: float) -> Anim`

Escala alrededor del pivote del objeto; cámbialo con `with_pivot(x, y)`.

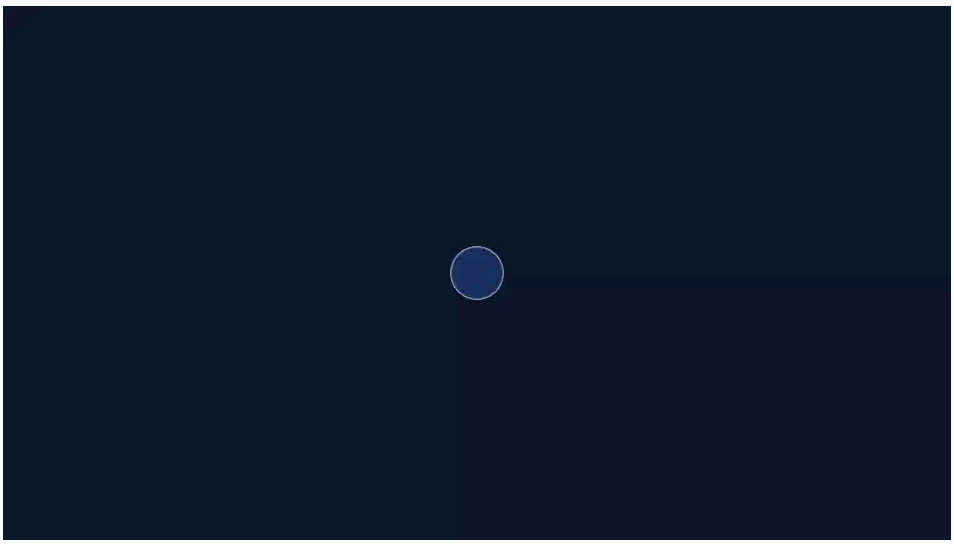
Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`factor`	`float`	obligatorio	Multiplica la escala actual: mayor que 1 agranda y menor que 1 encoge.

Devuelve: `Anim` — Cambio de escala relativo.

Ejemplo:

```
from gaanim import Easing, BLUE, WHITE, Scene
scene = Scene(frame=(16, 9), background="#0f172a")
icon = scene.geometry.circle(0.45).fill(BLUE).stroke(WHITE, 0.025).move_to(0, 0)
scene.play([icon.animate.scale_by(1.8).duration(0.7).easing(Easing.spring(stiffness=90,
damping=12))])
scene.render()
```



48.1.8. Anim.rotate_to

MÉTODO `rotate_to(radians: ScalarSource) -> Anim`

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`radians`	`ScalarSource`	obligatorio	Rotación absoluta alrededor de Z, en radianes.

Devuelve: `Anim` — Giro hasta un ángulo.

Ejemplo:

```
scene.play(needle.animate.rotate_to(1.2))
```

48.1.9. Anim.skew_to

MÉTODO `skew_to(x: float, y: float) -> Anim`

Interpola el sesgo alrededor del pivote, como `Drawable.skew_to`; fija el pivote antes con `with_pivot(x, y)`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`x`	`float`	obligatorio	Corte horizontal de destino, sin unidades.
`y`	`float`	obligatorio	Corte vertical de destino, sin unidades.

Devuelve: `Anim` — Sesgo hasta los factores dados.

Ejemplo:

```
scene.play(frame.animate.skew_to(0.15, 0).duration(0.4))
```

48.1.10. Anim.matrix_to

MÉTODO `matrix_to(matrix: tuple[tuple[float, float], tuple[float, float]]) -> Anim`

Anima hasta una transformación lineal 2D alrededor del pivote, como `Drawable.matrix_to`. Se descompone en giro, sesgo y escala, que se interpolan juntos. Lanza `ValueError` con una matriz no finita o no invertible.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`matrix`	`tuple[tuple[float, float], tuple[float, float]]`	obligatorio	Filas `((a, b), (c, d))` de la transformación

			lineal de destino.

Devuelve: `Anim` — Transformación hasta la matriz dada.

Ejemplo:

```
scene.play(face.animate.matrix_to(((0.87, 0), (-0.5, 1))).duration(0.6))
```

48.1.11. Anim.rotate_by

MÉTODO `rotate_by(radians: float) -> Anim`

Gira alrededor del pivote del objeto: el que fija `with_pivot` o, sin él, el centro de su caja en las figuras declaradas con coordenadas de escena, como `line`, `polygon` o `curved_arrow_arc`. Para una bisagra o una órbita, fija el pivote antes con `with_pivot(x, y)`. Un giro de cualquier tamaño, incluidas varias vueltas, sigue un solo easing durante toda la duración y el pivote queda fijo.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`radians`	`float`	obligatorio	Ángulo relativo en radianes; positivo es antihorario.

Devuelve: `Anim` — Giro relativo.

Ejemplo:

```
from gaanim import BLUE, Scene
from math import pi
scene = Scene(frame=(16, 9), background="#0f172a")
arm = scene.geometry.rect(1, 0.175).fill(BLUE).move_to(0.5, 0).with_pivot(0, 0)
scene.play([arm.animate.rotate_by(pi/2).duration(0.9)])
scene.render()
```

48.1.12. Anim.pivot

MÉTODO `pivot(x: float, y: float) -> Anim`

Hace que el `rotate_by(...)` ya encadenado gire alrededor de `(x, y)` en lugar del pivote del objeto, sin cambiar ese pivote para animaciones posteriores. Llámalo después de `rotate_by`; en animaciones sin giro 2D no tiene efecto. Para fijar el pivote de forma duradera usa `Drawable.with_pivot(x, y)`.

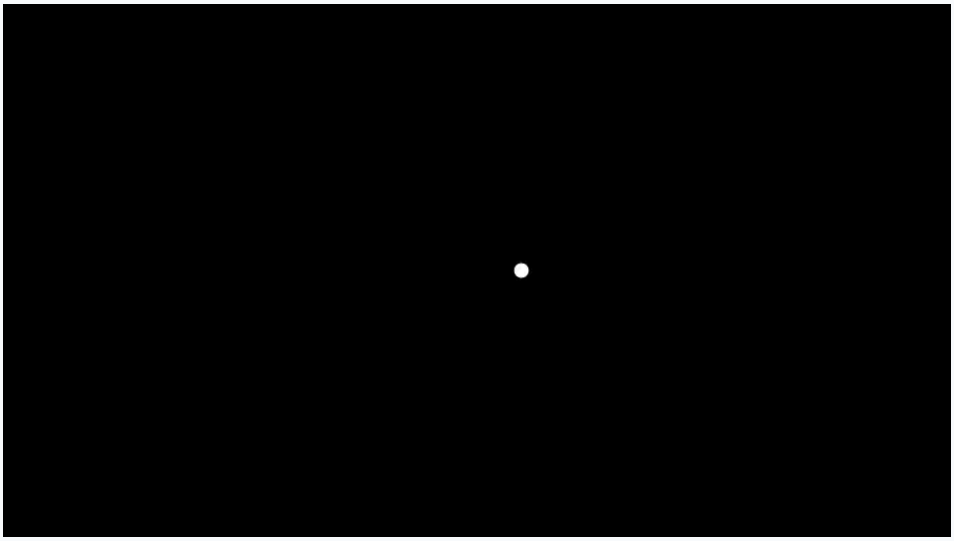
Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`x, y`	`float`	obligatorio	Punto de giro en unidades de escena.

Devuelve: `Anim` — El mismo `Anim`.

Ejemplo:

```
import math
from gaanim import Easing, BLACK, WHITE, Scene
scene = Scene(frame=(16, 9), background=BLACK)
dot = scene.geometry.dot(0.125).fill(WHITE).move_to(0.75, 0)
scene.play([dot.animate.rotate_by(math.tau).pivot(0, 0).duration(1.5).easing(Easing.LINEAR)])
scene.render()
```



48.1.13. Anim.about_point

MÉTODO `about_point(x: float, y: float) -> Anim`

Alias de `Anim.pivot`: `arm.animate.rotate_by(pi / 2).about_point(0, 0)`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`x, y`	`float`	obligatorio	Punto de giro en unidades de escena.

Devuelve: `Anim` — El mismo `Anim`.

48.1.14. Anim.opacity

MÉTODO `opacity(value: ScalarSource) -> Anim`

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`value`	`ScalarSource`	obligatorio	Opacidad de destino, limitada a `[0, 1]`.

Devuelve: `Anim` — Cambio de opacidad.

Ejemplo:

```
scene.play([box.animate.opacity(0.35).duration(0.6)])
```

48.1.15. Anim.fill

MÉTODO `fill(color: Paint) -> Anim`

Los gradientes del mismo tipo interpolan su geometría y sus paradas normalizadas; un color sólido puede pasar a un gradiente y al revés. Tipos de gradiente incompatibles (lineal y radial) lanzan `ValueError`. Tras `no_fill()`, el relleno aparece desde transparente. En un `Text`, cada glifo interpola desde su color actual, así que los fragmentos de color propio convergen al destino. En una `Primitive3D` cambia el color base de su `Material3D` y exige un color sólido. Ver Temas y colores.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`color`	`Paint`	obligatorio	Color o `Brush` de destino.

Devuelve: `Anim` — Cambio de relleno.

Ejemplo:

```
paint = Brush.linear([BLUE, GOLD], start=(-1, 0), end=(1, 0))
scene.play(circle.animate.fill(paint).duration(1.5))
```

48.1.16. Anim.stroke

MÉTODO `stroke(color: Paint, width: float) -> Anim`

La pintura se interpola como en `fill`. Tras `no_stroke()`, el trazo aparece desde transparente y crece desde ancho cero. En un `Text` cambia el contorno de cada glifo sin tocar sus rellenos. No existe en `Primitive3D`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`color`	`Paint`	obligatorio	Color o `Brush` del trazo.
`width`	`float`	obligatorio	Ancho del trazo en unidades lógicas.

Devuelve: `Anim` — Cambio de trazo.

Ejemplo:

```
scene.play(square.animate.stroke(GOLD, 0.08))
```

48.1.17. Anim.stroke_width

MÉTODO `stroke_width(value: float) -> Anim`

En un `Anim` de propiedades anima el ancho del trazo. En `write`, `create`, `uncreate`, `unwrite` o `draw_border_then_fill` fija el ancho del trazo dibujado.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`value`	`float`	obligatorio	Ancho del trazo en unidades lógicas.

Devuelve: `Anim` — El mismo `Anim`.

Ejemplo:

```
scene.play(ring.animate.stroke_width(0.12))
scene.play(ring.animate.uncreate().stroke_width(0.05))
```

48.1.18. Anim.set

MÉTODO `set(value: float) -> Anim`

Anima un `Parameter` o una `Variable`; todo lo que depende de ellos se actualiza en cada fotograma.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`value`	`float`	obligatorio	Valor finito de destino.

Devuelve: `Anim` — Cambio del valor.

Ejemplo:

```
theta = scene.viz.parameter(0.0)
scene.play(theta.animate.set(3.14).duration(2))
```

48.1.19. Anim.crop

MÉTODO `crop(x: float, y: float, width: float, height: float, *, normalized: bool = False) -> Anim`

Anima qué parte de una `Image` o un `Video` se ve dentro de su marco fijo. Un rectángulo inválido o un objeto que no es un medio lanzan `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`x, y, width, height`	`float`	obligatorio	Rectángulo de la fuente, en píxeles desde la esquina superior izquierda.
`normalized`	`bool`	`False`	Interpreta el rectángulo como fracciones del tamaño original.

Devuelve: `Anim` — Recorte animado.

Ejemplo:

```
photo = scene.media.image("assets/cover.png").frame(6, 3.4, fit="cover")
scene.play(photo.animate.crop(0.25, 0.25, 0.5, 0.5, normalized=True).duration(1.2))
```

48.1.20. Anim.fill_level

MÉTODO `fill_level(level: float) -> Anim`

Anima un relleno creado con `scene.geometry.fill_level`. Si el nivel está enlazado a una fuente, lanza `ValueError`: anima la fuente o fija antes un número con `set_fill_level`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`level`	`float`	obligatorio	Nivel normalizado en `[0, 1]`.

Devuelve: `Anim` — Cambio de nivel.

Ejemplo:

```
drop = scene.geometry.circle(1.2).no_fill().stroke(WHITE, 0.04)
water = scene.geometry.fill_level(drop, "#38bdf8", 0.0)
scene.play([water.animate.fill_level(0.72).duration(1.4)])
```

`Anim.custom` sustituye todos los destinos por una función propia; está en Animaciones personalizadas. Los efectos animables `glow`, `blur`, `shadow` y `trim` están en Efectos.

48.1.21. Entradas y salidas

Una entrada programada mantiene oculto el objeto antes de empezar, también si se declaró después de otras animaciones o dentro de un grupo, y durante la transición que abre su segmento. Son entradas:

- por opacidad: `fade_in` y `fade_in_from`;
- por trazo: `create`, `write` y `draw_border_then_fill`;
- por escala desde cero: `grow_from_center`, `grow_from_point`, `grow_from_edge`, `grow_arrow` y `spin_in_from_nothing`;
- de texto: `typewriter` y `reveal`.

Las demás animaciones (`move_to`, `scale_to`, `indicate`…) parten del estado visible del objeto: si no debe verse antes, empieza con una entrada o dale `opacity(0)` y anímala con `animate.opacity(1)`. Las salidas (`fade_out`, `uncreate`, `unwrite`, `shrink_to_center`) lo dejan oculto al terminar.

48.1.22. Anim.fade_in

MÉTODO `fade_in() -> Anim`

Lleva la opacidad de 0 a la que el objeto tenía declarada o fijada, o a la que tenía antes de un `fade_out`: `opacity(0.3).hidden()` termina en 0.3. Un objeto declarado con `opacity(0)` termina en 1.

Devuelve: `Anim` — Aparición por opacidad.

Ejemplo:

```
from gaanim import BLUE, GOLD, WHITE, Scene
scene = Scene(frame=(16, 9), background="#0f172a")
title = scene.text("Fade").fill(WHITE).move_to(0, 0.5)
box = scene.geometry.rect(1.5, 0.625).fill(BLUE).move_to(0, -0.5)
scene.play([title.animate.fade_in().duration(0.5)])
scene.play([box.animate.opacity(0.35).duration(0.6)])
scene.play([title.animate.fade_out().duration(0.4)])
scene.render()
```

48.1.23. Anim.fade_out

MÉTODO `fade_out() -> Anim`

Devuelve: `Anim` — Desaparición por opacidad.

48.1.24. Anim.fade_in_from

MÉTODO `fade_in_from(direction: Direction, distance: float = 0.48) -> Anim`

Empieza invisible y desplazado, y llega a su sitio mientras aparece.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`direction`	`Direction`	obligatorio	Lado desde el que entra: `Direction.UP`, `DOWN`, `LEFT`, `RIGHT` o una diagonal.
`distance`	`float`	`0.48`	Distancia recorrida hasta su posición.

Devuelve: `Anim` — Entrada con desplazamiento.

Ejemplo:

```python
from gaanim import Direction, WHITE, Scene
scene = Scene(frame=(16, 9), background="#0f172a")
caption = scene.text("Enters from below").fill(WHITE).move_to(0, 0)
scene.play([caption.animate.fade_in_from(Direction.DOWN, distance=0.6).duration(0.8)])
scene.render()
```

48.1.25. Anim.write

MÉTODO `write(*, by: Literal["grapheme", "word", "line", "part"] = "grapheme", order: Literal["forward", "reverse", "center", "random"] = "forward", stagger: Optional[float] = None, brush: Literal["pen", "chalk"] = "pen", seed: int = 0) -> Anim`

Dibuja los contornos y luego los rellena, como el `Write` de Manim. Cada pieza (un glifo, una forma, un miembro de un grupo) ocupa su parte de la animación, escalonada por `stagger`, y la curva (lineal por defecto) se aplica al avance de cada pieza. En la primera mitad de su parte la pieza traza su contorno, un tramo de Bézier tras otro y sus contornos en orden, centrado y de un 0,25 % del alto del cuadro, en el color de su trazo (el de su relleno si no tiene). En la segunda mitad el contorno pasa a su aspecto propio: aparece el relleno y el trazo toma su ancho, o desaparece. Sin duración, dura 1 s con menos de 15 piezas y 2 s con más; `scene.play(duration=...)` la sigue fijando. Sin `stagger`, el retardo entre piezas es el de Manim, `min(4 / piezas, 0.2)`; agrupada con `by` por palabras, líneas o partes es `min(4 / grupos, 1)`, así que hasta cuatro grupos se escriben uno tras otro. Los descendientes reactivos siguen ocultos hasta la animación y conservan el progreso al regenerarse. La segmentación es la de `text.words`, `text.lines` y `text.parts`. Después de un destino de propiedad o de otro efecto lanza `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
by	`"grapheme" | "word" | "line" | "part"`	`"grapheme"`	En un texto, qué unidades

			empiezan juntas: grafemas, palabras, líneas explícitas o partes semánticas. La puntuación se une a su vecina.
`order`	`"forward" | "reverse" | "center" | "random"`	`"forward"`	Orden de los grupos: hacia delante, al revés, desde el centro hacia fuera o en una permutación aleatoria que fija `seed`.
`stagger`	`float | None`	`None`	Retardo relativo entre grupos; `None` lo adapta al número de grupos.
`brush`	`"pen" | "chalk"`	`"pen"`	`"chalk"` dibuja el objeto como tiza en una pizarra durante toda la escena,

			como `Drawable.chalk(seed)`.
`seed`	`int`	`0`	Semilla del orden `"random"` y del temblor y el grano de la tiza.

Devuelve: `Anim` — Escritura trazo a trazo.

Ejemplo:

```
from gaanim import Scene, part
scene = Scene(frame=(16, 9), background="#0f172a")
# Grouping is resolved by the specialized Text API.
eq = scene.text("$", part("energy", "E"), " = ", part("mass", "m"), " ", part("speed", "c^2"),
"$").move_to(0, 0)
scene.play([eq.animate.write(by="part", stagger=0.08).duration(1.4)])
scene.render()
```

Con `brush="chalk"` la fórmula se escribe como en una pizarra:

```
from gaanim import WHITE, Scene
scene = Scene(frame=(16, 9), background="#1f3a2e")
eq = scene.text.equation("E = m c^2").fill(WHITE).scale_to(2)
scene.play([eq.animate.write(brush="chalk", seed=3).duration(1.5)])
scene.render()
```

48.1.26. Anim.unwrite

MÉTODO `unwrite() -> Anim`

Borra como el `Unwrite` de Manim: `write` al revés, desde la última pieza. Cada pieza vuelve primero a su contorno y luego el contorno se recoge hasta su inicio. Dura lo que duraría `write`.

Devuelve: `Anim` — La escritura al revés.

Ejemplo:

```
from gaanim import GOLD, Scene
scene = Scene(frame=(16, 9), background="#0f172a")
formula = scene.text("$E = m c^2$").fill(GOLD).move_to(0, 0)
scene.play([formula.animate.write().duration(1.0)])
scene.play([formula.animate.unwrite().duration(0.7)])
scene.render()
```

48.1.27. Anim.create

MÉTODO `create() -> Anim`

Dibuja el contorno durante el primer 70 % de la duración y funde el relleno de las formas cerradas en el 30 % final. El ancho del trazo no cambia; si el objeto no tiene contorno, usa uno temporal de 0.03 unidades. Los caminos abiertos usan toda la duración para el trazo. El easing predeterminado es `Easing.DOUBLE_SMOOTH`. En una malla 3D, crece desde el centro mientras aparece. A diferencia de `write`, no sigue el orden de los glifos.

Devuelve: `Anim` — Trazado progresivo seguido del relleno.

Ejemplo:

```
from gaanim import Easing, BLUE, WHITE, Scene
scene = Scene(frame=(16, 9), background="#0f172a")
circle = scene.geometry.circle(1).fill(BLUE).stroke(WHITE, 0.04).move_to(0, 0)
scene.play([circle.animate.create().duration(1.0).easing(Easing.SMOOTH)])
scene.play([circle.animate.uncreate().duration(0.6)])
scene.render()
```

48.1.28. Anim.uncreate

MÉTODO `uncreate() -> Anim`

Devuelve: `Anim` — El trazado al revés: borra el objeto.

48.1.29. Anim.draw_border_then_fill

MÉTODO `draw_border_then_fill() -> Anim`

Dibuja el borde y luego inunda el interior; queda bien en formas rellenas.

Devuelve: `Anim` — Primero el contorno, después el relleno.

Ejemplo:

```python
from gaanim import BLUE, WHITE, Scene
scene = Scene(frame=(16, 9), background="#0f172a")
rect = scene.geometry.rect(1.75, 1).fill(BLUE).stroke(WHITE, 0.04).move_to(0, 0)
scene.play([rect.animate.draw_border_then_fill().duration(1.3)])
scene.render()
```

48.1.30. Anim.with_pen_tip

MÉTODO `with_pen_tip() -> Anim`

En `write`, `create` y `draw_border_then_fill`, dibuja una punta de pluma en el extremo de cada trazo mientras se traza, como una escritura a mano: un punto del color del trazo (o del relleno, si la forma no tiene trazo) que desaparece cuando el trazo está completo. Un trazo con puntas propias (`tip(...)`) las conserva; las demás animaciones lo ignoran.

Devuelve: `Anim` — El mismo `Anim`.

Ejemplo:

```python
from gaanim import WHITE, Scene
scene = Scene(frame=(16, 9), background="#0f172a")
path = scene.geometry.path([(-1.5, 0), (0, 0.5), (1.5, 0)]).no_fill().stroke(WHITE, 0.04)
scene.play([path.animate.write().with_pen_tip().duration(1.4)])
scene.render()
```

48.1.31. Anim.grow_from_center

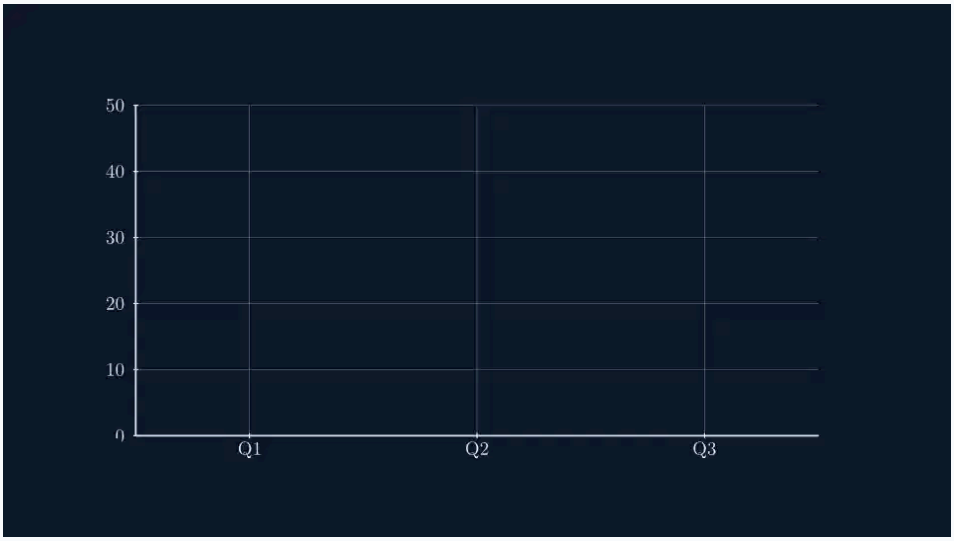
MÉTODO `grow_from_center() -> Anim`

Aparición desde el centro de la caja; útil para gráficas e insignias.

Devuelve: `Anim` — Crecimiento desde escala cero.

Ejemplo:

```
# Visualization uses immutable ChartSpec batches.
from gaanim import Easing, Axis, BLUE, ChartSpec, Scene
scene = Scene(frame=(16, 9), background="#0f172a")
spec = ChartSpec({"x": [0, 1, 2], "value": [18, 42, 31]}) \
  .mark("bar").encode(x="x", y="value") \
  .axes(x=Axis.category(["Q1", "Q2", "Q3"]), y=Axis.linear(0, 50))
chart = scene.viz.chart(spec)
scene.play([chart.layer("marks").animate.grow_from_center().duration(0.7).easing(Easing.spring(stiffness=90,
damping=12))])
scene.play([chart.layer("marks").animate.shrink_to_center().duration(0.5)])
scene.render()
```



48.1.32. Anim.shrink_to_center

MÉTODO `shrink_to_center() -> Anim`

Devuelve: `Anim` — Reducción hasta escala cero hacia el centro.

48.1.33. Anim.grow_from_edge

MÉTODO `grow_from_edge(direction: Direction) -> Anim`

Usa la caja en su posición final: `Direction.DOWN` fija el punto medio del borde inferior, así una barra sube desde su base, y `Direction.custom(x, y)` fija el punto correspondiente de la caja. Termina en la posición y el tamaño declarados. Easing predeterminado: `Easing.SMOOTH`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`direction`	`Direction`	obligatorio	Lado de la caja que queda fijo; una diagonal fija una esquina.

Devuelve: `Anim` — Crecimiento desde un borde.

Ejemplo:

```
from gaanim import BLUE, CYAN, GOLD, Direction, Scene
scene = Scene(frame=(16, 9), background="#0f172a")
bars = [scene.geometry.rect(1.2, h).fill(c).move_to(x, h / 2 - 2) for x, h, c in [(-3, 2.5, BLUE),
(-1.4, 4, CYAN), (0.2, 3, BLUE)]]
badge = scene.geometry.circle(0.9).fill(GOLD).move_to(4, 1.5)
scene.play([bar.animate.grow_from_edge(Direction.DOWN).duration(0.8) for bar in bars])
scene.play([badge.animate.grow_from_point(2.5, 0).duration(0.6)])
scene.render()
```

48.1.34. Anim.grow_from_point

MÉTODO `grow_from_point(x: float, y: float) -> Anim`

Como `grow_from_center`, pero con un punto fijo arbitrario, por ejemplo el origen de un globo. Coordenadas no finitas lanzan `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`x, y`	`float`	obligatorio	Punto de la escena que queda fijo.

Devuelve: `Anim` — Crecimiento desde un punto.

48.1.35. Anim.grow_arrow

MÉTODO `grow_arrow() -> Anim`

La cola queda fija y la punta recorre el eje de la flecha, recto o curvo en `curved_arrow` y `curved_arrow_arc`. La cabeza aparece con sus proporciones durante la primera longitud de cabeza y luego conserva su tamaño mientras el cuerpo se alarga; el grosor no cambia nunca. Un `scene.geometry.connector` crece por su polilínea viva, pasando por cada punto `via`, mientras sus extremos siguen a sus referencias. Cualquier otro objeto, o una flecha deformada por una transformación, usa `create()`; en un trazo con `tip(...)` la punta viaja con el extremo que crece. Easing predeterminado: `Easing.SMOOTH`. En una selección de texto lanza `TypeError`.

Devuelve: `Anim` — Crecimiento de una flecha desde la cola.

Ejemplo:

```
from gaanim import Anchor, CYAN, GOLD, Scene
scene = Scene(frame=(16, 9), background="#0f172a")
straight = scene.geometry.arrow(-5, 1, 0, 1, head_length=0.4, head_width=0.34,
```

```
body_width=0.08).fill(CYAN).stroke(CYAN, 0.02)
curved = scene.geometry.curved_arrow(-5, -1.5, 0, -1.5, 1.2, head_length=0.4, head_width=0.34,
body_width=0.08).fill(GOLD).stroke(GOLD, 0.02)
box = scene.geometry.rect(1.6, 1.0).no_fill().stroke(CYAN, 0.03).move_to(4.5, 2.2)
link = scene.geometry.connector((1.5, 0.6), box.anchor_point(Anchor.BOTTOM), via=[(4.5,
0.6)]).fill(GOLD)
scene.play([straight.animate.grow_arrow().duration(1.2),
curved.animate.grow_arrow().duration(1.2), link.animate.grow_arrow().duration(1.2)])
scene.render()
```

48.1.36. Anim.spin_in_from_nothing

MÉTODO `spin_in_from_nothing() -> Anim`

Una entrada juguetona para estrellas e iconos.

Devuelve: `Anim` — Entrada girando desde escala cero.

Ejemplo:

```
from gaanim import Easing, GOLD, Scene
scene = Scene(frame=(16, 9), background="#0f172a")
star = scene.geometry.star(5, 0.69, 0.325).fill(GOLD).move_to(0, 0)
scene.play([star.animate.spin_in_from_nothing().duration(0.9).easing(Easing.spring(stiffness=90,
damping=12))])
scene.render()
```

48.1.37. Anim.show_passing_flash

MÉTODO `show_passing_flash(*, time_width: float = 0.2) -> Anim`

Como con `create()`, el objeto queda oculto antes del destello (salvo que una animación de trazo anterior lo muestre) y vuelve a ocultarse cuando la ventana sale por el final. Para un pulso sobre una línea visible, dibuja una segunda línea encima.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`time_width`	`float`	`0.2`	Longitud de la ventana visible, como fracción del camino, en `(0, 1]`.

Devuelve: `Anim` — Una ventana que recorre el trazo.

Ejemplo:

```
pulse = scene.geometry.line(-5, 0, 5, 0).stroke(CYAN, 0.06)
scene.wait(0.5)                                              # oculta
scene.play(pulse.animate.show_passing_flash(time_width=0.3).duration(1.0))
scene.wait(0.5)                                              # oculta
```

Las animaciones de trazo de un mismo objeto (`create`, `write`, `trim`, `show_passing_flash`) comparten el canal `effect`, así que no pueden solaparse dentro de un `play`. El error indica las dos animaciones, sus tramos y el tipo de objeto; encadénalas con `sequence` o combínalas en una sola. Dos tramos que solo se tocan no se consideran solapados.

48.1.38. Énfasis

48.1.39. Anim.indicate

MÉTODO `indicate(*, color: Optional[ColorLike] = None, scale_factor: float = 1.2) -> Anim`

Crece desde su centro visual la primera mitad y vuelve la segunda, mientras se tiñe de `color` y recupera su color; el objeto termina en su estado.

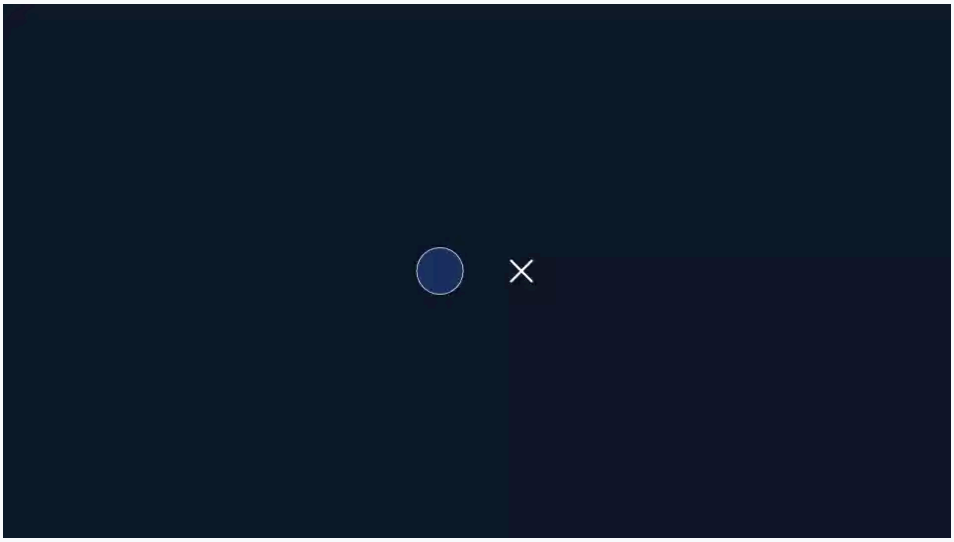
Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`color`	`ColorLike | None`	`None`	Tinte de los rellenos y trazos sólidos; sin él, el `"accent"` del tema, o su `"foreground"` si el objeto ya tiene el color del acento.
`scale_factor`	`float`	`1.2`	Cuánto crece. Debe ser positivo.

Devuelve: `Anim` — Un salto con resaltado.

Ejemplo:

```
from gaanim import BLUE, WHITE, Scene
scene = Scene(frame=(16, 9), background="#0f172a")
key = scene.geometry.circle(0.4).fill(BLUE).stroke(WHITE, 0.025).move_to(-0.625, 0)
wrong = scene.geometry.cross(0.35).stroke(WHITE, 0.04).move_to(0.75, 0)
scene.play([key.animate.indicate().duration(0.7)])
scene.play([wrong.animate.wiggle().duration(0.5)])
scene.render()
```



48.1.40. Anim.wiggle

MÉTODO `wiggle(amplitude: float = 0.1, count: int = 3) -> Anim`

Sacude el objeto de lado a lado y lo deja donde estaba. Útil para señalar una respuesta incorrecta.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción

`amplitude`	`float`	`0.1`	Desplazamiento a cada lado, en unidades de escena. No negativo.
`count`	`int`	`3`	Vaivenes a derecha e izquierda. Al menos 1.

Devuelve: `Anim` — Una sacudida breve.

48.1.41. Anim.circumscribe

MÉTODO `circumscribe(*, color: Optional[ColorLike] = None, width: Optional[float] = None, padding: Optional[Padding] = None, corner_radius: float = 0.08) -> Anim`

Dibuja un marco vivo alrededor del objeto la primera mitad y lo borra desde su inicio la segunda, como el `Circumscribe` de Manim: es `flash_around(time_width=1.0)`. Fuera de la animación el marco es invisible. El `Anim` se reproduce sobre el marco, no sobre el objeto.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`color`	`ColorLike | None`	`None`	Color del marco; sin él, el trazo del tema.
`width`	`float | None`	`None`	Grosor del marco; necesita `color`.
`padding`	`float | Padding | None`	`None`	Separación del marco a los límites; 0,12 por defecto.
`corner_radius`	`float`	`0.08`	Radio de las esquinas.

Devuelve: `Anim` — Un contorno que rodea el objeto.

Ejemplo:

```
scene.play(result.animate.circumscribe().duration(1.0))
```

48.1.42. Anim.flash

MÉTODO `flash(*, color: Optional[ColorLike] = None, width: Optional[float] = None, lines: int = 12, length: float = 0.3, gap: float = 0.15) -> Anim`

Los rayos son un objeto vivo nuevo, repartidos desde la derecha y que siguen al objeto si se mueve. Cada uno se dibuja hacia fuera y luego se va desde su extremo interior; fuera de la animación son invisibles. El `Anim` se reproduce sobre los rayos, no sobre el objeto.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`color`	`ColorLike | None`	`None`	Color de los rayos; sin él, el trazo del tema.

`width`	`float | None`	`None`	Grosor de los rayos; necesita `color`.
`lines`	`int`	`12`	Número de rayos, de 1 a 64.
`length`	`float`	`0.3`	Largo de cada rayo. Positivo.
`gap`	`float`	`0.15`	Distancia entre el círculo que contiene los límites del objeto y el inicio de los rayos. No negativa.

Devuelve: `Anim` — Un destello de líneas alrededor del objeto.

Ejemplo:

```
scene.play(star.animate.flash().duration(0.6))
```

48.1.43. Anim.blink

MÉTODO `blink(count=3) -> Anim`

Cada parpadeo es un fundido rápido a nada y de vuelta, y dura medio segundo: la duración por defecto es `0.5 * count`, y `duration` fija el ciclo completo. Al final el objeto vuelve a verse.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`count`	`int`	`3`	Cuántas veces parpadea; menos de 1 lanza `ValueError`.

Devuelve: `Anim` — Apaga y enciende el objeto `count` veces.

Ejemplo:

```
scene.play(caret.animate.blink(3))
```

48.1.44. Anim.broadcast

MÉTODO `broadcast(count=4, max_scale=3.0, lag=0.2) -> Anim`

Cada onda es una copia del objeto, colocada donde está cuando empieza la animación, que crece hasta `max_scale` mientras se desvanece. `duration` cubre todas las ondas. El objeto no cambia y las copias son invisibles fuera de su onda. Valores fuera de rango lanzan `ValueError`.

Parámetros:

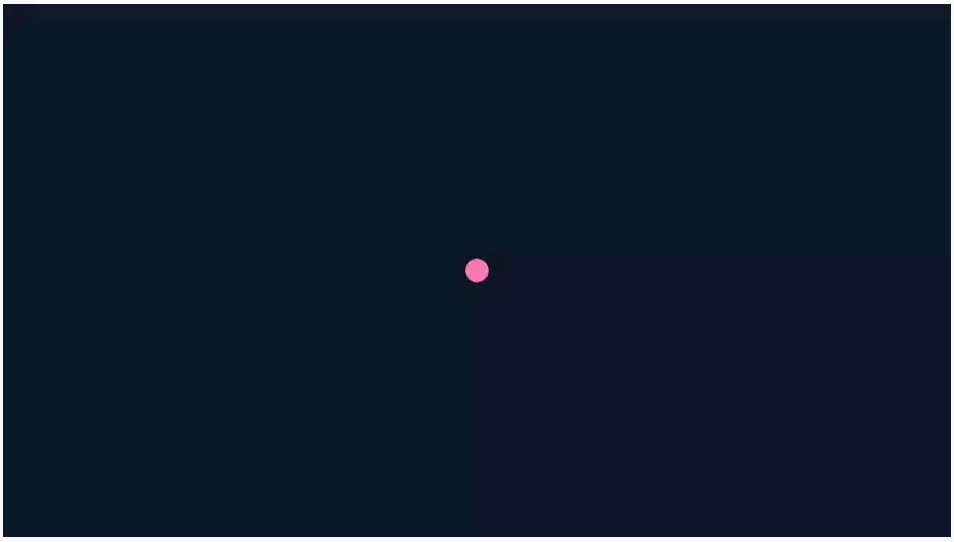
Nombre	Tipo	Predeterminado	Descripción
`count`	`int`	`4`	Ondas, de 1 a 64.
`max_scale`	`float`	`3.0`	Tamaño final de cada onda respecto al objeto; al menos 1.

`lag`	`float`	`0.2`	Cuánto tarda cada onda en salir tras la anterior, como fracción de una onda.

Devuelve: `Anim` — Ondas concéntricas que salen del objeto.

Ejemplo:

```
from gaanim import PINK, Scene

scene = Scene(frame=(16, 9), background="#0f172a")
pin = scene.geometry.dot(0.2).fill(PINK)
scene.play([pin.animate.broadcast(count=4, max_scale=4.0, lag=0.25).duration(2.0)])
scene.render()
```



48.1.45. Anim.flash_around

MÉTODO `flash_around(*, color=None, width=None, padding=None, corner_radius=0.08, time_width=0.4) -> Anim`

El marco es un contorno vivo nuevo (como `Geometry.surrounding_rect`) que sigue los límites del objeto y es invisible fuera del destello. La animación devuelta se reproduce sobre el marco. Sirve también sobre un `TextSelection`: `texto["palabra"].animate.flash_around()`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`color, width`	`ColorLike | None, float | None`	`None`	Reestilizan el marco; un ancho necesita un color. Sin ellos usa el trazo del tema.
`padding`	`Padding | None`	`0.12`	Separación entre el objeto y el marco.

`corner_radius`	`float`	`0.08`	Radio de las esquinas.
`time_width`	`float`	`0.4`	Longitud de la ventana brillante, como fracción del contorno, en `(0, 1]`.

Devuelve: `Anim` — Una ventana brillante que recorre un marco alrededor del objeto.

Ejemplo:

```
scene.play(line["importante"].animate.flash_around(color=PINK, width=0.04).duration(1.2))
```

48.1.46. Anim.flash_under

MÉTODO `flash_under(*, color=None, width=None, gap=0.06, overhang=0.04, time_width=0.5) -> Anim`

La línea es un subrayado vivo nuevo, tan ancho como el objeto, invisible fuera del destello. Valores fuera de rango lanzan `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`color, width`	`ColorLike | None, float | None`	`None`	Como en `flash_around`.
`gap`	`float`	`0.06`	Distancia de la línea bajo el objeto.
`overhang`	`float`	`0.04`	Cuánto sobresale la línea a cada lado.
`time_width`	`float`	`0.5`	Longitud de la ventana brillante, en `(0, 1]`.

Devuelve: `Anim` — Una ventana brillante que recorre una línea bajo el objeto.

Ejemplo:

```
scene.play(line["resalta"].animate.flash_under(color=GOLD, width=0.05).duration(1.0))
```

48.1.47. Fx.spotlight

MÉTODO `scene.fx.spotlight(target, *, dim=0.7, padding=None, corner_radius=0.1) -> Anim`

Un velo negro con un hueco vivo alrededor del objetivo (sigue al objetivo si se mueve) llega a `dim` a mitad de la animación y vuelve a nada; `duration` fija el ciclo entero y una curva como `Easing.SMOOTH` lo deja oscurecido al final. Con varios objetivos hay un hueco alrededor de cada uno, y los que se solapan se unen; para iluminar la caja conjunta, agrúpalos. La animación devuelta se reproduce sobre el velo. Una secuencia vacía, un `dim` fuera de rango o un objetivo de otra escena lanzan `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`target`	`Drawable | TextSelection | Sequence`	obligatorio	Lo que queda iluminado. Con una secuencia, cada objetivo tiene su hueco.
`dim`	`float`	`0.7`	Opacidad máxima del velo negro, en `(0, 1]`.
`padding`	`Padding | None`	`0.2`	Separación entre el objetivo y el borde del hueco.
`corner_radius`	`float`	`0.1`	Radio de las esquinas del hueco.

Devuelve: `Anim` — Un velo con un hueco que se oscurece y vuelve.

Ejemplo:

```
from gaanim import BLUE, WHITE, Scene

scene = Scene(frame=(16, 9), background="#0f172a")
title = scene.text("Un título", role="title").fill(WHITE).move_to(0, 2.4)
card = scene.geometry.rounded_rect(4, 2.2, 0.25).fill("#1e293b").stroke(BLUE, 0.03).move_to(0,
-0.6)
scene.play([scene.fx.spotlight(card, dim=0.8).duration(2.0)])
scene.render()
```

48.1.48. Drawable.animated_boundary

MÉTODO `animated_boundary(colors, *, cycle_rate=0.5, width=0.04, padding=None, corner_radius=0.08) -> Drawable`

El marco es un objeto nuevo, visible desde el principio, que sigue los límites del objeto; anímalo como cualquier otro (`fade_in`, `fade_out`) para que aparezca o desaparezca. El color es función pura del tiempo, así que los seeks y la exportación coinciden con la reproducción. Menos de dos colores, un `width` no positivo o un `cycle_rate` no finito lanzan `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`colors`	`Sequence[ColorLike]`	obligatorio	Al menos dos colores; el último se mezcla de nuevo con el primero.
`cycle_rate`	`float`	`0.5`	Vueltas a la lista de colores por segundo, desde el cursor de la línea de tiempo.
`width`	`float`	`0.04`	Ancho del trazo, en unidades de escena.
`padding`	`Padding | None`	`0.12`	Separación entre el objeto y el marco.
`corner_radius`	`float`	`0.08`	Radio de las esquinas.

Devuelve: `Drawable` — Un marco vivo con el trazo que cambia de color.

Ejemplo:

```
border = card.animated_boundary([BLUE, PURPLE, CYAN], cycle_rate=0.5)
scene.wait(2)
```

48.1.49. Texto

Estas animaciones solo existen sobre un `Text` completo; en otro objeto lanzan `TypeError`. Se evalúan de forma nativa a partir del tiempo del clip, así que los seeks, las capturas y la exportación coinciden. El `Text` que escriben queda oculto hasta que empiezan, así las filas de un `stagger` esperan sin verse. Las selecciones, anotaciones y transiciones de fórmulas están en Texto.

48.1.50. Anim.typewriter

MÉTODO `typewriter(cps: float = 18.0, cursor: Optional[str] = "▌", blink: float = 2.0, jitter: float = 0.2, seed: int = 0, keep_cursor: bool = True) -> Anim`

Vacía el `Text` y lo vuelve a escribir un grafema por pulsación, con el diseño final: los glifos aparecen en su sitio y nada se reacomoda. El cursor sigue al último grafema, también entre líneas. Sin duración explícita dura hasta la última pulsación (unos `grafemas / cps` segundos); `.duration(...)` reescala el ritmo. Valores inválidos lanzan `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`cps`	`float`	`18.0`	Pulsaciones por segundo; positivo.
`cursor`	`str | None`	`"▌"`	Cadena dibujada tras el último grafema; `None` o `""` para ninguno. Los caracteres de bloque (de▏ a ▉) son rectángulos exactos, independientes de la fuente.
`blink`	`float`	`2.0`	Parpadeos por segundo del cursor en reposo; `0` lo deja fijo. Mientras escribe está fijo.
`jitter`	`float`	`0.2`	Cada intervalo se multiplica por un factor en `[1 - jitter, 1 + jitter]`; debe estar en `[0, 1)`.
`seed`	`int`	`0`	Semilla del ritmo de pulsaciones.
`keep_cursor`	`bool`	`True`	Mantiene el cursor al terminar; `False` lo quita.

Devuelve: `Anim` — Escritura tecla a tecla, con tiempo lineal.

Ejemplo:

```
from gaanim import CYAN, Scene
scene = Scene(frame=(16, 9), background="#0f172a")
prompt = scene.text("gaanim render", role="code").fill(CYAN).move_to(-3, 0)
scene.play([prompt.animate.typewriter(cps=18, cursor="▌")])
scene.wait(0.4)
scene.play([prompt.animate.backspace(6)])
scene.play([prompt.animate.retype("gaanim export --from clímax")])
scene.wait(0.6)
scene.render()
```

48.1.51. Anim.backspace

MÉTODO `backspace(count: Optional[int] = None, cps: float = 24.0) -> Anim`

El cursor retrocede con cada borrado; un `Text` sin cursor recibe el parpadeante predeterminado. Dura `count / cps` por defecto.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`count`	`int | None`	`None`	Grafemas que se borran desde el final, como máximo los visibles; `None` borra todos.
`cps`	`float`	`24.0`	Borrados por segundo; positivo.

Devuelve: `Anim` — Borrado tecla a tecla.

48.1.52. Anim.retype

MÉTODO `retype(text: str, cps: float = 18.0, jitter: float = 0.2, seed: int = 0) -> Anim`

Conserva el prefijo que comparte con el texto visible, borra lo demás y escribe el resto con el estilo y el origen del `Text`. Después el `Text` muestra `text`, pero su contenido declarado (tamaño de layout, selecciones `text[...]`) no cambia.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`text`	`str`	obligatorio	Texto plano nuevo, no vacío.
`cps, jitter, seed`	`float, float, int`	`18.0, 0.2, 0`	Como en `typewriter`.

Devuelve: `Anim` — Borrado hasta el prefijo común y escritura del resto.

48.1.53. Anim.scramble

MÉTODO `scramble(charset: str = "upper", reveal_delay: float = 0.3, speed: float = 20.0, seed: int = 0) -> Anim`

Cada grafema muestra glifos aleatorios del juego de caracteres, centrados en su celda final, y después se asienta de izquierda a derecha. El ancho del texto final queda reservado, así que no salta. Dura `reveal_delay` más 0.05 s por grafema (al menos 0.6 s).

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`charset`	`str`	`"upper"`	`"upper"`, `"lower"`, `"digits"`, `"hex"`, `"symbols"` o una cadena literal como `"01"`.
`reveal_delay`	`float`	`0.3`	Segundos que todas las posiciones cambian antes de que se asiente la primera.
`speed`	`float`	`20.0`	Cambios de glifo por segundo; positivo.
`seed`	`int`	`0`	Semilla de la elección de glifos.

Devuelve: `Anim` — Revelado por decodificación, con tiempo lineal.

Ejemplo:

```
from gaanim import GOLD, Scene
scene = Scene(frame=(16, 9), background="#0f172a")
label = scene.text("LAUNCH SEQUENCE", role="title").fill(GOLD).move_to(0, 0)
scene.play([label.animate.scramble(charset="upper", reveal_delay=0.3, speed=20)])
scene.wait(0.4)
scene.play([label.animate.scramble_to("LANZAMIENTO", charset="01")])
scene.render()
```

48.1.54. Anim.scramble_to

MÉTODO `scramble_to(text: str, charset: str = "upper", reveal_delay: float = 0.3, speed: float = 20.0, seed: int = 0) -> Anim`

Los glifos actuales desaparecen al empezar y cada posición de `text` se decodifica en el diseño de `text`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`text`	`str`	obligatorio	Texto plano nuevo, no vacío.
`charset, reveal_delay, speed, seed`			Como en `scramble`.

Devuelve: `Anim` — Decodificación hacia un texto nuevo.

48.1.55. Anim.reveal

MÉTODO `reveal(style: Optional[Literal["slide_up", "slide_down", "fade", "scale", "blur"]] = None, *, by: Literal["grapheme", "word", "line", "part"] = "line", mask: bool = True, stagger: float = 0.06) -> Anim`

Con `"slide_up"` cada unidad sube una altura de fila desde detrás de su máscara, así que también funciona en la exportación SVG. Sin máscara, los deslizamientos recorren menos y aparecen por opacidad. `.easing(...)` suaviza cada unidad (cúbica de salida por defecto) y la duración cubre toda la cascada. Ver Texto para la versión sobre selecciones.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`style`	`"slide_up" | "slide_down" | "fade" | "scale" | "blur" | None`	`None`	Estilo de entrada; `None` es `"slide_up"`.
`by`	`"grapheme" | "word" | "line" | "part"`	`"line"`	Unidad que entra de una vez.
`mask`	`bool`	`True`	Los estilos de deslizamiento salen de detrás de una

			máscara vectorial recortada a su fila.
`stagger`	`float`	`0.06`	Segundos entre unidades.

Devuelve: `Anim` — Revelado unidad a unidad.

Ejemplo:

```
title = scene.text("Una idea\npor línea", role="title")
scene.play(title.animate.reveal(by="line", style="slide_up", stagger=0.06))
scene.play(title.animate.conceal(by="line", style="slide_up"))
```

48.1.56. Anim.conceal

MÉTODO `conceal(style: Literal["slide_up", "slide_down", "fade", "scale", "blur"] = "slide_up", *, by: Literal["grapheme", "word", "line", "part"] = "line", mask: bool = True, stagger: float = 0.06) -> Anim`

La salida que corresponde a `reveal`: las unidades se van en orden de lectura desde el reposo y quedan ocultas. Los deslizamientos conservan la dirección del revelado. Aceleran al salir (cúbica de entrada por defecto).

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`style, by, mask, stagger`		`"slide_up", "line", True, 0.06`	Como en `reveal`.

Devuelve: `Anim` — Salida unidad a unidad.

48.1.57. Anim.blur_in

MÉTODO `blur_in(sigma: float = 0.3, *, by: Literal["grapheme", "word", "line", "part"] = "grapheme", stagger: float = 0.02) -> Anim`

Cada unidad pierde el desenfoque y aparece con una curva de salida. Un `sigma` o `stagger` negativo lanza `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`sigma`	`float`	`0.3`	Desenfoque inicial en

			unidades de escena.
`by`	`"grapheme" | "word" | "line" | "part"`	`"grapheme"`	Unidad que entra de una vez.
`stagger`	`float`	`0.02`	Segundos entre unidades.

Devuelve: `Anim` — Entrada desde un desenfoque transparente.

Ejemplo:

```
title = scene.text("Enfoque", role="title")
scene.play(title.animate.blur_in(sigma=0.3, by="grapheme", stagger=0.02))
```

48.1.58. Anim.tracking

MÉTODO `tracking(value: float) -> Anim`

Los glifos se desplazan por la línea base sin rehacer el diseño, anclados según la alineación del texto. Easing predeterminado: suave. Se combina con animaciones que no mueven glifos, como `blur_in`; `scene.play` rechaza dos animaciones simultáneas que escriben el mismo canal de glifos.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`value`	`float`	obligatorio	Espacio extra entre glifos vecinos, en unidades de escena; `0` restaura el original.

Devuelve: `Anim` — Cambio animado del espaciado.

Ejemplo:

```
title = scene.text("ESPACIO", role="title")
title.tracking(0.4)
scene.play(title.animate.tracking(0.0).duration(1.2))
```

48.1.59. Anim.path_offset

MÉTODO `path_offset(value: float) -> Anim`

Mueve un texto creado con `scene.text.on_path` por su trayectoria. En una trayectoria cerrada `1.0` es una vuelta completa. Cada fotograma se calcula a partir del desplazamiento, así que un seek muestra lo mismo que la reproducción. Easing predeterminado: suave; usa `Easing.LINEAR` para un giro constante. Un texto que no sea de `on_path` o un valor no finito lanzan `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`value`	`float`	obligatorio	Desplazamiento a lo largo de la trayectoria, como fracción de su longitud.

Devuelve: `Anim` — Recorrido animado del texto por su trayectoria.

Ejemplo:

```
ring = scene.text.on_path("GAANIM · MOTION · ", scene.geometry.circle(2.0), reverse=True)
scene.play([ring.animate.path_offset(1.0).duration(4).easing(Easing.LINEAR)])
```

48.1.60. Transformaciones

48.1.61. Anim.transform_to

MÉTODO `transform_to(target: Drawable) -> Anim`

Transforma la geometría del objeto hasta la del destino. Respeta los tiempos compuestos y un relleno ausente no se inventa. Cada contorno se empareja con el sentido de recorrido y el punto de inicio que menos mueven sus puntos, así que una forma y su espejo no colapsan a mitad del morph. Un relleno o trazo con degradado se interpola hacia el del destino (posición y paradas); dos degradados de tipos distintos pasan por un color liso a mitad. De un `Text` a otro `Text` es una transición estructural: las partes con el mismo nombre viajan a su sitio, cada glifo conserva su color (también los de `part()`) y al terminar se muestra el destino; entre textos de Layouts distintos es un morph de forma única (ver Texto). Entre grupos o SVG importados (también una parte <g> de un SVG) el morph es pieza a pieza y en el sitio: las partes con el mismo nombre o `id` de SVG se emparejan, a cualquier profundidad, y las demás hojas se emparejan por forma dentro de su pareja; las hojas sobrantes del objeto se desvanecen y las del destino aparecen donde está el destino. El objeto conserva su identidad, así que sus `part()` siguen valiendo después. También conserva su propia opacidad: el destino es solo una plantilla, oculta durante y después del morph, así que un destino declarado con `.opacity(0)` no lo deja invisible. Para cambiarla mientras se transforma, reproduce un `animate.opacity` del mismo objeto en el mismo `play`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`target`	`Drawable`	obligatorio	Objeto de la misma escena cuya forma se adopta.

Devuelve: `Anim` — Transformación en el sitio.

Ejemplo:

```
from gaanim import Easing, BLUE, GOLD, Scene
scene = Scene(frame=(16, 9), background="#0f172a")
circle = scene.geometry.circle(0.525).fill(BLUE).move_to(-1.125, 0)
target = scene.geometry.rect(1.125, 0.75).fill(GOLD).move_to(1, 0)
scene.play([circle.animate.create().duration(0.6)])
scene.play([circle.animate.transform_to(target).duration(1.0).easing(Easing.spring(stiffness=90,
damping=12))])
scene.render()
```

48.1.62. Anim.fade_transform_to

MÉTODO `fade_transform_to(target: Drawable) -> Anim`

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`target`	`Drawable`	obligatorio	Objeto de la misma escena.

Devuelve: `Anim` — Fundido cruzado hacia el destino.

48.1.63. Anim.replacement_transform_to

MÉTODO `replacement_transform_to(target: Drawable) -> Anim`

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`target`	`Drawable`	obligatorio	Objeto de la misma escena.

Devuelve: `Anim` — Transformación que termina sustituyendo el origen por el destino.

Ejemplo:

```
scene.play(square.animate.replacement_transform_to(circle))
```

48.1.64. Geometry.transform_matching_shapes

MÉTODO `transform_matching_shapes(source: Drawable, target: Drawable, *, duration: float = 1.0) -> None`

Empareja las piezas de origen y destino por forma, posición y color, transforma las parejas y funde el resto. Se reproduce como un `scene.play`: lo siguiente empieza cuando termina. El destino está oculto hasta entonces; al final se muestra y el origen se oculta, así que las animaciones siguientes del destino mueven lo que se ve.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`source`	`Drawable`	obligatorio	Objeto o grupo de origen.
`target`	`Drawable`	obligatorio	Objeto o grupo de destino.
`duration`	`float`	`1.0`	Segundos; finito y positivo.

Devuelve: `None` — Reproduce la transformación en el cursor y lo avanza `duration` segundos.

Ejemplo:

```
from gaanim import GOLD, WHITE, Scene

scene = Scene(frame=(16, 9), background="#0f172a")
e1 = scene.text("$E = m c$").fill(WHITE).scale_to(1.3)
e2 = scene.text("$p = m v$").fill(GOLD).scale_to(1.3)
scene.play(e1.animate.write())
scene.geometry.transform_matching_shapes(e1, e2, duration=1.2)
scene.render()
```

48.1.65. Geometry.transform_matching

MÉTODO `transform_matching(source: Drawable, target: Drawable, *, mode: str = "shapes", duration: float = 1.0) -> None`

Versión general de `transform_matching_shapes`: al final se muestra el destino y se oculta el origen. Otro `mode` lanza `ValueError`. Para textos estructurados es preferible `animate.transform_to`, que es un `Anim` normal.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`source, target`	`Drawable`	obligatorio	Objetos de origen y destino.
`mode`	`str`	`"shapes"`	`"shapes"` empareja por geometría; `"tex"` (o `"text"`) empareja los glifos de dos textos o ecuaciones por carácter, en orden de lectura, y vuelve a la geometría si alguno no tiene glifos.
`duration`	`float`	`1.0`	Segundos; finito y positivo.

Devuelve: `None` — Reproduce la transformación en el cursor y lo avanza `duration` segundos.

48.1.66. magic_move

FUNCIÓN `magic_move(before: Drawable, after: Drawable, *, key: str | Callable[[Drawable], object] = "name", unmatched: str = "fade", duration: float | None = None) -> Anim`

Los miembros de `before` y `after` con la misma clave interpolan posición, tamaño, color y forma; una pareja de grupos transforma también sus piezas, como `transform_matching`. Los candidatos se buscan en profundidad y un miembro con clave se mueve entero. Las claves se leen al llamar a `magic_move` y las repetidas se emparejan en orden de declaración. `after` queda oculto hasta el movimiento y lo sustituye al final, así que las animaciones siguientes continúan desde él. Sin `duration` dura un segundo, o la `duration` que reciba `scene.play`. Objetos de escenas distintas, el mismo objeto dos veces, una duración no positiva o un `key`/`unmatched` desconocido lanzan `ValueError`; una excepción de la función clave se propaga.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`before, after`	`Drawable`	obligatorio	Estados de origen y destino, normalmente grupos de la misma escena.
`key`	`str | Callable[[Drawable], object]`	`"name"`	`"name"` usa los nombres de `Drawable.named` (una parte SVG sin nombre responde a su `id`); `"id"` usa solo los `id` SVG; una función recibe cada candidato y devuelve su clave, o `None` para buscar dentro de él.
`unmatched`	`str`	`"fade"`	`"fade"` funde a la vez los miembros sin pareja; `"cut"` los quita al

			empezar y muestra los nuevos al terminar.
`duration`	`float`	`1.0`	Segundos; finito y positivo.

Devuelve: `Anim` — Magic move listo para `scene.play`.

Ejemplo:

```
from gaanim import BLUE, GOLD, GREEN, RED, Scene, magic_move
scene = Scene(frame=(16, 9), background="#0f172a")
def board(rows):
    bars = [
        scene.geometry.rect(score, 0.6).fill(color).move_to(score / 2 - 4, 1.5 - rank).named(name)
        for rank, (name, score, color) in enumerate(rows)
    ]
    return scene.geometry.group(bars)
before = board([("ana", 5, BLUE), ("bo", 4, GOLD), ("cy", 3, RED)])
after = board([("bo", 6, GOLD), ("ana", 4.5, BLUE), ("dee", 2, GREEN)])
scene.wait(0.3)
scene.play([magic_move(before, after, key="name", unmatched="fade").duration(0.8)])
scene.wait(0.3)
scene.render()
```

48.1.67. Animaciones personalizadas

48.1.68. Anim.custom

MÉTODO `custom(callback: Callable[[float], CustomAnimationValues], *, channels: Sequence[AnimationChannel]) -> Anim`

Los valores son absolutos: `position` es una pareja o terna local en unidades de escena, `rotation` un ángulo Z en radianes, `scale` un factor o una terna, `opacity` un número entre 0 y 1, `fill` y `stroke` pinturas y `stroke_width` un ancho no negativo. La función se evalúa en el instante exacto al reproducir, buscar o exportar; debe ser síncrona y pura, sin modificar la escena. El

progreso puede salir de `[0, 1]` con algunos easings. No mezcles `custom` y setters en un mismo `Anim`: usa `parallel`. Si falla durante la reproducción, los canales vuelven a su valor inicial y aparece un diagnóstico; la exportación falla. Los tipos `AnimationChannel` y `CustomAnimationValues` se importan desde `gaanim`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`callback`	`Callable[[float], CustomAnimationValues]`	obligatorio	Recibe el progreso ya con easing y devuelve un diccionario con exactamente los canales declarados.
`channels`	`Sequence[AnimationChannel]`	obligatorio	Canales que escribe: `"position"`, `"rotation"`, `"scale"`, `"opacity"`, `"fill"`, `"stroke"`, `"stroke_width"`.

Devuelve: `Anim` — Una animación normal: admite duración, retraso, easing y composición.

Ejemplo:

```
motion = dot.animate.custom(
    lambda alpha: {
        "position": (3 * alpha, alpha * alpha),
        "opacity": 1 - 0.5 * alpha,
    },
    channels=("position", "opacity"),
).duration(2).easing(Easing.SMOOTH)
scene.play(parallel(motion, dot.animate.fill(BLUE).duration(2)))
```

48.1.69. Keyframes y física ligera

48.1.70. Anim.keyframes

MÉTODO `keyframes(times: Sequence[float], *, position: Optional[Sequence[Optional[tuple[float, float] | tuple[float, float, float]]]] = None, offset: Optional[Sequence[Optional[tuple[float, float] | tuple[float, float, float]]]] = None, rotation: Optional[Sequence[Optional[float]]] = None, scale: Optional[Sequence[Optional[float | tuple[float, float] | tuple[float, float, float]]]] = None, opacity: Optional[Sequence[Optional[float]]] = None, fill: Optional[Sequence[Optional[Paint]]] = None, stroke:`

```
Optional[Sequence[Optional[Paint]]] = None, values:
Optional[Sequence[Optional[float]]] = None, easing: Optional[Easing |
Sequence[Easing]] = None, spatial: Literal["linear", "catmull_rom"] = "linear") -
> Anim
```

Describe un movimiento de varios tiempos sin encadenar `sequence` del mismo objeto, como los `keyframes` de GSAP o `interpolate` de Remotion. El easing propio del clip queda lineal para que cada parada caiga en su instante: `Scene.play(easing=)` no lo cambia y `.easing()` remapea el clip entero. Dura 1 s o lo que indique `.duration()`, y admite `repeat` y `loop`. Longitudes distintas, instantes que no van de 0 a 1 o valores inválidos lanzan `ValueError`; no se combina con otros destinos en el mismo `Anim` ni con `settle`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`times`	`Sequence[float]`	obligatorio	Instantes de las paradas como fracciones de la animación: empiezan en `0`, terminan en `1` y crecen.
`position / rotation / scale / opacity / fill / stroke`	`Sequence`	`None`	Un valor por instante. `None` conserva el valor que tiene el objeto al empezar. `position`

es `(x, y)` o `(x, y, z)` y fija el origen del objeto, no su centro visual como `move_to`: coinciden en una forma creada en el origen, pero no en un SVG o un texto dibujado en otro sitio. `rotation` es un ángulo absoluto en radianes, `scale`

			un factor o `(sx, sy[, sz])`, `opacity` entre 0 y 1, `fill` y `stroke` pinturas.
`offset`	`Sequence[tuple | None]`	`None`	Desplazamientos desde donde está el objeto al empezar, en lugar de posiciones: `offset=[None, (3, 0), None]` va 3 unidades a la derecha y vuelve, esté donde esté su dibujo. `None` equivale a

			`(0, 0)`. No se combina con `position`.
`values`	`Sequence[float | None]`	`None`	En `Parameter.anima` los valores del parámetro en lugar de canales.
`easing`	`Easing | Sequence[Easing]`	`None`	Un easing para todos los tramos o uno por tramo; lineal si se omite.
`spatial`	`"linear" | "catmull_rom"`	`"linear"`	Con `"catmull_rom"` la posición recorre una curva suave que pasa por las paradas, como los

keyframes espaciales de After Effects.

Devuelve: `Anim` — Un solo clip con todas las paradas.

Ejemplo:

```
from gaanim import CORAL, GOLD, Easing, EasingCurve, Scene
scene = Scene(frame=(16, 9))
ball = scene.geometry.circle(0.45).fill(GOLD).move_to(-5.5, -1.5)
level = scene.viz.parameter(0.4)
dot = scene.geometry.circle(0.5).fill(CORAL).move_to(4.5, 2.0)
dot.scale_to(level)
scene.play([
    ball.animate.keyframes(
        times=[0.0, 0.35, 0.7, 1.0],
        position=[None, (-2.0, 1.5), (1.0, -1.5), (5.0, -1.5)],
        scale=[1.0, 1.0, (1.35, 0.7), 1.0],
        easing=[Easing.ease_out(EasingCurve.QUADRATIC), Easing.ease_in(EasingCurve.QUADRATIC),
Easing.SMOOTH],
        spatial="catmull_rom",
    ).duration(1.6),
    level.animate.keyframes(times=[0, 0.3, 1], values=[None, 1.6, 0.8],
easing=Easing.SMOOTH).duration(1.6),
])
scene.render()
```

48.1.71. Anim.throw

MÉTODO `throw(velocity: float | tuple[float, float], *, gravity: float = 9.8, floor: Optional[float] = None, restitution: float = 0.55) -> Anim`

Balística por tramos analíticos, como `Physics2D` de GSAP: un arco, rebotes cada vez más bajos y reposo cuando un rebote subiría menos de una milésima de unidad. Es una función pura del tiempo, así que cualquier seek es exacto. Si no se indica, la duración es la del lanzamiento y

`Scene.play(duration=)` no la cambia; `.duration()` reproduce el mismo arco más rápido o más lento. Un segundo `throw` sigue desde donde reposó el primero. La física de cuerpos rígidos con colisiones queda fuera: hornéala con `add_updater_fn(fixed_dt=...)`. Valores inválidos, empezar bajo el suelo o lanzar hacia abajo sin suelo lanzan `ValueError`.

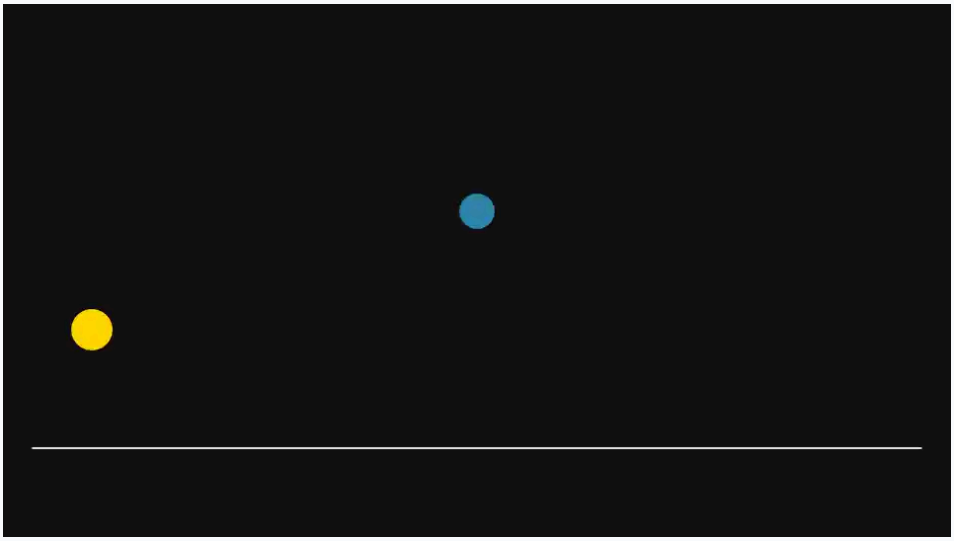
Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`velocity`	`float | tuple[float, float]`	obligatorio	`(vx, vy)` en unidades por segundo; un número lanza en horizontal.
`gravity`	`float`	`9.8`	Aceleración hacia abajo en unidades por segundo al cuadrado; positiva.
`floor`	`float | None`	`None`	Altura en la que rebota el punto más bajo del objeto, medido al llamar a `throw`. Sin suelo vuela hasta volver a su altura inicial, lo que pide una velocidad hacia arriba.
`restitution`	`float`	`0.55`	Fracción de la velocidad que conserva cada rebote, en `[0, 1)`.

Devuelve: `Anim` — El lanzamiento desde donde esté el objeto.

Ejemplo:

```python
from gaanim import GOLD, TEAL, WHITE, Scene
scene = Scene(frame=(16, 9))
scene.geometry.line(-7.5, -3.0, 7.5, -3.0).stroke(WHITE, 0.03)
ball = scene.geometry.circle(0.35).fill(GOLD).move_to(-6.5, -1.0)
coin = scene.geometry.circle(0.3).fill(TEAL).move_to(0.0, 1.0)
scene.play([
    ball.animate.throw(velocity=(3.0, 4.0), floor=-3.0, restitution=0.6),
    coin.animate.inertia(velocity=7.0, friction=2.5, snap=[0.0, 2.0, 4.0, 6.0]),
])
scene.render()
```



48.1.72. Anim.inertia

MÉTODO `inertia(velocity: float | tuple[float, float], *, friction: float = 3.0, snap: Optional[Sequence[float | tuple[float, float]]] = None) -> Anim`

Deceleración exponencial, como `Inertia` de GSAP: el objeto reposaría a `velocity / friction` de donde empieza y termina, en cambio, en el punto de `snap` más cercano a ese reposo, aunque quede detrás. Si no se indica, dura `ln(1000) / friction` segundos, lo que tarda en recorrer el 99,9 %. Valores inválidos lanzan `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`velocity`	`float | tuple[float, float]`	obligatorio	`(vx, vy)` en unidades por segundo; un número desliza en horizontal.
`friction`	`float`	`3.0`	Frenado por segundo, positivo: la velocidad cae a `1/e` cada

			`1 / friction` segundos.
`snap`	`Sequence[float | tuple[float, float]] | None`	`None`	Puntos de reposo: un número fija solo `x` y `(x, y)` un punto entero.

Devuelve: `Anim` — El deslizamiento desde donde esté el objeto.

48.1.73. Efectos y trazos

Estos destinos interpolan los efectos estáticos del mismo nombre de Drawable y se combinan con otros destinos de propiedad.

48.1.74. Anim.glow

MÉTODO `glow(color: Optional[ColorLike] = None, radius: float = 0.16, intensity: float = 1.0) -> Anim`

Un objeto sin resplandor lo hace crecer desde intensidad cero. Valores inválidos lanzan `ValueError`; en una selección de texto, `TypeError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`color`	`Color | None`	`None`	Color del resplandor; `None` lo desvanece.
`radius`	`float`	`0.16`	Radio en unidades de escena.
`intensity`	`float`	`1.0`	Intensidad.

Devuelve: `Anim` — Resplandor animado.

Ejemplo:

```
scene.play(orb.animate.glow(CYAN, radius=0.5, intensity=2.0).repeat(3, yoyo=True))
```

48.1.75. Anim.blur

MÉTODO `blur(sigma: float = 0.04) -> Anim`

Parámetros:

Nombre	Tipo	Predeterminado	Descripción

`sigma`	`float`	`0.04`	Desenfoque final; `0` termina nítido.

Devuelve: `Anim` — Desenfoque animado.

Ejemplo:

```
hero.blur(0.3)
scene.play(hero.animate.blur(0.0).duration(0.6))   # entrada desde el desenfoque
```

48.1.76. Anim.shadow

MÉTODO `shadow(color: Optional[ColorLike] = None, x: float = 0.08, y: float = -0.08, blur: float = 0.06) -> Anim`

Una sombra nueva crece desde debajo del objeto.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`color`	`Color | None`	`None`	Color de la sombra; `None` la desvanece. Su alfa escala la opacidad de forma continua.
`x, y`	`float`	`0.08, -0.08`	Desplazamiento de la sombra.
`blur`	`float`	`0.06`	Desenfoque de la sombra.

Devuelve: `Anim` — Sombra animada.

Ejemplo:

```
card.shadow(BLACK, 0, -0.05, 0.05)
scene.play(card.animate.shadow(BLACK, 0, -0.25, 0.4).scale_to(1.04))   # levantar
```

48.1.77. Anim.trim

MÉTODO `trim(start: Optional[float] = None, end: Optional[float] = None, offset: Optional[float] = None) -> Anim`

Los valores omitidos conservan el actual. Animar `offset` hace viajar un segmento. El recorte también se mantiene en trazos que se regeneran en cada fotograma, como conectores, líneas entre extremos y curvas reactivas. Ver `Drawable.trim`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`start, end`	`float | None`	`None`	Ventana visible del trazo, en fracciones de longitud de arco en `[0, 1]`.
`offset`	`float | None`	`None`	Desplaza la ventana y da la vuelta al final del camino.

Devuelve: `Anim` — Recorte animado del trazo.

Ejemplo:

```
logo.trim(end=0.0)
scene.play(logo.animate.trim(end=1.0).duration(1.2))            # dibujar
ring.trim(start=0.5, end=0.5)
scene.play(ring.animate.trim(start=0.0, end=1.0))               # desde el centro
orbit.trim(start=0.0, end=0.15)
scene.play(orbit.animate.trim(offset=1.0).duration(2))          # segmento viajero
```

48.1.78. Anim.dash_offset

MÉTODO `dash_offset(offset: float) -> Anim`

Anima el `dash_offset` de todos los trazos del objeto, incluidos los de un grupo o un texto. El trazo necesita guiones (`StrokeStyle(dashes=...)`). Un valor mayor lleva el patrón hacia el inicio del camino, así que animarlo hace marchar los guiones. La siguiente animación parte del valor final. Para un flujo sin fin usa `Updater.dash_flow`. Un valor no finito lanza `ValueError`.

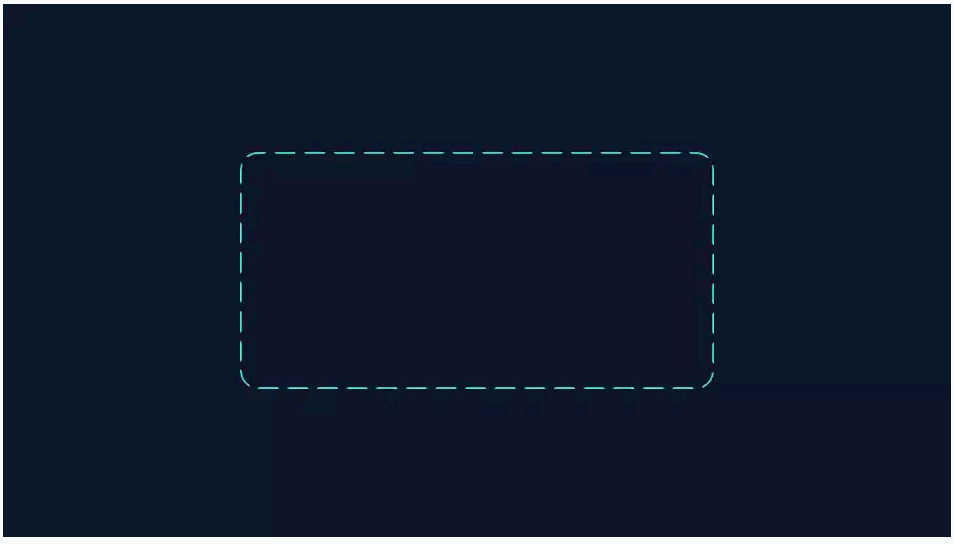
Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`offset`	`float`	obligatorio	Desplazamiento del patrón de guiones, en unidades de escena.

Devuelve: `Anim` — Desplazamiento animado de los guiones.

Ejemplo:

```
from gaanim import CYAN, Scene, StrokeStyle
scene = Scene(frame=(16, 9), background="#0f172a")
border = scene.geometry.rounded_rect(8, 4, 0.3).no_fill().stroke_style(StrokeStyle(CYAN, 0.06,
dashes=[0.3, 0.2]))
scene.play([border.animate.dash_offset(2.0).duration(2)])
scene.render()
```



48.1.79. Anim.count

MÉTODO `count(count: float) -> Anim`

Anima cuántas copias de un grupo creado con `Geometry.repeat` o `Geometry.duplicate` se ven. Las copias aparecen en orden desde la primera: la copia `i` se ve cuando el recuento supera `i` y la parte fraccionaria funde la siguiente, así que `count(24)` desde 0 construye el patrón copia a copia. La opacidad de cada copia se conserva. El máximo es el número de copias con que se creó el grupo; para hacer crecer un patrón, créalo con todas y baja antes su recuento con `Drawable.count`. Un valor fuera de rango, o un objeto que no sea uno de esos grupos, lanza `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`count`	`float`	obligatorio	Copias visibles al terminar, de 0 al número de copias del grupo.

Devuelve: `Anim` — Recuento animado de las copias.

Ejemplo:

```
from gaanim import GOLD, Easing, Scene
scene = Scene(frame=(16, 9), background="#0d1322")
leaf = scene.geometry.ellipse(0.12, 0.34).fill(GOLD).move_to(0, 2.4)
spiral = scene.geometry.repeat(leaf, 36, rotate=0.5, scale=0.95, about=(0, 0)).count(0)
scene.play([spiral.animate.count(36).duration(2.0).easing(Easing.LINEAR)])
scene.render()
```

48.1.80. Anim.points

MÉTODO `points(points: Sequence[tuple[float, float]]) -> Anim`

Para fijar los vértices sin animar, usa `Drawable.points`. Lleva cada vértice de un polígono o una polilínea en línea recta a su nueva posición: el vértice `i` va a `points[i]`. A diferencia de `transform_to` no remuestrea el contorno, así que cada fotograma es la mezcla exacta de los dos

y las figuras de píxel o técnicas conservan sus esquinas. Un `move_to` o `shift` posterior se sigue aplicando encima. La figura conserva su número de vértices: otro número de puntos, una figura que no sea polígono o polilínea (o que lo sea con coordenadas reactivas), o un punto no finito lanzan `ValueError`. Se combina con `fill`, `move_to` y los demás destinos de propiedades en un mismo `Anim`, y `echo` repite el morph en sus copias.

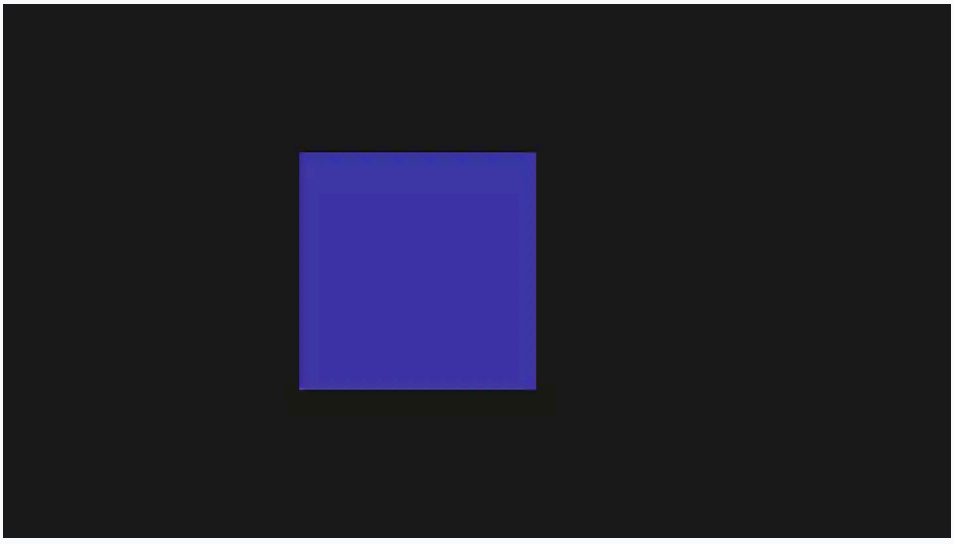
Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`points`	`Sequence[tuple[float, float]]`	obligatorio	Posición final de cada vértice, en las coordenadas en que se declaró la figura.

Devuelve: `Anim` — Morph vértice a vértice.

Ejemplo:

```
from gaanim import Easing, Scene
# El símbolo de Gaanim: un fotograma de píxeles que avanza y deja los anteriores con echo.
FRAMES = ((8,) * 8, (6,) + (8,) * 6 + (6,), (4, 6, 8, 8, 8, 8, 6, 4))
COLORS = ("#3F37A8", "#7C6CFF", "#FFC933")

def outline(widths, cx, pixel=0.5):
    top, right, left = len(widths) * pixel / 2, [], []
    for row, width in enumerate(widths):
        y0, y1, half = top - row * pixel, top - (row + 1) * pixel, width * pixel / 2
        right += [(cx + half, y0), (cx + half, y1)]
        left += [(cx - half, y0), (cx - half, y1)]
    return right + left[::-1]

scene = Scene(frame=(16, 9), background="#1A1A1A")
symbol = scene.geometry.polygon(outline(FRAMES[0], -1)).fill(COLORS[0]).echo(2, delay=0.6,
decay=1.0)
for index in (1, 2):
    scene.play([symbol.animate.points(outline(FRAMES[index], index -
1)).fill(COLORS[index]).duration(0.6).easing(Easing.LINEAR)])
scene.wait(1.0)
scene.render()
```



48.1.81. Tiempo

48.1.82. Anim.duration

MÉTODO `duration(seconds: float) -> Anim`

`duration(0)` es un corte: el destino se aplica de golpe en ese punto de la composición. Sirve para cambiar un valor a mitad de un `sequence` o de un `play` con `delay`, sin partir el bloque: `sequence(a.animate.shift_by(2, 0), a.animate.opacity(0.3).duration(0), a.animate.shift_by(-2, 0))`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`seconds`	`float`	obligatorio	Duración finita y no negativa.

Devuelve: `Anim` — El `Anim` configurado.

Ejemplo:

```
scene.play(circle.animate.shift_by(3, 0).duration(1.0).delay(0.3).easing(Easing.LINEAR))
```

48.1.83. Anim.delay

MÉTODO `delay(seconds: float) -> Anim`

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`seconds`	`float`	obligatorio	Espera antes de empezar; finita y no negativa.

Devuelve: `Anim` — El `Anim` configurado.

48.1.84. Anim.easing

MÉTODO `easing(easing: Easing) -> Anim`

No se aceptan nombres en texto: usa siempre un `Easing`. Ver Easing.

Parámetros:

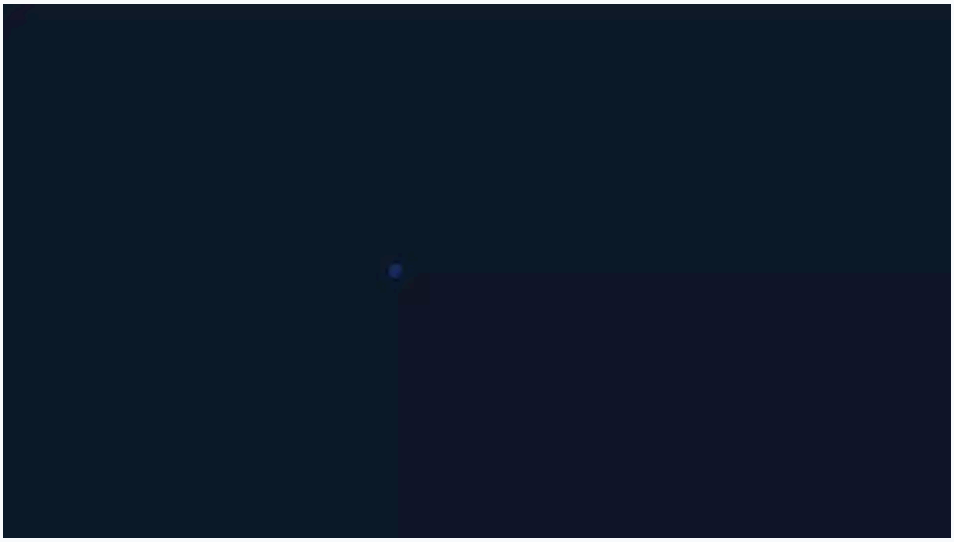
Nombre	Tipo	Predeterminado	Descripción
`easing`	`Easing`	obligatorio	Preset o resultado de una fábrica de `Easing`.

Devuelve: `Anim` — El `Anim` configurado.

Ejemplo:

```
from gaanim import Easing, BLUE, Scene
scene = Scene(frame=(16, 9), background="#0f172a")
dot = scene.geometry.dot(0.125).fill(BLUE).move_to(-1.375, 0)
```

```
scene.play([dot.animate.shift_by(2.75, 0).duration(0.9).easing(Easing.spring(stiffness=90,
damping=12))])
scene.play([dot.animate.shift_by(-2.75, 0).duration(0.9).easing(Easing.SMOOTH)])
scene.render()
```



48.1.85. Anim.lag_ratio

MÉTODO `lag_ratio(value: float) -> Anim`

Escalona los subcaminos o miembros dentro de un mismo objeto, por ejemplo los de un grupo.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`value`	`float`	obligatorio	Solapamiento entre subtrazos, en `[0, 1]`: `0` los anima a la vez y `1` uno detrás de otro.

Devuelve: `Anim` — El `Anim` configurado.

Ejemplo:

```
from gaanim import BLUE, WHITE, Scene
scene = Scene(frame=(16, 9), background="#0f172a")
g = scene.geometry.group([scene.geometry.circle(0.225).fill(BLUE).move_to(-0.625,0),
scene.geometry.circle(0.225).fill(BLUE).move_to(0,0),
scene.geometry.circle(0.225).fill(BLUE).move_to(0.625,0)])
scene.play([g.animate.create().duration(1.0).lag_ratio(0.25)])
scene.render()
```

48.1.86. Anim.repeat

MÉTODO `repeat(count: int, *, yoyo: bool = False, delay: float = 0.0) -> Anim`

`duration` y `easing` describen un ciclo. Con `yoyo=True` un número par de ciclos termina donde empezó y las animaciones siguientes continúan desde ahí. La duración total es `count * duration + (count - 1) * delay`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`count`	`int`	obligatorio	Número de ciclos, entre 1 y 10000.
`yoyo`	`bool`	`False`	Alterna el sentido de cada ciclo.
`delay`	`float`	`0.0`	Segundos entre ciclos; no negativo.

Devuelve: `Anim` — El `Anim` repetido.

Ejemplo:

```
scene.play(spinner.animate.rotate_by(math.tau).duration(1.2).repeat(3))
scene.play(badge.animate.scale_to(1.08).duration(0.4).repeat(4, yoyo=True, delay=0.1))
```

48.1.87. Anim.loop

MÉTODO `loop(mode: Literal["cycle", "pingpong", "offset"] = "cycle", *, until: float, delay: float = 0.0) -> Anim`

El bucle es finito, así que los seeks y la exportación son exactos.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`mode`	`"cycle" | "pingpong" | "offset"`	`"cycle"`	`"cycle"` reinicia cada ciclo, `"pingpong"` alterna el sentido y `"offset"` continúa desde donde terminó el anterior, así `rotate_by` o `shift_by` se acumulan.
`until`	`float`	obligatorio	Segundos disponibles; caben tantos ciclos completos como sea posible, al menos uno.
`delay`	`float`	`0.0`	Segundos entre ciclos.

Devuelve: `Anim` — El `Anim` repetido.

Ejemplo:

```
scene.play(arrow.animate.shift_by(0.3, 0).duration(0.5).loop("pingpong", until=4.0))
```

48.1.88. Anim.settle

MÉTODO `settle(overshoot: float = 0.12, frequency: float = 3.0, decay: float = 6.0) -> Anim`

Follow-through: al terminar el movimiento, cada propiedad animada rebasa su destino y vuelve oscilando como `v * overshoot * sin(2π * frequency * t) / e^(decay * t)`, donde `t` son los segundos tras el movimiento y `v` su velocidad final en unidades por segundo. Si el easing termina en reposo (como `smooth`) se usa la velocidad media, así que un movimiento suavizado también rebota. El clip se alarga hasta que el rebote cae bajo el 0,1 % de la distancia recorrida (como mucho 10 s más) y termina exactamente en el destino; `play` lo espera. Va después del último ciclo de `repeat` o `loop`. Es una función pura del tiempo, así que cualquier seek es exacto. Valores negativos o no finitos lanzan `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`overshoot`	`float`	`0.12`	Segundos de rebote por unidad de velocidad final (el `amp` de After Effects); `0` desactiva el rebote.
`frequency`	`float`	`3.0`	Oscilaciones por segundo; positiva.
`decay`	`float`	`6.0`	Amortiguación exponencial en 1/s; positiva.

Devuelve: `Anim` — El `Anim` con rebote inercial.

Ejemplo:

```
scene.play(card.animate.move_to(0, 0).duration(0.5).settle(overshoot=0.12, frequency=3.0,
decay=6.0))
```

48.1.89. Easing

Un `Easing` es una función de tiempo inmutable: convierte el progreso lineal del clip en el progreso que se ve. Hay presets con nombre y fábricas que validan sus argumentos; no se aceptan nombres en texto ni hay un valor de reserva silencioso. Todas rechazan números no finitos y dominios inválidos con `ValueError`.

48.1.90. Easing.LINEAR / SMOOTH / DOUBLE_SMOOTH / THERE_AND_BACK / LINGERING / RUNNING_START / EXPONENTIAL_DECAY / NOT_QUITE_THERE

CONSTANTE `Easing.LINEAR · Easing.SMOOTH · …: Easing`

Curvas clásicas.

Ejemplo:

Preset	Forma
`LINEAR`	Velocidad constante.
`SMOOTH`	Arranca y frena suave ($3t^2 - 2t^3$). Es el easing predeterminado de muchas animaciones.
`DOUBLE_SMOOTH`	`SMOOTH` aplicado dos veces: arranque y frenado más marcados. Predeterminado de `create`.
`THERE_AND_BACK`	Llega al destino a mitad del clip y vuelve al inicio.
`LINGERING`	Curva suave de quinto grado ($6t^5 - 15t^4 + 10t^3$), con extremos todavía más lentos que `SMOOTH`.
`RUNNING_START`	Retrocede un poco antes de arrancar hacia el destino.
`EXPONENTIAL_DECAY`	Sale rápido y se acerca al destino cada vez más despacio.
`NOT_QUITE_THERE`	Como `EXPONENTIAL_DECAY`, pero se queda en el 95 % del recorrido.

```
from gaanim import Anchor, Easing, GOLD, Scene
scene = Scene(frame=(16, 9), background="#0f172a")
presets = [("LINEAR", Easing.LINEAR), ("SMOOTH", Easing.SMOOTH), ("DOUBLE_SMOOTH",
Easing.DOUBLE_SMOOTH), ("RUNNING_START", Easing.RUNNING_START), ("EXPONENTIAL_DECAY",
Easing.EXPONENTIAL_DECAY)]
moves = []
for row, (name, easing) in enumerate(presets):
    y = 2.4 - 1.2 * row
    scene.text(name, size=0.32).move_to(-3.4, y, Anchor.RIGHT)
    dot = scene.geometry.dot(0.16).fill(GOLD).move_to(-1.6, y)
    moves.append(dot.animate.move_to(6, y).duration(1.5).easing(easing))
scene.play(moves)
scene.wait(0.3)
scene.render()
```

48.1.91. Easing.SMOOTH_SPRING / GENTLE / QUICK / SNAPPY / BOUNCY

CONSTANTE `Easing.SMOOTH_SPRING · Easing.GENTLE · …: Easing`

Resortes perceptuales con nombre: se asientan dentro de la duración de la animación y terminan exactamente en el destino.

Ejemplo:

Preset	Carácter
`SMOOTH_SPRING`	Amortiguamiento crítico: el asentamiento más rápido sin pasarse.
`GENTLE`	Suave y sin prisa, con un sobrepaso apenas visible.
`QUICK`	Enérgico, con un sobrepaso pequeño que se asienta pronto.
`SNAPPY`	Rápido, con un sobrepaso nítido del 20 %.
`BOUNCY`	Juguetón, con un sobrepaso del 45 % y rebotes visibles.

```
scene.play(card.animate.scale_to(1.0).duration(0.6).easing(Easing.SNAPPY))
```

48.1.92. EasingCurve.QUADRATIC / CUBIC / QUARTIC / QUINTIC / EXPONENTIAL / SINE / CIRCULAR / BACK / ELASTIC / BOUNCE

CONSTANTE `EasingCurve.QUADRATIC · EasingCurve.CUBIC · …: EasingCurve`

Familias de curvas para `Easing.ease_in`, `ease_out` y `ease_in_out`. `QUADRATIC` a `QUINTIC` son potencias de grado 2 a 5; `EXPONENTIAL`, `SINE` y `CIRCULAR`, las curvas clásicas de CSS; `BACK` se pasa un poco del destino, `ELASTIC` oscila y `BOUNCE` rebota.

48.1.93. Easing.ease_in

FÁBRICA `ease_in(curve: EasingCurve) -> Easing`

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`curve`	`EasingCurve`	obligatorio	Familia de la curva.

Devuelve: `Easing` — Arranque lento y final rápido.

48.1.94. Easing.ease_out

FÁBRICA `ease_out(curve: EasingCurve) -> Easing`

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`curve`	`EasingCurve`	obligatorio	Familia de la curva.

Devuelve: `Easing` — Arranque rápido y final lento.

48.1.95. Easing.ease_in_out

FÁBRICA `ease_in_out(curve: EasingCurve) -> Easing`

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`curve`	`EasingCurve`	obligatorio	Familia de la curva.

Devuelve: `Easing` — Arranque y final lentos.

Ejemplo:

```
scene.play(circle.animate.shift_by(8, 0).easing(Easing.ease_in_out(EasingCurve.CUBIC)))
scene.play(circle.animate.shift_by(-8, 0).easing(Easing.ease_out(EasingCurve.BOUNCE)))
```

48.1.96. Easing.spring

FÁBRICA `spring(stiffness: Optional[float] = None, damping: Optional[float] = None, *, mass: float = 1.0, velocity: float = 0.0, bounce: Optional[float] = None) -> Easing`

Con `bounce`, el resorte se describe por cómo se ve: se asienta dentro de la duración de la animación, así que `duration` marca el ritmo. Sin él es físico (`stiffness`, `damping`, `mass`) y el clip abarca cinco segundos físicos. Combinar `bounce` con parámetros físicos lanza `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`stiffness`	`float | None`	`90`	Rigidez física; positiva.
`damping`	`float | None`	`12`	Amortiguamiento físico; no negativo.
`mass`	`float`	`1.0`	Masa física; positiva.
`velocity`	`float`	`0.0`	Velocidad inicial, en distancias por duración del clip.
`bounce`	`float | None`	`None`	Resorte perceptual: sobrepaso máximo en `[0, 1)`; `0` es amortiguamiento crítico.

Devuelve: `Easing` — Un resorte que termina exactamente en el destino.

Ejemplo:

```
scene.play(logo.animate.scale_to(1.0).duration(0.6).easing(Easing.spring(bounce=0.35)))
scene.play(logo.animate.rotate_by(1.0).easing(Easing.spring(stiffness=90, damping=12)))
```

48.1.97. Easing.back

FÁBRICA `back(overshoot: float = 1.70158, *, mode: Literal["in", "out", "in_out"] = "out") -> Easing`

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`overshoot`	`float`	`1.70158`	Cuánto retrocede o se pasa; no negativo.
`mode`	`"in" | "out" | "in_out"`	`"out"`	`"in"` retrocede al empezar, `"out"` se pasa al llegar y `"in_out"` hace las dos cosas.

Devuelve: `Easing` — Retroceso o sobrepaso.

48.1.98. Easing.elastic

FÁBRICA `elastic(amplitude: float = 1.0, period: float = 0.3, *, mode: Literal["in", "out", "in_out"] = "out") -> Easing`

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`amplitude`	`float`	`1.0`	Amplitud de la oscilación; al menos 1.
`period`	`float`	`0.3`	Periodo en fracciones del clip; positivo.
`mode`	`"in" | "out" | "in_out"`	`"out"`	Extremo en el que oscila.

Devuelve: `Easing` — Oscilación como una banda elástica.

48.1.99. Easing.bounce

FÁBRICA `bounce(strength: float = 1.0, *, mode: Literal["in", "out", "in_out"] = "out") -> Easing`

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`strength`	`float`	`1.0`	Mezcla de una curva cúbica (`0`) al rebote clásico (`1`).
`mode`	`"in" | "out" | "in_out"`	`"out"`	Extremo en el que rebota.

Devuelve: `Easing` — Rebotes al llegar. Con `strength=1` y `mode="out"` el objeto toca el destino por primera vez al 4/11 (≈ 0,364) de la duración y vuelve a tocarlo a 8/11, 10/11 y al final; para que el golpe caiga en un instante `t`, empieza la animación en `t - 0.364 * duración`. Con `strength < 1` solo llega exactamente al final: el primer rebote se queda en `1 - 0,258 · (1 - strength)` del recorrido, también a 4/11. Para un contacto en otro instante, escribe la curva con `Easing.custom`.

Ejemplo:

```
scene.play(ball.animate.move_to(0, -3).duration(1.2).easing(Easing.bounce()))
scene.play(ball.animate.move_to(0, 0).easing(Easing.elastic(amplitude=1.2, period=0.4)))
scene.play(ball.animate.shift_by(3, 0).easing(Easing.back(mode="in_out")))
```

48.1.100. Easing.slow_mo

FÁBRICA `slow_mo(linear_ratio: float = 0.7, power: float = 0.7) -> Easing`

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`linear_ratio`	`float`	`0.7`	Fracción central que avanza lenta y lineal, en `[0, 1]`.
`power`	`float`	`0.7`	Fuerza de la aceleración en los extremos, en `[0, 1]`.

Devuelve: `Easing` — Rápido, cámara lenta, rápido.

48.1.101. Easing.rough

FÁBRICA `rough(strength: float = 1.0, points: int = 20, seed: int = 0) -> Easing`

Parámetros:

Nombre	Tipo	Predeterminado	Descripción

`strength`	`float`	`1.0`	Intensidad del temblor.
`points`	`int`	`20`	Número de nudos aleatorios.
`seed`	`int`	`0`	Semilla de los nudos.

Devuelve: `Easing` — Rampa temblorosa determinista, para parpadeos y glitches.

Ejemplo:

```
scene.play(sign.animate.opacity(0.2).duration(1.0).easing(Easing.rough(strength=1, points=20,
seed=4)))
scene.play(sign.animate.shift_by(4, 0).duration(1.5).easing(Easing.slow_mo(0.7, 0.7)))
```

48.1.102. Easing.squish

FÁBRICA `squish(easing: Easing, start: float, end: float) -> Easing`

Útil para que una animación de una lista paralela empiece más tarde o termine antes que las demás sin cambiar su duración.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`easing`	`Easing`	obligatorio	Easing que se comprime.
`start, end`	`float`	obligatorio	Tramo del clip en el que actúa, con `0 <= start < end <= 1`.

Devuelve: `Easing` — El easing solo dentro del tramo; fuera mantiene sus extremos.

Ejemplo:

```
scene.play([
    a.animate.shift_by(8, 0).duration(2),
    b.animate.shift_by(8, 0).duration(2).easing(Easing.squish(Easing.SMOOTH, 0.5, 1.0)),
])
```

48.1.103. Easing.steps

FÁBRICA `steps(count: int, jump: Literal["start", "end", "none", "both"] = "end") -> Easing`

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`count`	`int`	obligatorio	Número de escalones.
`jump`	`"start" | "end" | "none" | "both"`	`"end"`	Dónde ocurren los saltos, como en `steps()` de CSS.

Devuelve: `Easing` — Interpolación discreta.

48.1.104. Easing.mirror

FÁBRICA `mirror(easing: Easing) -> Easing`

Convierte un `ease_in` en una curva simétrica de entrada y salida.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`easing`	`Easing`	obligatorio	Easing de la primera mitad.

Devuelve: `Easing` — El easing comprimido en la primera mitad y reflejado en la segunda.

48.1.105. Easing.there_and_back

FÁBRICA `there_and_back(pause: float = 0.0) -> Easing`

Llega al destino y regresa al inicio dentro del mismo clip.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`pause`	`float`	`0.0`	Fracción del clip que se detiene en el destino, en `[0, 1]`.

Devuelve: `Easing` — Ida y vuelta.

Ejemplo:

```
scene.play(clock.animate.rotate_by(-6.283).duration(2).easing(Easing.steps(12)))
scene.play(badge.animate.scale_to(1.3).duration(0.8).easing(Easing.there_and_back(pause=0.2)))
scene.play(badge.animate.shift_by(-6,
0).easing(Easing.mirror(Easing.ease_in(EasingCurve.QUINTIC))))
```

48.1.106. Easing.cubic_bezier

FÁBRICA `cubic_bezier(x1: float, y1: float, x2: float, y2: float) -> Easing`

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`x1, y1, x2, y2`	`float`	obligatorio	Puntos de control, como en `cubic-bezier()` de CSS; `x1` y `x2` en `[0, 1]`.

Devuelve: `Easing` — Una curva de Bézier cúbica.

48.1.107. Easing.from_svg

FÁBRICA `from_svg(path: str, samples: int = 256) -> Easing`

El camino debe ir de x = 0 a x = 1 sin retroceder en x y quedarse con y en `[-1, 2]`; si no, lanza `ValueError`. Sirve para una curva copiada de cualquier editor.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`path`	`str`	obligatorio	Camino SVG dibujado en el cuadrado unidad (x = tiempo, y = progreso).
`samples`	`int`	`256`	Muestras de la tabla interpolada.

Devuelve: `Easing` — La curva dibujada.

Ejemplo:

```
scene.play(panel.animate.shift_by(8, 0).easing(Easing.cubic_bezier(0.2, 0.8, 0.2, 1.0)))
scene.play(panel.animate.shift_by(-8, 0).easing(Easing.from_svg("M0,0 C0.3,0 0.2,1.2 1,1")))
```

48.1.108. Easing.custom

FÁBRICA `custom(function: Callable[[float], float], samples: int = 256) -> Easing`

Llama a la función `samples` veces, en tiempos equiespaciados, al crear el easing. El render nunca vuelve a Python, así que la previsualización, los seeks y la exportación coinciden. Los valores deben ser finitos y estar en `[-1, 2]` (se permite pasarse del destino); si no, lanza `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`function`	`Callable[[float], float]`	obligatorio	Función de `t` en `[0, 1]` que devuelve el progreso.
`samples`	`int`	`256`	Muestras, en `[2, 65536]`.

Devuelve: `Easing` — Una tabla interpolada linealmente.

Ejemplo:

```
salto = Easing.custom(lambda t: 1 - abs(math.cos(3 * math.pi * t)) * (1 - t) ** 2)
scene.play(ball.animate.move_to(0, -2).duration(1.2).easing(salto))
```

48.1.109. Composición

`scene.play([...])` ya reproduce una lista en paralelo. Para estructuras más ricas, estas funciones construyen un árbol inmutable, `Composition`, que `scene.play` resuelve de forma atómica: valores por defecto, tramos, solapes, conflictos de canales y destinos relativos.

48.1.110. parallel

FUNCIÓN `parallel(*items: Playable) -> Composition`

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`items`	`Playable`	obligatorio	Uno o más `Anim`, medios o composiciones.

Devuelve: `Composition` — Todos empiezan en el mismo origen.

48.1.111. sequence

FUNCIÓN `sequence(*items: Playable, gap: float = 0.0) -> Composition`

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`items`	`Playable`	obligatorio	Uno o más elementos.
`gap`	`float`	`0.0`	Segundos entre pasos; un valor negativo acotado los solapa.

Devuelve: `Composition` — Uno detrás de otro.

Ejemplo:

```
scene.play(
    sequence(
        title.animate.write().duration(0.8),
        parallel(
            box.animate.create(),
            stagger(label.animate.fade_in(), badge.animate.fade_in(), each=0.15),
        ),
        gap=-0.1,
    )
)
```

48.1.112. stagger

FUNCIÓN `stagger(*items: Playable, each: float = 0.1, total: float | None = None, origin: StaggerOrigin | None = None, grid: Literal["auto"] | tuple[int, int] | None = None, easing: Easing | None = None, seed: int = 0) -> Composition`

Sin `origin` ni `grid`, escalona por índice: `total` fija la duración de toda la onda y `easing` la acelera o frena, pero el orden sigue siendo el de la lista. Con `origin` o `grid`, el retardo crece con la distancia al origen, que es el primer elemento (`"start"`) si solo se pasa `grid`. Los elementos cuya posición depende de un layout usan su índice.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción

`items`	`Playable`	obligatorio	Elementos escalonados.
`each`	`float`	`0.1`	Retardo por índice, o por paso de separación con `origin`; no negativo.
`total`	`float | None`	`None`	Fija la duración de toda la onda en lugar de `each`.
`origin`	`StaggerOrigin | None`	`None`	`"start"`, `"end"`, `"center"`, `"edges"` (de los bordes hacia dentro), `"random"` o un punto `(x, y)`.
`grid`	`"auto" | tuple[int, int] | None`	`None`	Distancias según las posiciones declaradas (`"auto"`) o las celdas de `(filas, columnas)`.
`easing`	`Easing | None`	`None`	Da forma al reparto de retardos.
`seed`	`int`	`0`	Semilla de `origin="random"`.

Devuelve: `Composition` — Elementos desplazados en el tiempo.

Ejemplo:

```
scene.play(stagger(*[d.animate.grow_from_center() for d in dots], each=0.03, origin="center"))
scene.play(stagger(*[d.animate.indicate() for d in dots], total=1.2, origin="random", seed=7))
scene.play(stagger(*[d.animate.fill(GOLD) for d in dots], each=0.05, origin=(0.0, -3.0)))
scene.play(stagger(*[d.animate.fade_out() for d in dots], total=1.0,
easing=Easing.ease_out(EasingCurve.QUADRATIC)))
```

48.1.113. distribute

FUNCIÓN `distribute(items: Sequence[Drawable], low: float, high: float, *, origin: StaggerOrigin | None = None, grid: Literal["auto"] | tuple[int, int] | None = None, easing: Easing | None = None, seed: int = 0) -> list[float]`

Usa el mismo orden que `stagger` para repartir tamaños, colores u opacidades en lugar de tiempos; sin `origin` ni `grid`, el de la lista.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`items`	`Sequence[Drawable]`	obligatorio	Objetos que reciben un

			valor cada uno.
`low, high`	`float`	obligatorio	Rango de valores.
`origin, grid, easing, seed`		`None, None, None, 0`	Como en `stagger`.

Devuelve: `list[float]` — Un valor por objeto.

Ejemplo:

```
for dot, size in zip(dots, distribute(dots, 0.4, 1.4, origin="edges")):
    dot.scale_by(size)
```

`Playable` es la unión de todo lo que acepta `scene.play` (`Anim`, `Audio`, `Video`, `VideoSegment`, `Lottie` y `Composition`). Impórtala desde `gaanim` para anotar tus funciones; también funciona con `isinstance`.

```
from gaanim import Playable, parallel

def entrada(*items: Playable) -> Playable:
    return parallel(*items).delay(0.2)
```

48.1.114. label

FUNCIÓN `label(name: str) -> Composition`

En un `sequence` no ocupa un paso ni suma `gap`: marca dónde empieza el siguiente paso, o dónde termina el anterior si es el último. Se referencia desde `Composition.insert` y sus tiempos aparecen en `Schedule.labels`. Un nombre vacío, numérico o que empieza por <, >, +, `-` o = lanza `ValueError`. Para instantes de la línea de tiempo global, usa `scene.marker`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`name`	`str`	obligatorio	Nombre único en el árbol; se recortan los espacios.

Devuelve: `Composition` — Un instante con nombre y sin duración.

48.1.115. Composition.insert

MÉTODO `insert(item: Playable, at: str | float) -> Composition`

Las inserciones se colocan después de los hijos y antes de `stretch`, `repeat` y `delay`, que también las afectan. Una posición mal escrita lanza `ValueError` enseguida; una etiqueta desconocida (el error lista las definidas) o una posición antes de 0 lo lanzan en `schedule()` o `scene.play`. Un nombre exacto de etiqueta siempre gana, así que `"parte-2"` encuentra la etiqueta `"parte-2"`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`item`	`Playable`	obligatorio	Elemento que se añade; puede ser otro `label`.
`at`	`str | float`	obligatorio	Posición local, con la sintaxis de la tabla.

Devuelve: `Composition` — Una copia; el original no cambia.

Ejemplo:

`at`	Posición
`"nombre"`, `"nombre+0.15"`, `"nombre-0.2"`, `"nombre+=0.15"`	Una etiqueta más un desplazamiento.
`"<"`, `">"`, `"<+0.1"`, `">-0.2"`	Inicio o final del elemento anterior: la última inserción que no es etiqueta o, si no hay, el último hijo.
`"+=0.3"`, `"-=0.3"`	Relativo al final actual de la composición.
`2.5`	Segundos locales absolutos.

```
intro = (
    sequence(
        title.animate.write().duration(0.8),
        label("golpe"),
        subtitle.animate.fade_in().duration(0.4),
    )
    .insert(logo.animate.grow_from_center(), at="golpe+0.15")
    .insert(glow.animate.flash(), at="<")          # empieza con el logo
    .insert(footer.animate.fade_in(), at="-=0.2")  # se solapa con el final
)
print(intro.schedule().labels)  # {'golpe': 0.8}
scene.play(intro)

{'golpe': 0.8}
```

48.1.116. Composition.defaults

MÉTODO `defaults(*, duration: Optional[float] = None, easing: Optional[Easing] = None) -> Composition`

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`duration`	`float | None`	`None`	Duración para las animaciones descendientes que no fijan la suya.
`easing`	`Easing | None`	`None`	Easing para las que no fijan el suyo.

Devuelve: `Composition` — Una copia configurada.

48.1.117. Composition.delay

MÉTODO `delay(seconds: float) -> Composition`

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`seconds`	`float`	obligatorio	Espera antes de todo el subárbol.

Devuelve: `Composition` — Una copia retrasada.

48.1.118. Composition.stretch

MÉTODO `stretch(seconds: float) -> Composition`

Solo para árboles de animaciones: con medios lanza un error, porque cambiaría su velocidad de reproducción.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`seconds`	`float`	obligatorio	Duración exacta del subárbol.

Devuelve: `Composition` — Una copia reescalada.

48.1.119. Composition.speed_ramp

MÉTODO `speed_ramp(speeds: float | Mapping[float, float]) -> Composition`

Los *speed ramps* de los cortes modernos, como `ChangeSpeed` de Manim o `timeScale` de GSAP. Antes de la primera clave y después de la última la velocidad se mantiene. La duración pasa a ser la que tarda la rampa; los hijos conservan su orden y sus solapes, y el movimiento de cada clip sigue la rampa dentro de él, así que un seek coincide con la reproducción. Se anida y se combina con `stretch`, que se aplica después. Los updaters, la física de partículas y el temblor de cámara siguen en tiempo real. Claves fuera de `[0, 1]` o repetidas, velocidades no positivas o medios lanzan `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`speeds`	`float | Mapping[float, float]`	obligatorio	Velocidades en fracciones de la composición, de `0` a `1`, unidas linealmente: `1` es la velocidad normal, `0.15` una cámara lenta y `2` el doble. Un número es una velocidad constante.

Devuelve: `Composition` — Una copia con la rampa de velocidad.

Ejemplo:

```
from gaanim import CORAL, GOLD, TEAL, Scene, sequence
scene = Scene(frame=(16, 9))
dots = [scene.geometry.circle(0.3).fill(color).move_to(-6.0, 1.5 - 1.5 * i) for i, color in
enumerate((GOLD, CORAL, TEAL))]
steps = [dot.animate.shift_by(10, 0).duration(0.8) for dot in dots]
scene.play(sequence(*steps).speed_ramp({0.0: 1.0, 0.4: 0.15, 0.6: 0.15, 1.0: 1.0}))
scene.render()
```

48.1.120. Composition.time_remap

MÉTODO `time_remap(easing: Easing) -> Composition`

En la fracción `p` de su duración, la composición muestra lo que se escribió en `easing(p)`, como el *time remap* de After Effects o `playbackEase` de anime.js: `Easing.ease_in_out(...)` empieza y termina despacio, y `Easing.steps(4)` avanza a saltos. La duración no cambia. Los easings que retroceden (Back, Elastic, muelles) y los medios lanzan `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`easing`	`Easing`	obligatorio	Va de 0 a 1 sin retroceder.

Devuelve: `Composition` — Una copia con el tiempo remapeado.

48.1.121. Composition.repeat

MÉTODO `repeat(count: int, *, delay: float = 0.0) -> Composition`

Cada repetición parte del estado en que la dejó la anterior, así que las animaciones relativas se acumulan. Con medios, `count < 1` o un `delay` negativo lanza `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`count`	`int`	obligatorio	Repeticiones; al menos 1.
`delay`	`float`	`0.0`	Segundos entre repeticiones.

Devuelve: `Composition` — El árbol repetido.

Ejemplo:

```
scene.play(parallel(a.animate.rotate_by(math.tau), b.animate.shift_by(1, 0)).repeat(2))
```

48.1.122. Composition.schedule

MÉTODO `schedule(*, duration: Optional[float] = None) -> Schedule`

Inspecciona el árbol sin programarlo ni consumir sus hojas y sin mover el cursor.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`duration`	`float | None`	`None`	Duración por defecto exterior con la que se resuelve.

Devuelve: `Schedule` — Los tiempos locales resueltos.

Ejemplo:

```
plan = sequence(a.animate.fade_in().duration(0.5), b.animate.fade_in().duration(0.5), gap=0.2)
schedule = plan.schedule()
print(schedule.span, [(e.kind, e.start, e.end) for e in schedule.entries])

1.2 [('animation', 0.0, 0.5), ('animation', 0.7, 1.2)]
```

48.1.123. Schedule.span

PROPIEDAD `span: float`

Devuelve: `float` — Duración total de la composición.

48.1.124. Schedule.entries

PROPIEDAD `entries: tuple[ScheduleEntry, ...]`

Cada `ScheduleEntry` tiene `path` (índices en el árbol), `kind` (`"animation"`, `"audio"`, `"video"` o `"lottie"`), `start`, `duration` y `end`, en segundos locales; `duration` y `end` son `None` para un medio sin final.

Devuelve: `tuple[ScheduleEntry, ...]` — Una entrada por hoja.

48.1.125. Schedule.labels

PROPIEDAD `labels: dict[str, float]`

Incluye etiquetas anidadas e insertadas. Una composición reescalada las escala; una repetida informa de la primera repetición.

Devuelve: `dict[str, float]` — Tiempos de las etiquetas, en segundos locales y en orden temporal.

Los medios también son hojas de composición: `VideoSegment` admite `parallel`, `sequence` y `stagger`, pero no `stretch`, y su velocidad se fija al crearlo (ver Medios).

48.1.126. Updaters

Un updater mueve un objeto en cada fotograma sin programar un `Anim`. Los presets de `Updater` se evalúan en Rust; se asocian con `Drawable.add_updater` y se quitan con `Drawable.remove_updater`. Los objetos generados por updaters siguen ocultos hasta que su propia animación de entrada entra en `scene.play`.

48.1.127. Updater.orbit

FÁBRICA `orbit(cx: float, cy: float, radius: float, speed: float) -> Updater`

Coloca el objeto en el círculo según el tiempo transcurrido desde que se asocia.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`cx, cy`	`float`	obligatorio	Centro de la órbita.
`radius`	`float`	obligatorio	Radio.
`speed`	`float`	obligatorio	Velocidad angular en radianes por segundo.

Devuelve: `Updater` — Órbita circular.

Ejemplo:

```
planet = scene.geometry.dot(0.12).fill(BLUE)
planet.add_updater(Updater.orbit(0, 0, 2, 1.0))
scene.wait(2)
```

48.1.128. Updater.advance_x

FÁBRICA `advance_x(speed: float) -> Updater`

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`speed`	`float`	obligatorio	Unidades por segundo hacia la derecha; negativo va hacia la izquierda.

Devuelve: `Updater` — Avance horizontal constante.

48.1.129. Updater.bob

FÁBRICA `bob(amplitude: float, frequency: float) -> Updater`

Como `Updater.pulse` y `Updater.rotate` con un número, es una capa sobre la animación del objeto: función del tiempo desde que se añade, igual en un seek que en la reproducción, que sigue a `animate.move_to` en lugar de volver a la posición inicial. Las tres se combinan en un mismo objeto. `remove_updater()` la quita y el objeto vuelve a su posición.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`amplitude`	`float`	obligatorio	Desplazamiento vertical máximo.
`frequency`	`float`	obligatorio	Oscilaciones por segundo.

Devuelve: `Updater` — Oscilación vertical senoidal sumada a la posición del objeto.

48.1.130. Updater.rotate

FÁBRICA `rotate(speed: float | Parameter | Computed) -> Updater`

Con un número, el ángulo es la velocidad por el tiempo desde que se añade el updater. Con un `Parameter`, el ángulo es la integral de su valor desde que se añade el updater, así que un seek o una exportación desde la mitad caen en el mismo ángulo que la reproducción, y `remove_updater()` lo deja en el ángulo alcanzado. Para un `Parameter` movido por `drive_from_samples` o `add_updater_fn` se usa su valor de cada fotograma. Un `Computed` que depende solo del tiempo (`musica.level()`, `scene.noise`) se integra desde el segundo 0 con paso fijo, igual en reproducción y en seek; uno que lee un `Parameter` lanza `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`speed`	`float | Parameter | Computed`	obligatorio	Radianes por segundo. Con un `Parameter`, animarlo acelera o frena el giro sin saltos. Con un `Computed` del tiempo, como una señal de audio, gira a su valor.

Devuelve: `Updater` — Giro continuo alrededor de Z.

Ejemplo:

```
speed = scene.viz.parameter(0.5)
circle.add_updater(Updater.rotate(speed))
scene.play([speed.animate.set(6.0).duration(2)])  # el clímax: gira cada vez más rápido
scene.play([speed.animate.set(0.5).duration(1)])
```

48.1.131. Updater.pulse

FÁBRICA `pulse(min_scale: float, max_scale: float, frequency: float) -> Updater`

Una capa sobre la animación del objeto, como `Updater.bob`: se combina con `Updater.rotate` en el mismo objeto.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`min_scale, max_scale`	`float`	obligatorio	Factores de escala mínimo y máximo respecto de la escala del objeto.
`frequency`	`float`	obligatorio	Pulsos por segundo.

Devuelve: `Updater` — Escala que late entre dos factores.

Ejemplo:

```
boat.add_updater(Updater.advance_x(1.5))
fan.add_updater(Updater.rotate(3.0))
heart.add_updater(Updater.pulse(0.9, 1.1, 1.2))
scene.wait(2)
```

48.1.132. Updater.wiggle

FÁBRICA `wiggle(*, position: float | Parameter | Computed = 0.08, rotation: float | Parameter | Computed = 0.0, scale: float | Parameter | Computed = 0.0, frequency: float | Parameter | Computed = 2.0, octaves: int = 2, seed: int = 0) -> Updater`

Es una capa sobre la animación del objeto: empieza en cero, es función pura del tiempo de la línea de tiempo y se suma a `animate.move_to` y a otros clips en lugar de sustituirlos. Un seek cae en el mismo fotograma que la reproducción. Las amplitudes y la frecuencia aceptan un `Parameter`: animar la amplitud a 0 apaga el temblor poco a poco en lugar de cortarlo con `remove_updater()`, y cambiar la frecuencia cambia el ritmo sin saltos. También aceptan un `Computed` del tiempo, como `musica.pulse()`, para temblar con el sonido. Valores inválidos lanzan `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`position`	`float | Parameter | Computed`	`0.08`	Amplitud del ruido de posición, en unidades de escena.
`rotation`	`float | Parameter | Computed`	`0.0`	Amplitud de giro, en radianes.
`scale`	`float | Parameter | Computed`	`0.0`	Amplitud de escala, como fracción.
`frequency`	`float | Parameter | Computed`	`2.0`	Rapidez del temblor.

`octaves`	`int`	`2`	Capas de detalle, de 1 a 8.
`seed`	`int`	`0`	Semilla del ruido.

Devuelve: `Updater` — Temblor orgánico con semilla.

Ejemplo:

```
logo.add_updater(Updater.wiggle(position=0.08, rotation=0.03, seed=1))
scene.play([logo.animate.move_to(3, 0).duration(2)])  # sigue temblando mientras se mueve
```

Para atenuarlo, pasa la amplitud como `Parameter`:

```
shake = scene.viz.parameter(0.15)
logo.add_updater(Updater.wiggle(position=shake, frequency=3.0, seed=2))
scene.wait(1)
scene.play([shake.animate.set(0).duration(1.5)])  # el temblor se calma
```

48.1.133. Updater.oscillate

FÁBRICA `oscillate(channel: Literal["x", "y", "rotation", "scale", "opacity"], *, waveform: Literal["sine", "square", "triangle", "saw"] = "sine", frequency: float = 1.0, low: float = 0.0, high: float = 1.0, phase: float = 0.0) -> Updater`

En `x`, `y` y `rotation` el valor se suma al animado; en `scale` y `opacity` lo multiplica (los factores de opacidad deben estar en `[0, 1]`). Como `wiggle`, es función pura del tiempo y se combina con animaciones.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`channel`	`"x" | "y" | "rotation" | "scale" | "opacity"`	obligatorio	Canal que oscila.
`waveform`	`"sine" | "square" | "triangle" | "saw"`	`"sine"`	Forma de la onda.
`frequency`	`float`	`1.0`	Ciclos por segundo.
`low, high`	`float`	`0.0, 1.0`	Valores entre los que oscila; toda onda empieza en `low`.

`phase`	`float`	`0.0`	Desfase en ciclos.

Devuelve: `Updater` — Una capa periódica.

Ejemplo:

```
light = scene.geometry.circle(0.3).fill(GOLD)
light.add_updater(Updater.oscillate("opacity", waveform="triangle", frequency=0.5, low=0.4,
high=1.0))
scene.wait(2)
```

48.1.134. Updater.dash_flow

FÁBRICA `dash_flow(speed: float = 0.5) -> Updater`

Hace fluir los guiones de todos los trazos del objeto a lo largo de su camino, como hormigas en marcha o el flujo de una tubería. Se suma al `dash_offset` autorado y es función pura del tiempo, así que los seeks y la exportación coinciden con la reproducción. `remove_updater()` lo detiene donde está. El trazo necesita guiones (`StrokeStyle(dashes=...)`). Una velocidad no finita lanza `ValueError`.

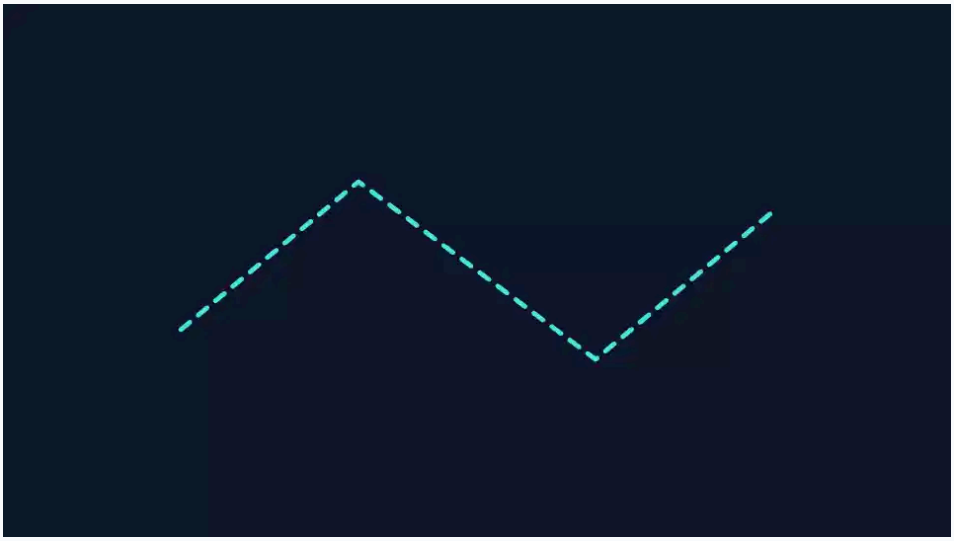
Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`speed`	`float`	`0.5`	Unidades de escena por segundo en el sentido del camino; un valor negativo lo invierte.

Devuelve: `Updater` — Un flujo de guiones.

Ejemplo:

```
from gaanim import CYAN, Scene, StrokeStyle, Updater
scene = Scene(frame=(16, 9), background="#0f172a")
pipe = scene.geometry.polyline([(-5, -1), (-2, 1.5), (2, -1.5), (5, 1)])
pipe.no_fill().stroke_style(StrokeStyle(CYAN, 0.08, cap="round", dashes=[0.2, 0.18]))
pipe.add_updater(Updater.dash_flow(speed=0.8))
scene.wait(2)
scene.render()
```



48.1.135. Drawable.add_updater_fn

MÉTODO `add_updater_fn(callback, *, reset=None, fixed_dt=None) -> Drawable`

Pasa `reset` y `fixed_dt` juntos para física o cualquier estado incremental: tras un seek y durante la exportación, Gaanim restaura la posición inicial, llama a `reset()` y repite los subpasos constantes, así el resultado es determinista. El updater empieza en el cursor donde se declara, no en segmentos anteriores. Sin esa pareja, la función está pensada para comportamientos ligeros por fotograma o de tiempo absoluto. Coordenadas inválidas o una excepción detienen el updater.

Parámetros:

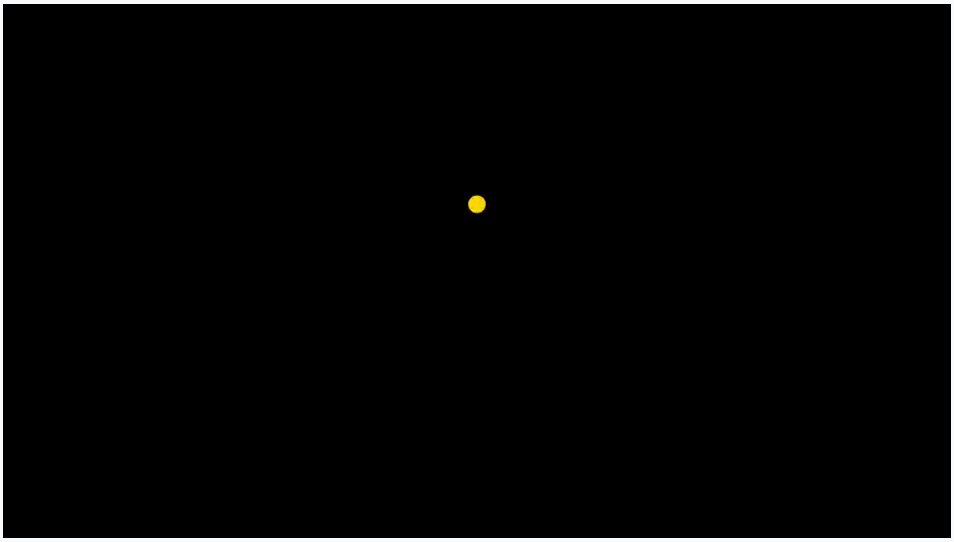
Nombre	Tipo	Predeterminado	Descripción
`callback`	`Callable`	obligatorio	`callback((x, y, z), dt, elapsed)` devuelve la nueva posición local.
`reset`	`Callable | None`	`None`	Restaura todo el estado de Python que guarda una simulación.
`fixed_dt`	`float | None`	`None`	Paso de simulación positivo, en segundos.

Devuelve: `Drawable` — El mismo objeto.

Ejemplo:

```
from gaanim import BLACK, GOLD, Scene

scene = Scene(frame=(16, 9), background=BLACK)
ball = scene.geometry.dot(0.15).fill(GOLD).move_to(0, 1.125)
state = {"velocity": 0.0}

def reset():
    state["velocity"] = 0.0

def step(pos, dt, elapsed):
```

```
    x, y, z = pos
    state["velocity"] -= 3.0 * dt
    y += state["velocity"] * dt
    if y < -1.125:
        y = -1.125
        state["velocity"] *= -0.8
    return (x, y, z)

ball.add_updater_fn(step, reset=reset, fixed_dt=1 / 240)
scene.wait(3.0)
scene.render()
```



48.1.136. Drawable.drive_from_samples

MÉTODO `drive_from_samples(times, values, property="x", *, interpolation="linear", scale=1.0, offset=0.0, colors=None, domain=None) -> Drawable`

Mueve el canal como función pura del tiempo, evaluada en Rust, sin llamar a Python en cada fotograma. Los ejes de traslación y `rotation` son relativos a la pose declarada (`base + offset + scale * muestra`); `scale`, `opacity` y `signal` son absolutos. Fuera de la serie se mantiene la primera o la última muestra. Los tiempos cuentan desde el cursor donde se llama: una serie declarada después de `scene.wait(2.0)` reproduce su muestra `t = 0` a los dos segundos. Cada canal es independiente; volver a mover el mismo canal lo reemplaza. `remove_updater()` lo quita. `Parameter.drive_from_samples(times, values, *, ...)` hace lo mismo con la señal de un parámetro.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`times, values`	`Sequence[float] | Sequence[tuple[float, float]]`	obligatorio	Series de igual longitud; los

			tiempos deben ser finitos y no decrecientes. Con `"xy"`, los valores son parejas `(x, y)`.
`property`	`"x" | "y" | "xy" | "z" | "rotation" | "scale" | "opacity" | "signal" | "fill"`	`"x"`	Canal que se mueve. `"xy"` mueve los dos ejes de traslación como canales `"x"` e `"y"`. `"fill"` colorea el relleno del objeto y de los miembros de un grupo que lo tengan.

`interpolation`	`"linear" | "step"`	`"linear"`	Interpolación entre muestras.
`scale, offset`	`float`	`1.0, 0.0`	Transformación aplicada a cada muestra.
`colors`	`Sequence[ColorLike] | None`	`None`	Solo con `"fill"`, y obligatorio: al menos dos colores, repartidos por igual sobre `domain`.
`domain`	`tuple[float, float] | None`	`None`	Solo con `"fill"`: valores que corresponden al primer y al último color, tras `scale` y `offset`. Por defecto, el rango de la serie. Fuera

de él se usa el color del extremo.

Devuelve: `Drawable` — El mismo objeto.

Ejemplo:

```
from gaanim import CYAN, Scene

scene = Scene(frame=(16, 9), background="#0f172a")
times = [i * 0.02 for i in range(len(accel))]
building = scene.geometry.rounded_rect(2, 4.5, 0.125).fill(CYAN).move_to(-2.5, -1.5)
# El edificio oscila con el registro medido; el seek es determinista.
building.drive_from_samples(times, accel, "x", scale=6.5)
scene.play([building.animate.grow_from_center()])
scene.wait(4.0)
```

Para un ejemplo con varilla, cota y estela, consulta `examples/mechanics/simulations.py` en el repositorio de Gaanim.

48.1.137. Partículas

`scene.fx.particles` crea un emisor de partículas determinista: chispas, polvo, humo, nieve o estallidos. Cada partícula tiene forma cerrada: la partícula `i` del flujo continuo nace en `inicio + i / rate`, las de una ráfaga nacen juntas en su instante, y su vida, velocidad, dirección, tamaño, color y giro salen de un generador con semilla indexado por `(seed, i)`. Con gravedad y arrastre lineal `k`, la posición es `p0 + v0·(1 − e^(−k·t))/k` más el término exacto de la gravedad. Así un seek muestra exactamente lo mismo que la reproducción, la exportación coincide con la vista previa y cada fotograma solo evalúa las partículas vivas en ese instante.

El emisor es un `Drawable`: su opacidad, sus fundidos y sus transformaciones se aplican a todas sus partículas. Las partículas se dibujan en coordenadas de escena, así que mover el emisor se hace con `Emitter.at(...)`, no con `move_to`.

48.1.138. Fx.particles

MÉTODO `scene.fx.particles(emitter=None, *, rate=30.0, duration=None, lifetime=(0.6, 1.2), speed=(1.0, 2.0), direction=pi/2, spread=tau, gravity=(0, 0), drag=0.0, size=(0.03, 0.06), size_end=1.0, fade=0.3, spin=0.0, flutter=0.0, shape="circle", color=WHITE, seed=0) -> Drawable`

Los rangos aceptan un número o un par `(low, high)` y cada partícula elige dentro de ellos. Las ráfagas se añaden con `Drawable.burst` y `animate.burst`. Valores no finitos o fuera de rango, más de 16 colores, `rate * lifetime` por encima de 100 000, una forma desconocida o un objeto de otra escena lanzan `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`emitter`	`Emitter | (x, y) | Drawable | None`	`(0, 0)`	De dónde salen las partículas. Un punto o un objeto equivalen a `Emitter.point().at(...)`.
`rate`	`float`	`30.0`	Partículas por segundo desde el cursor de la declaración; `0` solo emite ráfagas.
`duration`	`float | None`	`None`	Segundos de emisión continua; `None` no se detiene.
`lifetime`	`float | (low, high)`	`(0.6, 1.2)`	Vida de cada partícula en segundos; positiva.
`speed`	`float | (low, high)`	`(1.0, 2.0)`	Velocidad inicial en unidades por segundo.
`direction`	`float`	`pi/2`	Dirección media en radianes (hacia arriba).
`spread`	`float`	`tau`	Ángulo completo del cono de direcciones.
`gravity`	`(x, y)`	`(0, 0)`	Aceleración constante, en unidades por segundo al cuadrado.
`drag`	`float`	`0.0`	Arrastre lineal por segundo;

			frena cada partícula hacia su velocidad terminal.
`size`	`float | (low, high)`	`(0.03, 0.06)`	Diámetro o lado, en unidades de escena.
`size_end`	`float`	`1.0`	Factor de tamaño al final de cada vida; `0` encoge hasta desaparecer.
`fade`	`float`	`0.3`	Fracción final de cada vida en la que la partícula se desvanece, entre 0 y 1.
`spin`	`float | (low, high)`	`0.0`	Velocidad de giro en radianes por segundo; un número gira en ambos sentidos hasta ese valor.
`flutter`	`float`	`0.0`	Amplitud del vaivén lateral, en unidades.
`shape`	`str`	`"circle"`	`"circle"`, `"square"`, `"rect"` (tira 2:1), `"triangle"` o `"streak"` (chispa estirada según su velocidad).

`color`	`ColorLike | Sequence[ColorLike] | Brush`	`WHITE`	Un color, una lista de la que cada partícula elige uno, o un degradado `Brush` a lo largo del cual cada partícula toma un color.
`seed`	`int`	`0`	La misma semilla da las mismas partículas.

Devuelve: `Drawable` — El emisor.

Ejemplo:

```
import math
from gaanim import GOLD, ORANGE, RED, Brush, Emitter, Scene

scene = Scene(frame=(16, 9), background="#0b1020")
logo = scene.geometry.star(5, 0.6, 0.25).fill(GOLD)
sparks = scene.fx.particles(
    emitter=Emitter.circle(radius=0.2).at(logo),
    rate=60, lifetime=(0.6, 1.2), speed=(2.0, 4.0), spread=math.tau,
    gravity=(0, -3), drag=0.8, size=(0.03, 0.08), shape="streak",
    color=Brush.linear([GOLD, ORANGE, RED], start=(0, 0), end=(1, 0)), seed=9,
)
scene.play([logo.animate.move_to(3, 0).duration(1.5)])
scene.play([sparks.animate.burst(80)])
scene.render()
```

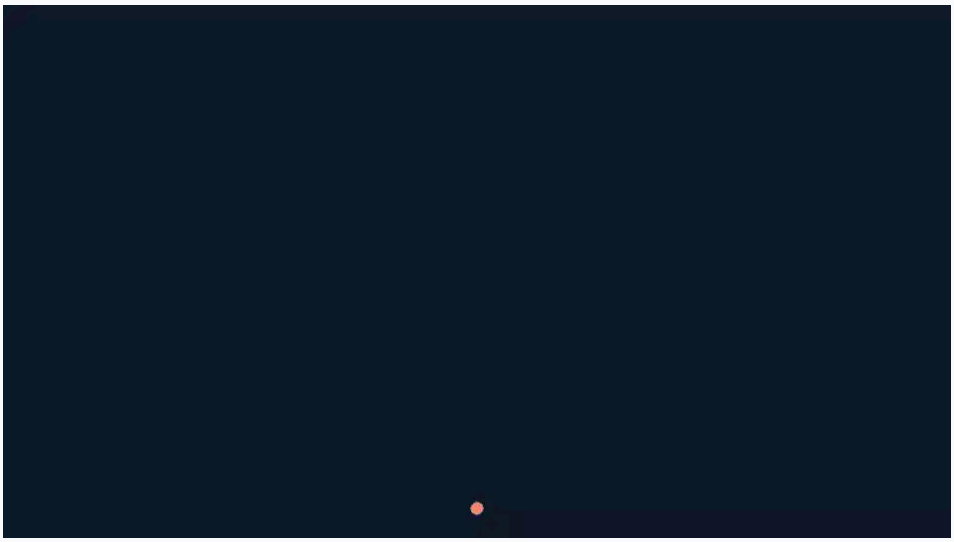
48.1.139. Fx.confetti

MÉTODO `scene.fx.confetti(origin=None, count=120, *, seed=0, colors=None, speed=(6, 10), direction=pi/2, spread=0.9, gravity=(0, -6), lifetime=(2.6, 3.6), size=(0.12, 0.2)) -> Drawable`

Un *preset* de `Fx.particles`: lanza `count` tiras de papel desde `origin` en el cursor, hacia `direction` dentro de un cono de `spread` radianes. Giran, se balancean y caen con `gravity` y arrastre, en una paleta festiva salvo que pases `colors`. Con `count=0` no lanza nada hasta `burst` o `animate.burst`, que lanzan más del mismo confeti. La misma `seed` lanza el mismo confeti. Un `count` por encima de 100 000 o valores no válidos lanzan `ValueError`.

Ejemplo:

```
from gaanim import Scene

scene = Scene(frame=(16, 9), background="#101828")
confetti = scene.fx.confetti(origin=(0, -4), count=120, seed=2)
scene.wait(3)
scene.render()
```



48.1.140. Emitter

CLASE `Emitter.point() · circle(radius, *, edge=False) · rect(width, height) · line(length, angle=0.0) · at(target, *, offset=(0, 0))`

De dónde salen las partículas de `Fx.particles`. `point`: todas del mismo punto. `circle`: puntos uniformes dentro de un disco, o sobre su borde con `edge`. `rect`: puntos uniformes dentro de un rectángulo. `line`: puntos uniformes sobre un segmento girado `angle` radianes. `at` devuelve el mismo emisor centrado en un punto `(x, y)` o siguiendo un objeto, desplazado `offset`: cada partícula sale de donde estaba el objeto al nacer ella, así que un objeto que se mueve deja una estela. Dimensiones negativas o no finitas lanzan `ValueError`; un destino que no sea un punto ni un objeto lanza `TypeError`.

Ejemplo:

```
snow = scene.fx.particles(Emitter.rect(16, 0.1).at((0, 4.6)), direction=-1.57, spread=0.3,
speed=(0.5, 1.0), lifetime=(8, 10), rate=20, fade=0.1)
print(Emitter.circle(0.2, edge=True).at((1, 2)))

Emitter(Circle { radius: 0.2, edge: true } at (1, 2))
```

48.1.141. Drawable.burst

MÉTODO `burst(count) -> Drawable`

Emite `count` partículas a la vez desde un emisor de `scene.fx` en el cursor, sin moverlo; antes del primer `scene.play` la ráfaga ocurre en el instante de la declaración. Un `count` menor que 1 o mayor que 100 000, o un objeto que no sea un emisor, lanzan `ValueError`.

Ejemplo:

```
dust = scene.fx.particles((0, -2), rate=0, speed=(0.5, 1.5), lifetime=(1, 2))
dust.burst(40)
scene.wait(2)
```

48.1.142. Anim.burst

MÉTODO `animate.burst(count) -> Anim`

Emite `count` partículas a la vez cuando empieza la animación. Si no fijas `duration`, la animación dura la vida más larga de las partículas, así que el cursor espera a que se apaguen. No se combina con destinos de propiedad ni con otro efecto en el mismo `Anim`. Los mismos errores que `Drawable.burst`.

Ejemplo:

```
sparks = scene.fx.particles((0, 0), rate=0, speed=(2, 4), color=[GOLD, RED], seed=9)
scene.play([sparks.animate.burst(80)])
```

48.1.143. Falloffs: influencia por instancia

Un `Falloff` es un valor por instancia calculado a partir de su posición en el grupo, su distancia a un objetivo o un ruido con semilla. `Drawable.drive` lo conecta a un canal de todos los miembros de un grupo, así que un solo movimiento del cursor escala, tiñe o gira cientos de elementos. Se evalúa en Rust por miembro y fotograma, sin llamar a Python, y es función pura del tiempo de la línea de tiempo: un seek cae en el mismo fotograma que la reproducción. Los valores autorados no se tocan: el efecto se suma o multiplica por encima, como `Updater.wiggle`.

Los falloffs se combinan con `+`, `-` y `*` (con números u otros falloffs), `maximum` y `minimum`, como los modos de combinación de Cavalry.

48.1.144. Falloff.distance

FÁBRICA `distance(target: Drawable | tuple[float, float], *, radius: float = 2.0, falloff: Literal["linear", "smooth", "sharp", "round"] = "smooth") -> Falloff`

Un `radius` no positivo o no finito, o una forma desconocida, lanza `ValueError`; un objetivo de otro tipo lanza `TypeError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`target`	`Drawable | tuple[float, float]`	obligatorio	Objetivo: un objeto, que se sigue allí donde lo lleven sus animaciones, o un punto `(x, y)`.
`radius`	`float`	`2.0`	Distancia, en unidades de escena, a la que la influencia llega a 0.
`falloff`	`"linear" | "smooth" | "sharp" | "round"`	`"smooth"`	Forma de la caída: lineal, suave (smoothstep), concentrada junto al objetivo o que llega lejos.

Devuelve: `Falloff` — 1 en el objetivo y 0 a `radius` y más allá.

Ejemplo:

```
near = Falloff.distance(cursor, radius=2.0, falloff="smooth")
```

48.1.145. Falloff.index

FÁBRICA `index(*, easing: Optional[Easing] = None, reverse: bool = False) -> Falloff`

Un grupo de un solo miembro da 0.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`easing`	`Easing | None`	`None`	Curva que da forma a la rampa.
`reverse`	`bool`	`False`	Cuenta desde el último miembro.

Devuelve: `Falloff` — 0 para el primer miembro y 1 para el último.

Ejemplo:

```
fade_in_order = Falloff.index(easing=Easing.SMOOTH).remap(0.2, 1.0)
```

48.1.146. Falloff.linear

FÁBRICA `linear(start: Drawable | tuple[float, float], end: Drawable | tuple[float, float], *, falloff: Literal["linear", "smooth", "sharp", "round"] = "linear") -> Falloff`

Los miembros antes de `start` reciben 0 y los que pasan de `end` reciben 1: al mover un extremo, una ola cruza el grupo.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`start, end`	`Drawable | tuple[float, float]`	obligatorio	Extremos de la línea, objetos o puntos.
`falloff`	`str`	`"linear"`	Forma de la rampa, como en `Falloff.distance`.

Devuelve: `Falloff` — 0 en `start`, 1 en `end` y proporcional entre ambos.

Ejemplo:

```
wave = Falloff.linear((-6, 0), (6, 0), falloff="smooth")
```

48.1.147. Falloff.noise

FÁBRICA `noise(*, frequency: float = 0.5, scale: float = 0.5, octaves: int = 1, seed: int = 0) -> Falloff`

Valores fuera de rango lanzan `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`frequency`	`float`	`0.5`	Rapidez con que el campo se desplaza en el tiempo.
`scale`	`float`	`0.5`	Ciclos de ruido por unidad de escena: los vecinos reciben valores parecidos.
`octaves`	`int`	`1`	Capas de detalle, de 1 a 8.
`seed`	`int`	`0`	Semilla: la misma da el mismo campo.

Devuelve: `Falloff` — Ruido simplex en `[0, 1]` sobre la posición de cada miembro y el tiempo.

Ejemplo:

```
sway = Falloff.noise(frequency=0.4, seed=2).remap(-0.3, 0.3)
```

48.1.148. Falloff.constant

FÁBRICA `constant(value: float) -> Falloff`

Sirve para combinar: `near.maximum(Falloff.constant(0.3))`. Un valor no finito lanza `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`value`	`float`	obligatorio	Valor finito.

Devuelve: `Falloff` — El mismo valor para todos los miembros.

Ejemplo:

```
half = Falloff.constant(0.5)
```

48.1.149. Falloff.source

FÁBRICA `source(value: Computed) -> Falloff`

Lleva al `drive` una señal del tiempo, por ejemplo para que el color siga al audio. Un `Computed` que lee un `Parameter` lanza `ValueError`. Ver Audio.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`value`	`Computed`	obligatorio	Un `Computed` que depende solo del tiempo: una señal de audio, `scene.noise` o `scene.time`.

Devuelve: `Falloff` — Su valor, el mismo para todos los miembros.

Ejemplo:

```
bajo = scene.media.audio("assets/ritmo.ogg").band(20, 150, smoothing=0.5)
dots.drive("fill", Falloff.source(bajo).gradient(BLUE, GOLD))
```

48.1.150. Falloff.remap

MÉTODO `remap(low, high) -> Falloff`

`near.remap(1.0, 1.8)` escala de 1 a 1.8 conforme el miembro se acerca al objetivo. `Falloff.invert()` devuelve `1 - valor`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`low, high`	`float`	obligatorio	Valores finitos: 0 se convierte en `low` y 1 en `high`.

Devuelve: `Falloff` — Un falloff nuevo.

Ejemplo:

```
grow = near.remap(1.0, 1.8)
far = near.invert()
```

48.1.151. Falloff.gradient

MÉTODO `gradient(*colors) -> FalloffColor`

Menos de dos colores lanzan `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`colors`	`ColorLike`	obligatorio	Al menos dos colores, repartidos a partes iguales: 0 es el primero y 1 el último.

Devuelve: `FalloffColor` — El falloff convertido en color, para `drive(\"fill\", ...)`.

Ejemplo:

```
heat = near.gradient(BLUE, GOLD)
```

48.1.152. Falloff.maximum / minimum

MÉTODO `maximum(other) -> Falloff · minimum(other) -> Falloff`

Un operando de otro tipo lanza `TypeError`. `a + b`, `a - b` y `a * b` combinan igual.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`other`	`Falloff | float`	obligatorio	Con quien se combina.

Devuelve: `Falloff` — El mayor o el menor de los dos valores, miembro a miembro.

Ejemplo:

```
both = a.maximum(b)
sum_of = a + b
```

48.1.153. Drawable.drive

MÉTODO `drive(channel, value) -> None`

Sobre un grupo (por ejemplo el de `scene.geometry.duplicate`) actúa sobre cada miembro; sobre un objeto suelto, sobre él mismo. Varios `drive` sobre un canal se combinan. `clear_drive()` los termina en el cursor. Un canal desconocido lanza `ValueError`, un valor del tipo equivocado lanza `TypeError` y un objetivo de otra escena lanza `ValueError`.

Parámetros:

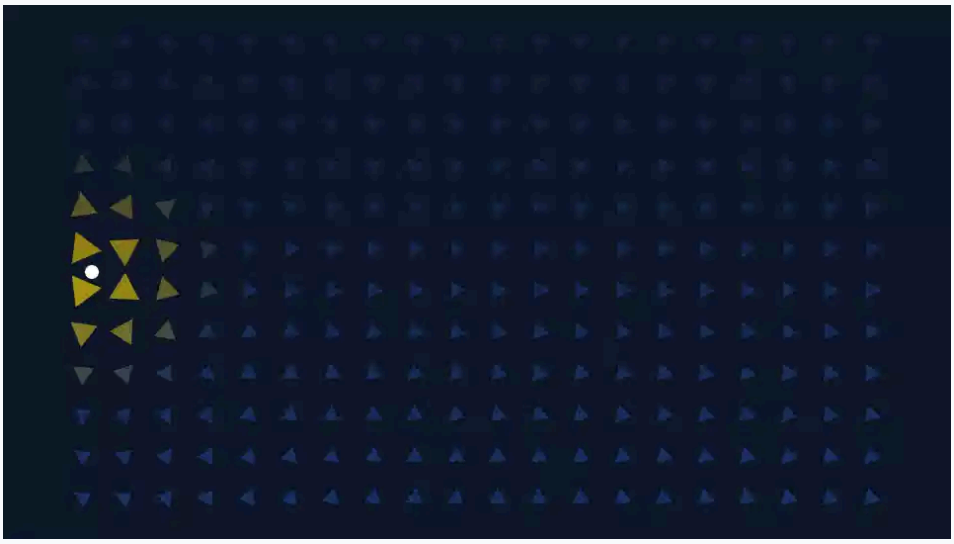
Nombre	Tipo	Predeterminado	Descripción
`channel`	`"scale" | "rotation" | "opacity" | "x" | "y" | "fill"`	obligatorio	Canal: `scale` y `opacity` multiplican el valor autorado, `rotation` suma radianes, `x` e `y` suman unidades de escena y `fill` sustituye el color de relleno.
`value`	`Falloff | FalloffColor`	obligatorio	Un `Falloff` para los canales numéricos; el `FalloffColor` de `Falloff.gradien`

para `fill`.

Devuelve: `None` — Se aplica desde el cursor de la línea de tiempo.

Ejemplo:

```
import math
from gaanim import BLUE, GOLD, WHITE, Distribution, Easing, Falloff, Scene

scene = Scene(frame=(16, 9), background="#0f172a")
tip = scene.geometry.regular_polygon(3, 0.17).fill(BLUE).no_stroke()
grid = scene.geometry.duplicate(tip, Distribution.grid(20, 12, 0.7))
cursor = scene.geometry.dot(0.12).fill(WHITE).move_to(-6.5, 0)

near = Falloff.distance(cursor, radius=2.5, falloff="smooth")
grid.drive("scale", near.remap(1.0, 1.9))
grid.drive("fill", near.gradient(BLUE, GOLD))
grid.drive("opacity", Falloff.index(easing=Easing.SMOOTH).remap(0.35, 1.0))
grid.look_at(cursor, offset=-math.pi / 2)  # la punta mira al cursor

scene.play([cursor.animate.move_to(6.5, 0).duration(2.5).easing(Easing.SMOOTH)])
scene.render()
```



48.1.154. Drawable.look_at

MÉTODO `look_at(target, *, offset=0.0) -> None`

Gira el objeto, o cada miembro de un grupo, para que su eje x apunte al objetivo. Se suma al giro autorado y a los `drive("rotation", ...)`. Un objetivo de otro tipo lanza `TypeError` y uno de otra escena `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción

`target`	`Drawable | tuple[float, float]`	obligatorio	Hacia dónde mira: un objeto, que se sigue allí donde vaya, o un punto.
`offset`	`float`	`0.0`	Radianes que se suman, para formas que apuntan hacia otro lado (un triángulo apunta hacia arriba: usa `-math.pi / 2`).

Devuelve: `None` — Se aplica desde el cursor de la línea de tiempo.

Ejemplo:

```
needles.look_at(cursor)
scene.render()
```

48.1.155. Drawable.clear_drive

MÉTODO `clear_drive() -> None`

Los `drive` dejan de sumar su desplazamiento, así que los valores autorados vuelven a verse desde ese instante. Un `look_at` deja el objeto girado como estaba en el cursor: ese giro, medido sobre las posiciones autoradas, pasa a su rotación (si el objetivo solo lo mueve un updater, puede quedar un poco desviado). Lo anterior no cambia.

Devuelve: `None` — Termina en el cursor todos los `drive` y `look_at` del objeto o de los miembros del grupo.

Ejemplo:

```
scene.wait(1)
dots.clear_drive()
scene.render()
```

48.1.156. Transformaciones 3D

EXPERIMENTAL

La API 3D es experimental: todavía tiene errores conocidos y puede cambiar entre versiones. Úsala para explorar, pero no la tomes como estable.

Los mismos destinos en tres ejes, para objetos en el espacio de una cámara en perspectiva. Las rotaciones de Euler usan orden XYZ y radianes.

48.1.157. Anim.move_to_3d

MÉTODO `move_to_3d(x: ScalarSource, y: ScalarSource, z: ScalarSource) -> Anim`

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`x, y, z`	`ScalarSource`	obligatorio	Posición de destino.

Devuelve: `Anim` — Movimiento a una posición 3D.

48.1.158. Anim.shift_by_3d

MÉTODO `shift_by_3d(dx: float, dy: float, dz: float) -> Anim`

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`dx, dy, dz`	`float`	obligatorio	Desplazamiento en unidades de escena.

Devuelve: `Anim` — Movimiento relativo en 3D.

48.1.159. Anim.rotate_to_3d

MÉTODO `rotate_to_3d(x: ScalarSource, y: ScalarSource, z: ScalarSource) -> Anim`

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`x, y, z`	`ScalarSource`	obligatorio	Orientación de Euler XYZ absoluta, en radianes.

Devuelve: `Anim` — Giro a una orientación.

48.1.160. Anim.rotate_by_3d

MÉTODO `rotate_by_3d(axis: Literal["x", "y", "z"], radians: float) -> Anim`

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`axis`	`"x" | "y" | "z"`	obligatorio	Eje de giro; otro valor lanza `ValueError`.
`radians`	`float`	obligatorio	Ángulo relativo.

Devuelve: `Anim` — Giro relativo alrededor de un eje.

48.1.161. Anim.scale_to_3d

MÉTODO `scale_to_3d(x: ScalarSource, y: ScalarSource, z: ScalarSource) -> Anim`

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`x, y, z`	`ScalarSource`	obligatorio	Escala absoluta por eje.

Devuelve: `Anim` — Cambio de escala por eje.

48.1.162. Anim.scale_by_3d

MÉTODO `scale_by_3d(x: float, y: float, z: float) -> Anim`

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`x, y, z`	`float`	obligatorio	Factores que multiplican la escala de cada eje.

Devuelve: `Anim` — Cambio de escala relativo por eje.

48.1.163. Anim.material

MÉTODO `material(material: Material3D) -> Anim`

Solo en una `Primitive3D`. Color, color emisivo, rugosidad, metalicidad e intensidad de emisión se interpolan de forma determinista y los extremos exactos se restauran al buscar. En una malla, `write()` lanza `TypeError`: usa `create()`, que crece desde el centro mientras aparece.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`material`	`Material3D`	obligatorio	Material de destino.

Devuelve: `Anim` — Interpolación del material.

Ejemplo:

```
cube = scene.geometry.cube(1.5).material(Material3D.matte(BLUE))
scene.play(cube.animate.material(Material3D.metal(GOLD)).rotate_by_3d("y", 1.2).duration(1.2))
```

48.1.164. Transiciones entre segmentos

`Transition` describe el paso de un segmento al siguiente en `scene.segment(...)` o `scene.link(...)`. Todas salvo `cut` aceptan `easing=`, con cualquier `Easing`, incluidos los resortes que se pasan del destino, y todas aceptan `overlay=`, un `Overlay` dibujado encima del corte. Ninguna de las dos opciones cambia la duración de los segmentos. Sin `easing`, `cross_fade`, `fade_through` y `zoom_through` avanzan de forma lineal, `morph` y `magic_move` se mueven con `Easing.SMOOTH` y funden de forma lineal, y los revelados vectoriales (`wipe`, `clock_wipe`, `iris`, `blinds`, `push` y `slide`) usan `Easing.SMOOTH`. Con un resorte o una curva de retroceso, el movimiento se pasa del destino y vuelve, pero los fundidos siguen la curva sin retroceder: una vez pasado el destino, el fundido ha terminado.

Todas aceptan también `sound=`, la ruta de un efecto de sonido (relativa a la carpeta de assets) que suena una vez, con volumen 1, cuando empieza la transición, es decir, al inicio del segmento al que

entra. Se mezcla como `scene.media.audio` en la vista previa y en MP4/WebM y no cambia ninguna duración; consulta Audio.

Los revelados vectoriales recortan ambos segmentos con caminos animados dentro del marco visible de la cámara: el entrante se ve dentro de la región revelada y el saliente en el resto; si los fondos son distintos, el entrante se revela con la misma forma. Son geometría vectorial, sin texturas intermedias, así que la transición es nítida a cualquier resolución y un seek reproduce exactamente el mismo fotograma. Los objetos persistentes (`scene.persist`) no se recortan ni se desplazan.

48.1.165. Transition.cut

FÁBRICA `cut(*, overlay: Optional[Overlay] = None, sound: Optional[str] = None) -> Transition`

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`overlay`	`Overlay | None`	`None`	Capa dibujada sobre el corte.

Devuelve: `Transition` — Cambio instantáneo.

48.1.166. Transition.cross_fade

FÁBRICA `cross_fade(duration: float, *, easing: Optional[Easing] = None, overlay: Optional[Overlay] = None, sound: Optional[str] = None) -> Transition`

El segmento saliente se desvanece mientras aparece el entrante.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`duration`	`float`	obligatorio	Segundos, positivo.
`easing, overlay`	`Easing | None, Overlay | None`	`None`	Comunes a todas las transiciones.

Devuelve: `Transition` — Fundido cruzado.

Ejemplo:

```
scene.segment("detalle", Transition.cross_fade(0.4))
scene.wait(1)
```

48.1.167. Transition.fade_through

FÁBRICA `fade_through(duration: float, color: ColorLike, *, easing: Optional[Easing] = None, overlay: Optional[Overlay] = None, sound: Optional[str] = None) -> Transition`

Funde a `color` durante la primera mitad y desde él al segmento entrante en la segunda.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`duration`	`float`	obligatorio	Segundos, positivo.
`color`	`Color`	obligatorio	Color intermedio.

Devuelve: `Transition` — Fundido a través de un color.

Ejemplo:

```
scene.segment("capítulo 2", Transition.fade_through(1.0, BLACK))
scene.wait(1)
```

48.1.168. Transition.slide

FÁBRICA `slide(duration: float, direction: str, *, easing: Optional[Easing] = None, overlay: Optional[Overlay] = None, sound: Optional[str] = None) -> Transition`

El segmento saliente queda quieto y el entrante lo cubre al recorrer un ancho o un alto de marco. Usa `push` para mover los dos. Otra dirección lanza `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`duration`	`float`	obligatorio	Segundos, positivo.
`direction`	`str`	obligatorio	Movimiento del marco entrante: `"left"`, `"right"`, `"up"` o `"down"`.

Devuelve: `Transition` — El entrante se desliza sobre el saliente.

Ejemplo:

```
scene.segment("detalle", Transition.slide(0.5, "left", sound="whoosh.wav"))
scene.wait(1)
```

48.1.169. Transition.push

FÁBRICA `push(duration: float, direction: str = "up", *, easing: Optional[Easing] = None, overlay: Optional[Overlay] = None, sound: Optional[str] = None) -> Transition`

El segmento entrante empuja al saliente fuera del marco: ambos se desplazan un ancho o un alto de marco y cada uno queda recortado a su propio marco.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción

`duration`	`float`	obligatorio	Segundos, positivo.
`direction`	`str`	`"up"`	Movimiento de ambos marcos: `"left"`, `"right"`, `"up"` o `"down"`.

Devuelve: `Transition` — Empuje.

Ejemplo:

```
scene.segment("siguiente", Transition.push(0.5, direction="up"))
scene.wait(1)
scene.segment("final", Transition.slide(0.5, "left", easing=Easing.spring(bounce=0.2)))
scene.wait(1)
```

48.1.170. Transition.shader

FÁBRICA `shader(source: str | os.PathLike[str], duration: float, *, uniforms: Optional[Mapping[str, float | Parameter | Variable | Computed | TimeInput]] = None, easing: Optional[Easing] = None, overlay: Optional[Overlay] = None, sound: Optional[str] = None) -> Transition`

Dibuja cada segmento por separado, con su fondo, y los mezcla en la GPU, como gl-transitions. La función lee `gaanim_from(uv)` (saliente), `gaanim_to(uv)` (entrante), `progress` (de 0 a 1, lineal salvo `easing=`), `gaanim_resolution()` y `gaanim_time()`; `uv` vale (0, 0) arriba a la izquierda del cuadro. Los objetos persistentes y el `overlay` quedan nítidos encima. WGSL o nombres de uniform inválidos lanzan `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`source`	`str | os.PathLike`	obligatorio	WGSL en línea o un archivo `.wgsl` (una ruta relativa se busca en el directorio de trabajo, luego en la

			carpeta `assets` del script y luego junto a él) que define `fn transition(uv: vec2<f32>) -> vec4<f32>`.
`duration`	`float`	obligatorio	Segundos, positivo.
`uniforms`	`Mapping[str, float | Parameter | Computed] | None`	`None`	Campos de `gaanim_uniforms`: los números son fijos; un `Parameter`, `Variable`, `Computed`, `scene.time` o una señal de audio de la misma escena se evalúa en cada fotograma de la

			transición, así que puede latir con el bombo. Los paquetes `.gaanim` guardan los valores.

Devuelve: `Transition` — Transición por shader.

Ejemplo:

```
mezcla = Transition.shader(
    "fn transition(uv: vec2<f32>) -> vec4<f32> { return mix(gaanim_from(uv), gaanim_to(uv),
progress); }",
    0.6,
)
scene.segment("despues", mezcla)
scene.wait(1)
```

48.1.171. Transition.preset

FÁBRICA `preset(name: Literal["cross_zoom", "directional_warp", "ripple", "glitch_displace", "luma"], duration: float, *, image: Optional[str | os.PathLike[str]] = None, invert: bool = False, easing: Optional[Easing] = None, overlay: Optional[Overlay] = None, sound: Optional[str] = None, **settings: float | int) -> Transition`

Un nombre o un ajuste desconocidos, `image` en otro preset o `luma` sin imagen lanzan `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`name`	`str`	obligatorio	`"cross_zoom"`, `"directional_warp"`, `"ripple"`, `"glitch_displace"` o `"luma"`.
`duration`	`float`	obligatorio	Segundos, positivo.

`image / invert`	`str | os.PathLike | None / bool`	`None / False`	Solo `luma`: mapa en escala de grises (reducido a 256 px como mucho; una ruta relativa se busca como en `source`); el segmento entrante aparece primero donde es más oscuro, o más claro con `invert=True`.
`**settings`	`float | int`	obligatorio	`bands` y `seed` son enteros; los demás, números reales. `cross_zoom`: `strength=0.4`. `directional_warp`: `dx=1`, `dy=0` (y hacia arriba), `smoothness=0.5`. `ripple`: `amplitude=100`, `speed=50`. `glitch_displace`: `strength=0.5`, `bands=24`, `seed=0`. `luma`: `softness=0.1`.

Devuelve: `Transition` — Transición por shader incorporada.

Ejemplo:

```
scene.segment("despues", Transition.preset("directional_warp", 0.8, dx=0, dy=1))
scene.wait(1)
```

48.1.172. Transition.zoom_through

FÁBRICA `zoom_through(duration: float, *, center: tuple[float, float] = (0.0, 0.0), max_zoom: float = 4.0, easing: Optional[Easing] = None, overlay: Optional[Overlay] = None, sound: Optional[str] = None) -> Transition`

Acerca la cámara a un punto del segmento saliente y la aleja en el entrante; útil cuando un detalle presenta la sección siguiente.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`duration`	`float`	obligatorio	Segundos, positivo.
`center`	`tuple[float, float]`	`(0.0, 0.0)`	Punto de la escena hacia el que se acerca.
`max_zoom`	`float`	`4.0`	Zoom máximo, en el corte.

Devuelve: `Transition` — Zoom a través del corte.

Ejemplo:

```
scene.segment("detalle", Transition.zoom_through(1.0, center=(2, 1), max_zoom=4))
scene.wait(1)
```

48.1.173. Transition.morph

FÁBRICA `morph(duration: float, *, pairs: Sequence[tuple[Drawable, Drawable]] = (), easing: Optional[Easing] = None, overlay: Optional[Overlay] = None, sound: Optional[str] = None) -> Transition`

Cada pareja comparte una caja que viaja de la caja del origen a la del destino, mientras el destino aparece en la primera mitad y el origen desaparece en la segunda: se lee como un solo objeto que cambia de lugar, tamaño, color y forma. El resto del contenido hace un fundido cruzado. Las parejas se resuelven al compilar, así que se pasan a `scene.link` cuando existen los dos segmentos. Una duración no positiva, un objeto emparejado consigo mismo o repetido en un lado lanzan `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`duration`	`float`	obligatorio	Segundos, positivo.
`pairs`	`Sequence[tuple[Drawable, Drawable]]`	`()`	Parejas `(origen, destino)`: el origen en el segmento saliente y el destino en el entrante.

Devuelve: `Transition` — Continuidad de objetos a través del corte.

Ejemplo:

```
overview = scene.segment("Resumen")
card = scene.geometry.rect(3, 2).fill(BLUE).move_to(-4, 0)
scene.wait(1)
detail = scene.segment("Detalle")
panel = scene.geometry.rect(12, 6).fill(BLUE)
scene.wait(1)
scene.link(overview, detail, Transition.morph(0.8, pairs=[(card, panel)]))
```

48.1.174. Transition.magic_move

FÁBRICA `magic_move(duration: float, *, key: Literal["name", "id"] = "name", easing: Optional[Easing] = None, overlay: Optional[Overlay] = None, sound: Optional[str] = None) -> Transition`

Como `Transition.morph`, pero las parejas se buscan al compilar entre los objetos de los dos segmentos que comparten clave: los de primer nivel y los miembros de sus grupos, donde un grupo con clave se mueve entero. Las claves repetidas se emparejan en orden de declaración y se ignoran los objetos ocultos al final del segmento saliente, como el origen de un `magic_move` terminado. El resto hace un fundido cruzado. Para claves calculadas con una función, usa `magic_move`. Una duración no positiva o una clave desconocida lanzan `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`duration`	`float`	obligatorio	Segundos, positivo.
`key`	`str`	`"name"`	`"name"` empareja los nombres de `Drawable.named` (o el `id` de una parte SVG sin nombre); `"id"`, solo los `id` SVG.
`easing, overlay`	`Easing | None, Overlay | None`	`None`	Comunes a todas las transiciones.

Devuelve: `Transition` — `Transition.morph` con parejas por clave.

Ejemplo:

```
scene.segment("v1")
scene.geometry.rect(3, 2).fill(BLUE).move_to(-4, 0).named("card")
scene.wait(1)
scene.segment("v2", transition=Transition.magic_move(0.8, key="name"))
scene.geometry.rect(8, 5).fill(GOLD).named("card")
scene.wait(1)
```

48.1.175. Transition.wipe

FÁBRICA `wipe(duration: float, direction: str = "left", feather: float = 0.1, *, easing: Optional[Easing] = None, overlay: Optional[Overlay] = None, sound: Optional[str] = None) -> Transition`

Un borde recto cruza el marco y descubre el segmento entrante. El borde suave es una rampa de alfa vectorial. Una duración no positiva, una dirección desconocida o un `feather` fuera de `[0, 1]` lanzan `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`duration`	`float`	obligatorio	Segundos, positivo.
`direction`	`str`	`"left"`	Sentido en que viaja el borde: `left`, `right`, `up`, `down` o una diagonal como `up_left`. Con `"left"` el revelado empieza en el lado derecho.
`feather`	`float`	`0.1`	Ancho del borde suave como fracción del recorrido, en `[0, 1]`; `0` da un borde duro.

Devuelve: `Transition` — Barrido lineal.

Ejemplo:

```
scene.segment("detalle", Transition.wipe(0.6, direction="left", feather=0.1))
```

48.1.176. Transition.clock_wipe

FÁBRICA `clock_wipe(duration: float, start_angle: float = 90.0, *, easing: Optional[Easing] = None, overlay: Optional[Overlay] = None, sound: Optional[str] = None) -> Transition`

Una aguja gira en sentido horario alrededor del centro del marco y deja ver el segmento entrante en el sector recorrido.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`duration`	`float`	obligatorio	Segundos, positivo.
`start_angle`	`float`	`90.0`	Grados en sentido antihorario desde +x; `90` empieza a las doce.

Devuelve: `Transition` — Barrido radial.

Ejemplo:

```
scene.segment("resumen", Transition.clock_wipe(0.8, start_angle=90))
```

48.1.177. Transition.iris

FÁBRICA `iris(duration: float, center: tuple[float, float] = (0.0, 0.0), shape: str | Drawable = "circle", *, easing: Optional[Easing] = None, overlay: Optional[Overlay] = None, sound: Optional[str] = None) -> Transition`

La forma crece desde `center` hasta que el segmento entrante ocupa todo el marco. Un `Drawable` se centra en su caja; se sigue dibujando en su propio segmento, así que conviene ocultarlo si solo sirve de plantilla. Uno sin camino vuelve al círculo. Un nombre desconocido lanza `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`duration`	`float`	obligatorio	Segundos, positivo.
`center`	`tuple[float, float]`	`(0.0, 0.0)`	Centro del iris en unidades de escena.
`shape`	`str | Drawable`	`"circle"`	`circle`, `diamond`, `square`, `star` o un `Drawable` cuyo contorno vectorial se usa como plantilla.

Devuelve: `Transition` — Iris que crece hasta cubrir el marco.

Ejemplo:

```
scene.segment("zoom", Transition.iris(0.7, center=(2, 1), shape="star"))
```

48.1.178. Transition.blinds

FÁBRICA `blinds(duration: float, count: int = 8, angle: float = 0.0, *, easing: Optional[Easing] = None, overlay: Optional[Overlay] = None, sound: Optional[str] = None) -> Transition`

Todas las lamas se abren a la vez y en la misma fracción.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`duration`	`float`	obligatorio	Segundos, positivo.
`count`	`int`	`8`	Número de lamas, entre 1 y 512.
`angle`	`float`	`0.0`	Inclinación de las lamas en grados; `0` son lamas horizontales que se abren hacia abajo.

Devuelve: `Transition` — Persiana veneciana.

Ejemplo:

```
scene.segment("datos", Transition.blinds(0.6, count=8, angle=0))
```

48.1.179. Overlay.flash

FÁBRICA `flash(color: Optional[ColorLike] = None, duration: float = 0.2) -> Overlay`

Cubre el marco con un color que sube durante la primera mitad y se apaga en la segunda, centrado en el punto medio de la transición (en `cut`, en el propio corte).

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`color`	`Color | None`	`None`	Color del destello; blanco si se omite.
`duration`	`float`	`0.2`	Segundos, positivo.

Devuelve: `Overlay` — Capa para `overlay=` de cualquier `Transition`.

Ejemplo:

```
scene.segment("impacto", Transition.cut(overlay=Overlay.flash(WHITE, 0.15)))
```

48.1.180. Overlay.light_leak

FÁBRICA `light_leak(seed: int = 0, hue: float = 0.1, duration: float = 0.8, intensity: float = 0.8) -> Overlay`

Manchas de luz suaves en modo pantalla (`screen`) que derivan por el marco alrededor del corte. Es función pura del tiempo y de la semilla, así que la previsualización, los seeks y la exportación coinciden. Valores fuera de rango lanzan `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`seed`	`int`	`0`	Semilla de la disposición de las manchas de luz.
`hue`	`float`	`0.1`	Tono base en vueltas: `0.1` naranja cálido, `0.6` azul.
`duration`	`float`	`0.8`	Segundos, positivo.
`intensity`	`float`	`0.8`	Brillo máximo, en `[0, 4]`.

Devuelve: `Overlay` — Capa para `overlay=`.

Ejemplo:

```
scene.segment("cálido", Transition.cross_fade(0.4, overlay=Overlay.light_leak(seed=2, hue=0.1)))
```

CAPÍTULO

49. Temas y colores

Colores, pinceles, fondos, postprocesado, temas y mapas de color

El aspecto de una escena sale de tres capas: el tema activo (colores semánticos, tipografía, reglas por tipo de objeto y tokens de espaciado), el fondo y los estilos que escribes en cada objeto, que siempre ganan.

```
from gaanim import BLUE, GOLD, Scene

scene = Scene(frame=(16, 9), theme="technical")
circle = scene.geometry.circle(1).fill(BLUE).stroke(GOLD, 0.05)
label = scene.text("Escena con tema", role="title").move_to(0, 2.5)
scene.play([circle.animate.create(), label.animate.write()])
scene.render()
```

Toda escena empieza con el tema `technical`: `Scene()` equivale a `Scene(theme="technical")`, `scene.canvas.theme` vale `"technical"` y el fondo es gris casi negro (`#121212`), con texto y ejes claros y formas rellenas con el color de acento. Otro tema (`Scene(theme=...)` o `scene.canvas.set_theme(...)`) sustituye al predeterminado, un fondo explícito (`Scene(background=...)`) gana al del tema y los colores que pones en cada objeto ganan a los del tema.

Para un lienzo sin tema, pasa `Scene(theme=None)` o llama a `scene.canvas.set_theme(None)`. Sin tema el fondo vuelve a ser blanco (salvo que hayas elegido uno) y el texto, las formas y los ejes sin estilo propio son blancos, así que dales color o usa un fondo que contraste; `gaanim check` avisa cuando no se verían. Sin tema, `scene.canvas.color(...)` y `scene.canvas.validate_theme()` lanzan `ValueError`, y los componentes como `badge` o `card` usan los colores de `technical`.

```
from gaanim import Scene

scene = Scene(frame=(16, 9), theme=None, background="white")
dot = scene.geometry.circle(1).fill("#2563eb")
label = scene.text("Sin tema", color="black").move_to(0, 2)
scene.render()
```

49.1.1. Colores

Todo argumento `ColorLike` acepta un `Color`, una cadena CSS Color 4 (`"#0f172a"`, `"tomato"`, `"oklch(62% 0.2 260)"`, `"rgb(37 99 235 / 65%)"`) o una tupla `(r, g, b)` o `(r, g, b, a)` de enteros de 0 a 255. Una sintaxis inválida o un componente fuera de rango lanzan `ValueError`.

Para anotar tus propias funciones, importa esos tipos desde `gaanim`: `ColorLike`; `Paint`, que también acepta un `Brush`; `ColorMapLike`, y `BackgroundLike`, que también acepta un `Background`. Funcionan en el comprobador de tipos y al ejecutar la escena. El resto de alias de los stubs (`Padding`, `TextAlign`, `Endpoint`…) se importa desde `gaanim.gaanim_core`.

```
from gaanim import Paint, Scene

def disc(scene: Scene, paint: Paint):
```

```
    return scene.geometry.circle(1).fill(paint)

scene = Scene(frame=(16, 9))
disc(scene, "#2563eb").move_to(-2, 0)
disc(scene, (220, 38, 38)).move_to(2, 0)
scene.render()
```

49.1.2. Color

CLASE `Color(value: str) | Color(r: int, g: int, b: int, a: int = 255)`

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`value`	`str`	obligatorio	Cualquier color CSS Color 4.
`r, g, b, a`	`int`	`a = 255`	Canales de 0 a 255.

Devuelve: `Color` — Un color RGBA inmutable.

Ejemplo:

```
navy = Color("#0f172a")
accent = Color("oklch(62.3% 0.214 259.815)")
translucent = Color("rgb(37 99 235 / 65%)")
purple = Color(128, 51, 204)
circle = scene.geometry.circle(1).fill(purple)
```

49.1.3. Color.from_hex

FÁBRICA `from_hex(value: str) -> Color`

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`value`	`str`	obligatorio	`"#rgb"`, `"#rrggbb"` o `"#rrggbbaa"`.

Devuelve: `Color`

49.1.4. Color.from_rgb

FÁBRICA `from_rgb(r: int, g: int, b: int) -> Color`

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`r, g, b`	`int`	obligatorio	Canales de 0 a 255.

Devuelve: `Color` — Un color opaco.

49.1.5. Color.from_rgba

FÁBRICA `from_rgba(r: int, g: int, b: int, a: int) -> Color`

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`r, g, b, a`	`int`	obligatorio	Canales de 0 a 255; `a = 0` es transparente.

Devuelve: `Color`

49.1.6. Color.from_hsl

FÁBRICA `from_hsl(h: float, s: float, l: float, a: float = 1.0) -> Color`

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`h`	`float`	obligatorio	Tono en grados.
`s, l`	`float`	obligatorio	Saturación y luminosidad, de 0 a 1.
`a`	`float`	`1.0`	Opacidad, de 0 a 1.

Devuelve: `Color`

49.1.7. Color.from_oklch

FÁBRICA `from_oklch(l: float, c: float, h: float, a: float = 1.0) -> Color`

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`l`	`float`	obligatorio	Luminosidad perceptual, de 0 a 1.
`c`	`float`	obligatorio	Croma.
`h`	`float`	obligatorio	Tono en grados.
`a`	`float`	`1.0`	Opacidad, de 0 a 1.

Devuelve: `Color` — Un color definido en un espacio perceptual, útil para paletas con luminosidad uniforme.

Ejemplo:

```
hexa = Color.from_hex("#ff8800")
rgb = Color.from_rgb(15, 23, 42)
glass = Color.from_rgba(255, 255, 255, 64)
computed = Color.from_hsl(215, 0.9, 0.55, 0.8)
perceptual = Color.from_oklch(0.68, 0.17, 240)
```

`colors.tailwind` contiene las 26 familias de Tailwind CSS v4.3.3, con escalas de 50 a 950: `colors.tailwind.blue[500]` o `colors.tailwind["rose"][600]`. `colors.tailwind.families()` las enumera y `colors.tailwind.version` da la versión incluida.

```
from gaanim import colors

brand = colors.tailwind.blue[600]
warning = colors.tailwind["amber"][500]
print(len(colors.tailwind.families()), colors.tailwind.version)

26 4.3.3
```

49.1.7.1. Constantes de color

Nombre	Valor	Color
`WHITE`	`#FFFFFF`	Blanco
`BLACK`	`#000000`	Negro
`GRAY`	`#808080`	Gris
`BLUE`	`#193264`	Azul oscuro
`NAVY`	`#1B1F3B`	Azul marino
`TEAL`	`#2E86AB`	Azul verdoso
`CYAN`	`#4BE5E5`	Cian
`GREEN`	`#4BE57C`	Verde
`YELLOW`	`#F5D04B`	Amarillo
`GOLD`	`#FFD700`	Dorado
`ORANGE`	`#FF9F43`	Naranja
`CORAL`	`#FF6464`	Coral
`RED`	`#E54B4B`	Rojo
`PINK`	`#FF7AB6`	Rosa
`PURPLE`	`#9B59B6`	Morado

49.1.8. Pinceles y gradientes

`Drawable.fill(...)` y `Drawable.stroke(...)` aceptan un `ColorLike` o un `Brush` reutilizable. Las coordenadas de un gradiente están en el espacio local del objeto, así que la pintura acompaña sus transformaciones. Los colores se reparten uniformemente y hacen falta al menos dos. `extend` decide qué pasa fuera de la rampa: `"pad"` (predeterminado) prolonga los extremos, `"repeat"` la repite y `"reflect"` la repite en espejo.

Un grupo pasa el pincel a cada miembro, que lo pinta en sus propias unidades locales: un gradiente no cruza el grupo, sino que se repite en cada pieza. Para que un solo gradiente recorra varias formas, únelas con `scene.geometry.union(...)` o da a cada una un pincel desplazado según su posición.

49.1.9. Brush.solid

FÁBRICA `solid(color: ColorLike) -> Brush`

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`color`	`ColorLike`	obligatorio	Color del pincel.

Devuelve: `Brush` — Un pincel de color uniforme.

49.1.10. Brush.linear

FÁBRICA `linear(colors: Sequence[ColorLike], *, start: tuple[float, float], end: tuple[float, float], extend: Literal["pad", "repeat", "reflect"] = "pad") -> Brush`

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`colors`	`Sequence[ColorLike]`	obligatorio	Colores de la rampa, en orden.
`start, end`	`tuple[float, float]`	obligatorio	Extremos del gradiente, en unidades locales.
`extend`	`"pad" | "repeat" | "reflect"`	`"pad"`	Comportamiento fuera de la rampa.

Devuelve: `Brush` — Un gradiente lineal.

Ejemplo:

```
gradient = Brush.linear(["#7AA2F7", "#BB9AF7", "#F7768E"], start=(-3, 0), end=(3, 0))
card = scene.geometry.rounded_rect(6, 2.75, 0.35).fill(gradient)
```

49.1.11. Brush.radial

FÁBRICA `radial(colors: Sequence[ColorLike], *, center: tuple[float, float] = (0.0, 0.0), radius: float, extend: Literal["pad", "repeat", "reflect"] = "pad") -> Brush`

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`colors`	`Sequence[ColorLike]`	obligatorio	Colores del centro hacia fuera.
`center`	`tuple[float, float]`	`(0.0, 0.0)`	Centro, en unidades locales.
`radius`	`float`	obligatorio	Radio del último color.
`extend`	`"pad" | "repeat" | "reflect"`	`"pad"`	Comportamiento fuera de la rampa.

Devuelve: `Brush` — Un gradiente radial.

Ejemplo:

```
orb = scene.geometry.circle(1.5).fill(
    Brush.radial(["white", scene.canvas.color("accent"), "#0000"], center=(-0.375, 0.44),
radius=1.875)
)
```

49.1.12. Brush.sweep

FÁBRICA `sweep(colors: Sequence[ColorLike], *, center: tuple[float, float] = (0.0, 0.0), start_angle: float = 0.0, end_angle: float = 360.0, extend: Literal["pad", "repeat", "reflect"] = "pad") -> Brush`

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`colors`	`Sequence[ColorLike]`	obligatorio	Colores alrededor del centro.
`center`	`tuple[float, float]`	`(0.0, 0.0)`	Centro, en unidades locales.
`start_angle, end_angle`	`float`	`0.0, 360.0`	Ángulos del barrido, en grados.
`extend`	`"pad" | "repeat" | "reflect"`	`"pad"`	Comportamiento fuera de la rampa.

Devuelve: `Brush` — Un gradiente angular.

Ejemplo:

```
ring = scene.geometry.circle(1.375).no_fill().stroke(
    Brush.sweep(["#7DCFFF", "#9ECE6A", "#E0AF68", "#F7768E", "#7DCFFF"], center=(0, 0)),
    0.25,
)
```

`animate.fill(paint)` y `animate.stroke(paint, width)` interpolan entre pinturas: dos gradientes del mismo tipo interpolan su geometría y sus colores (con números de paradas distintos, se normalizan sus posiciones), y un color sólido puede convertirse en gradiente y al revés. Cambiar entre tipos incompatibles, como lineal y radial, o entre modos de `extend` distintos, lanza un error al programar la animación. Consulta `Anim.fill`.

`StrokeStyle` agrupa todo un trazo reutilizable; se aplica con `Drawable.stroke_style` o dentro de un tema.

49.1.13. StrokeStyle

CLASE `StrokeStyle(paint: Paint, width: float = 0.02, *, cap: str = "round", join: str = "round", miter_limit: float = 4.0, dashes: Sequence[float] = (), dash_offset: float = 0.0, align: str | None = None)`

Métricas inválidas lanzan `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`paint`	`Paint`	obligatorio	Color, `Brush` o nombre de un token del tema.
`width`	`float`	`0.02`	Ancho en unidades lógicas.
`cap`	`"butt" | "round" | "square"`	`"round"`	Remate de los extremos.
`join`	`"bevel" | "miter" | "round"`	`"round"`	Unión entre segmentos.
`miter_limit`	`float`	`4.0`	Límite de inglete.
`dashes`	`Sequence[float]`	`()`	Longitudes alternas de trazo y hueco; vacío es continuo. Un trazo de longitud cero con remate `"round"` o `"square"` dibuja un punto o un cuadrado.
`dash_offset`	`float`	`0.0`	Desplazamiento del patrón.
`align`	`"inside" | "center" | "outside" | None`	`None`	Posición del trazo en los

			contornos cerrados, como en `Drawable.stroke`; `None` conserva la del objeto.

Devuelve: `StrokeStyle`

Ejemplo:

```
dashed = StrokeStyle(GOLD, 0.04, cap="butt", dashes=(0.2, 0.1))
guide = scene.geometry.line(-4, 0, 4, 0).stroke_style(dashed)
```

49.1.14. Fondos

`Scene(background=...)`, `scene.canvas.background` y `scene.segment(..., background=...)` aceptan un color, un `Brush` o un `Background`. Sus coordenadas son las de la escena: con el marco de 16 × 9, un gradiente lineal de ancho completo va de `x=-8` a `x=8`. El fondo explícito tiene prioridad sobre el del tema.

49.1.15. Canvas.background

PROPIEDAD `background: BackgroundLike | None`

Sin fondo explícito devuelve el del tema, o `None` si no hay tema.

Devuelve: `BackgroundLike | None` — El fondo de la escena; se puede leer y asignar.

Ejemplo:

```
scene.canvas.background = Color(40, 42, 54)
scene.canvas.background = Brush.linear(["#071022", "#164E8A", "#7DD3FC"], start=(-8, 0), end=(8,
0))
```

49.1.16. Background

CLASE `Background(paint: Paint)`

El fondo se ajusta al cuadro de la escena (16 × 9 por defecto) y se mueve con la cámara como el resto del mundo. Si la cámara se desplaza o se aleja más allá del cuadro, el fondo sigue llenando la vista: sus bordes se prolongan, así que un degradado o un shader continúan con el color de su borde en lugar de dejar una franja vacía.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`paint`	`Paint`	obligatorio	Color o `Brush` que cubre toda la escena.

Devuelve: `Background`

49.1.17. Background.shader

FÁBRICA `shader(source: str | os.PathLike[str], *, fallback: Optional[ColorLike] = None, uniforms: Optional[Mapping[str, float | Parameter | Variable | Computed | TimeInput]] = None, audio: Optional[Audio] = None) -> Background`

WGSL inválido, un nombre de uniform inválido o reservado lanzan `ValueError` y un asset ilegible, `RuntimeError`. Como el `Parameter` o la pista se crean con la escena, un fondo con uniforms se asigna después: `scene.canvas.background = Background.shader(...)`. Los paquetes `.gaanim` guardan los valores de cada fotograma.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`source`	`str | os.PathLike[str]`	obligatorio	WGSL en línea, o la ruta de un asset `.wgsl` (con `pathlib.Path`), que se lee al crear el fondo. Una ruta relativa se busca en el directorio de trabajo, luego en la carpeta

			assets del script y luego junto a él.
fallback	ColorLike | None	None	Color fuera del marco, para calcular el contraste automático y si falla la rasterización; negro si se omite.
uniforms	Mapping[str, float | Parameter | Computed] | None	None	Campos f32 de gaanim_uniforms, como en PostProcess.shader: números fijos o un Parameter, Variable, Computed, scene.time o una señal de audio,

			evaluados en cada fotograma, así que el fondo reacciona a la música. Como máximo 32, con nombres que sean identificadores WGSL.
`audio`	`Audio | None`	`None`	Da al fondo el espectrograma de la pista con `gaanim_audio_spectr ago)` y las demás funciones `gaanim_audio_*` (ver Audio); reserva el uniform `gaanim_audio_time`.

Devuelve: `Background` — Un fondo procedural que sigue la línea de tiempo.

Ejemplo:

```python
from gaanim import Background, Scene

shader = Background.shader("""
fn gaanim_background(
    uv: vec2<f32>,
    resolution: vec2<f32>,
    time: f32,
) -> vec4<f32> {
    let center = vec2<f32>(0.35 + 0.1 * sin(time), 0.45);
    let glow = exp(-8.0 * distance(uv, center));
    return vec4<f32>(0.02, 0.08 + 0.4 * glow, 0.18 + 0.6 * glow, 1.0);
}
""", fallback="#071022")
scene = Scene(frame=(16, 9), background=shader)

from pathlib import Path
asset_shader = Background.shader(Path("assets/background.wgsl"), fallback="#071022")
```

`source` debe definir esta función:

```
fn gaanim_background(uv: vec2<f32>, resolution: vec2<f32>, time: f32) -> vec4<f32>
```

- `uv=(0, 0)` es la esquina superior izquierda y `uv=(1, 1)` la inferior derecha: `uv.y` crece hacia abajo, aunque la escena use y hacia arriba. Usa `1.0 - uv.y` para trabajar hacia arriba.
- `resolution` es el tamaño efectivo en píxeles.
- `time` es la posición absoluta de la línea de tiempo en segundos, así que las búsquedas, las capturas y la exportación son deterministas.
- `gaanim_frame_size(resolution)` devuelve el tamaño del marco en unidades de la escena (por ejemplo `(16, 9)`), para dibujar en las mismas unidades que las formas.

El shader cubre el mismo rectángulo de escena que muestra el editor. En el visor se ejecuta en la misma GPU que dibuja la escena y solo se repite cuando cambian el tiempo o el tamaño en píxeles. Las exportaciones y las capturas producen los mismos píxeles.

49.1.18. Background.fallback

PROPIEDAD `fallback: Color`

Devuelve: `Color` — El color representativo del fondo, usado para limpiar y para el contraste.

49.1.18.1. Fondos vivos

Fondos procedurales listos para usar, construidos sobre `Background.shader`: se mueven con el tiempo de la línea de tiempo y la misma semilla da siempre la misma imagen. Los colores se mezclan en luz lineal. Un valor inválido lanza `ValueError`.

49.1.19. Background.mesh_gradient

FÁBRICA `mesh_gradient(colors, *, speed=0.1, seed=0) -> Background`

Manchas suaves de 2 a 8 `colors` que derivan por órbitas y se mezclan como un *mesh gradient*. `speed` escala el movimiento (`0` lo congela) y `seed` elige otra disposición.

Ejemplo:

```python
from gaanim import WHITE, Background, Scene
scene = Scene(frame=(16, 9))
```

```
scene.canvas.background = Background.mesh_gradient(
    ["#0b1040", "#6a2cc8", "#12b8a6", "#f25f8b"], speed=0.3, seed=4
)
scene.text("Mesh gradient", role="title").fill(WHITE)
scene.wait(2.0)
scene.render()
```

49.1.20. Background.noise_gradient

FÁBRICA `noise_gradient(colors, *, scale=1.5, speed=0.05, seed=0) -> Background`

Ruido fBm con dominio deformado, al estilo de los gradientes de Stripe, recorriendo una rampa de 2 a 8 `colors`. `scale` es el número de rasgos por altura del cuadro.

49.1.21. Background.aurora

FÁBRICA `aurora(colors, *, sky=None, speed=0.2, seed=0) -> Background`

Cortinas de aurora boreal, una por color (1 a 8), sobre un cielo `sky` oscuro (azul noche si se omite).

49.1.22. Background.dot_grid

FÁBRICA `dot_grid(spacing=0.4, *, radius=0.03, color=None, background=None, drift=(0.0, 0.0)) -> Background`

Puntos cada `spacing` unidades de la escena, con uno en el origen. `radius` está en unidades y es como mucho la mitad de `spacing`; `color` (blanco translúcido por defecto, respeta su alfa) sobre `background` oscuro. `drift` desplaza la rejilla en unidades por segundo.

Ejemplo:

```
scene.canvas.background = Background.dot_grid(0.5, radius=0.05, color="#8fa3ff80", drift=(0.25, 0.0))
```

49.1.23. Postprocesado

Un `PostProcess` aplica una función WGSL sobre todo lo que la escena dibuja en 2D: fondo, texto, formas, imágenes y Lottie. Se ejecuta en el editor, en el modo presentación, en las capturas y en la exportación, siempre con el tiempo exacto de la línea de tiempo.

49.1.24. PostProcess.shader

FÁBRICA `shader(source: str | os.PathLike[str], *, uniforms: Optional[Mapping[str, float | Parameter | Variable | Computed | TimeInput]] = None, audio: Optional[Audio] = None) -> PostProcess`

WGSL inválido, un campo sin declarar, un nombre inválido o un número no finito lanzan `ValueError`, y un asset ilegible, `RuntimeError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`source`	`str | os.PathLike[str]`	obligatorio	WGSL en línea, o la ruta de un asset `.wgsl` que se lee al crear el objeto. Una ruta relativa se busca en el directorio de trabajo, luego en la carpeta `assets` del

			script y luego junto a él.
`uniforms`	`Mapping[str, float | Parameter | Computed] | None`	`None`	Valores que el shader lee como campos `f32` de `gaanim_uniforms` (`gaanim_uniforms.am` Un `Parameter`, `Variable`, `Computed` o `scene.time` se evalúa en cada fotograma, así que animarlo anima el efecto. Como máximo 32 por pasada, con nombres que sean identificadores WGSL.

`audio`	`Audio | None`	`None`	Da al shader el espectrograma de la pista: `gaanim_audio_spect ago)`, `gaanim_audio_level` `gaanim_audio_band( high, ago)` y `gaanim_audio_x(hz)`. Reserva el uniform `gaanim_audio_time`. Ver Audio.

Devuelve: `PostProcess` — Un postprocesado para `Scene(post=...)`, `scene.canvas.post` o `scene.segment(..., post=...)`.

Ejemplo:

```
from gaanim import PostProcess, Scene

vignette = PostProcess.shader("""
fn gaanim_post(uv: vec2<f32>, resolution: vec2<f32>, time: f32) -> vec4<f32> {
    let color = gaanim_scene(uv);
    let edge = smoothstep(0.9, 0.35, distance(uv, vec2<f32>(0.5)));
    return vec4<f32>(color.rgb * edge, color.a);
}
""")
scene = Scene(frame=(16, 9), post=vignette)
```

`source` debe definir esta función:

```
fn gaanim_post(uv: vec2<f32>, resolution: vec2<f32>, time: f32) -> vec4<f32>
```

- `uv=(0, 0)` es la esquina superior izquierda del cuadro de la cámara y `uv=(1, 1)` la inferior derecha.
- `resolution` es el tamaño del cuadro en píxeles de salida.
- `time` es la posición absoluta de la línea de tiempo en segundos.
- `gaanim_scene(uv)` devuelve el color ya renderizado en ese punto: sRGB con alfa sin premultiplicar. Fuera del cuadro devuelve el píxel más cercano del borde.
- El resultado se limita a `[0, 1]`.

Solo se procesa el cuadro de la cámara. El shader recibe píxeles, así que un radio fijo como `1.0 / 1920.0` cambia con la resolución de salida: para un número de píxeles usa `offset / resolution` y, para que se vea igual a cualquier resolución, expresa el radio como fracción de `uv`. También se aplica con una cámara en perspectiva, pero no a la salida SVG.

49.1.25. PostProcess.source / uniforms

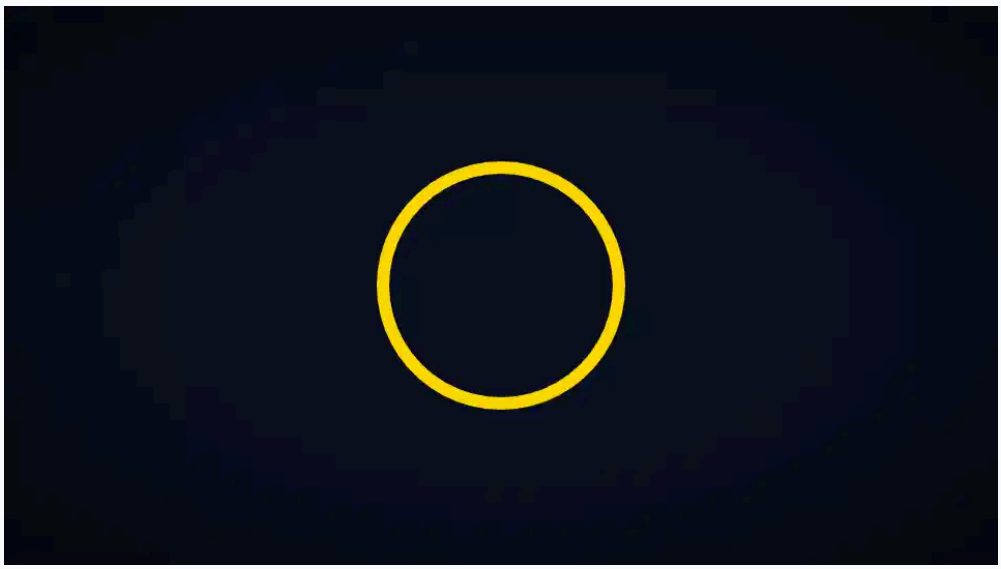
PROPIEDAD `source: str · uniforms: list[str]`

Devuelve: `str · list[str]` — El código WGSL de la función y los nombres de sus uniforms, en orden.

49.1.25.1. Uniforms animables y cadenas

Un uniform enlazado a un `Parameter` se anima como cualquier valor, y el efecto se ve igual en el editor y en la exportación. Varias pasadas se encadenan con una lista: cada una lee con `gaanim_scene` lo que escribió la anterior.

```
from gaanim import GOLD, PostProcess, Scene
scene = Scene(frame=(16, 9), background="#0b1020")
amount = scene.viz.parameter(0.0)
split = PostProcess.shader("""
fn gaanim_post(uv: vec2<f32>, resolution: vec2<f32>, time: f32) -> vec4<f32> {
    let shift = vec2<f32>(gaanim_uniforms.amount * 0.02, 0.0);
    let base = gaanim_scene(uv);
    return vec4<f32>(gaanim_scene(uv + shift).r, base.g, gaanim_scene(uv - shift).b, base.a);
}
""", uniforms={"amount": amount})
scene.canvas.post = [split, PostProcess.vignette(0.5)]
ring = scene.geometry.circle(2).no_fill().stroke(GOLD, 0.2)
scene.play([amount.animate.set(1.0).duration(0.5).repeat(2, yoyo=True)])
scene.render()
```



49.1.25.2. Presets de acabado

Pasadas listas para usar, sin escribir WGSL. Sus valores aceptan un número o un `Parameter` para animarlos. Los tamaños se miden en píxeles de un cuadro de 1080 de alto, así que se ven igual a cualquier resolución de salida, y el ruido cambia cada 1/24 s del tiempo de la línea de tiempo, así que un fotograma es siempre el mismo. Un número fuera de rango lanza `ValueError`.

49.1.26. PostProcess.grain

FÁBRICA `grain(amount=0.06, size=1.0, animated=True) -> PostProcess`

Grano de película: ruido de brillo de `amount` en celdas de `size` píxeles. Con `animated=False` el patrón queda fijo.

49.1.27. PostProcess.vignette

FÁBRICA `vignette(strength=0.35, softness=0.6) -> PostProcess`

Oscurece hacia las esquinas hasta `strength` (0 a 1). `softness` (0 a 1) marca desde dónde empieza la caída.

49.1.28. PostProcess.chromatic_aberration

FÁBRICA `chromatic_aberration(amount=0.004) -> PostProcess`

Separa el rojo hacia fuera y el azul hacia dentro, `amount` del cuadro en las esquinas (0 a 0.5).

49.1.29. PostProcess.color_grade

FÁBRICA `color_grade(exposure=0.0, contrast=1.0, saturation=1.0, temperature=0.0) -> PostProcess`

Corrección de color: `exposure` en pasos y `temperature` (-1 frío a 1 cálido) en luz lineal; después `contrast` alrededor del gris medio y `saturation` alrededor de la luminancia. `1.0` deja contraste y saturación igual.

49.1.30. PostProcess.lut

FÁBRICA `lut(path, strength=1.0) -> PostProcess`

Aplica la tabla 3D de un archivo `.cube` (`LUT_3D_SIZE` de 2 a 65, con `DOMAIN_MIN`/`DOMAIN_MAX` opcionales), mezclada según `strength`. El archivo se lee al crear el objeto (una ruta relativa se busca en el directorio de trabajo, luego en la carpeta `assets` del script y luego junto a él): si no se puede leer lanza `RuntimeError` y una tabla inválida, `ValueError`.

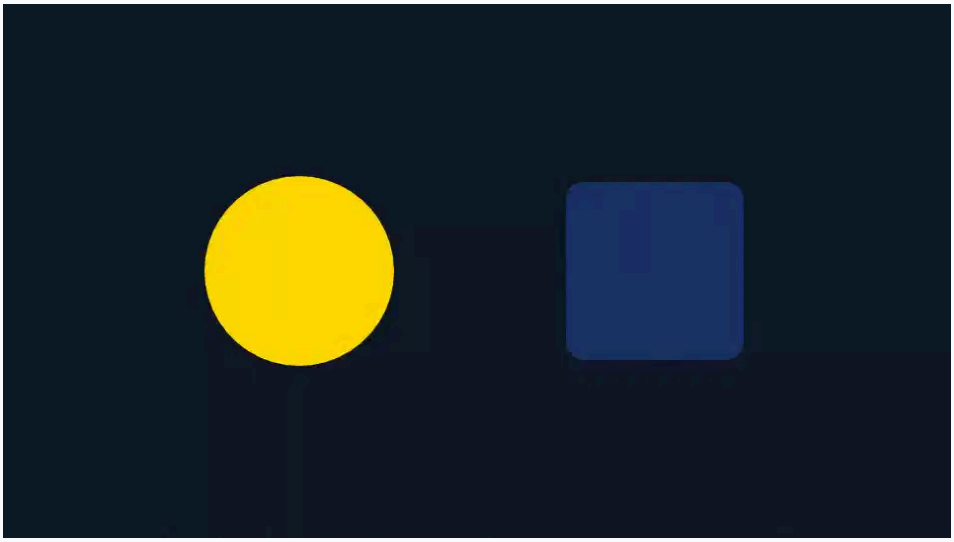
49.1.31. PostProcess.halftone / dither / crt / pixelate / glitch

FÁBRICA `halftone(dot=6.0) · dither(levels=4) · crt(strength=1.0) · pixelate(size=8.0) · glitch(intensity=0.5, seed=0)`

`halftone`: puntos de imprenta en celdas de `dot` píxeles, más grandes cuanto más clara es la celda. `dither`: tramado ordenado (Bayer 4×4) a `levels` valores por canal. `crt`: pantalla curva, líneas de barrido y máscara RGB. `pixelate`: bloques de `size` píxeles. `glitch`: franjas que saltan de lado con separación RGB 12 veces por segundo; `seed` cambia el patrón y `intensity` se puede animar para dar golpes.

Ejemplo:

```python
from gaanim import BLUE, GOLD, PostProcess, Scene
scene = Scene(frame=(16, 9), background="#101826")
burst = scene.viz.parameter(0.0)
scene.canvas.post = [PostProcess.glitch(intensity=burst, seed=3), PostProcess.grain(0.05)]
scene.geometry.circle(1.6).fill(GOLD).move_to(-3, 0)
scene.geometry.rounded_rect(3, 3, 0.3).fill(BLUE).move_to(3, 0)
scene.play([burst.animate.set(1.0).duration(0.3).repeat(2, yoyo=True)])
scene.render()
```



49.1.32. PostProcess.bloom

FÁBRICA `bloom(threshold=0.8, intensity=0.6, radius=0.5) -> PostProcess`

Bloom multipaso: lo que supera `threshold` (brillo sRGB de 0 a 1, con rodilla suave) brilla. Extrae las zonas claras, las reduce en una cadena de mips y las vuelve a sumar con un filtro tienda, como el bloom de Blender. `intensity` escala la luz añadida y `radius` (0 a 1) la extiende de un halo ceñido a una neblina amplia. La luz se suma en luz lineal y los núcleos muy brillantes se vuelven blancos en vez de saturarse. Su alcance es una fracción del cuadro, así que se ve igual a cualquier resolución. Complementa el `glow` vectorial de cada objeto.

Ejemplo:

```python
from gaanim import PostProcess, Scene
scene = Scene(frame=(16, 9), background="#05060d")
glow = scene.viz.parameter(0.3)
scene.canvas.post = PostProcess.bloom(threshold=0.55, intensity=glow, radius=0.6)
ring = scene.geometry.circle(1.6).no_fill().stroke("#ff4fd8", 0.12).move_to(-3, 0)
star = scene.geometry.regular_polygon(5, 1.6).no_fill().stroke("#ffe45c", 0.12).move_to(3, 0)
scene.play([ring.animate.create(), star.animate.create()])
scene.play([glow.animate.set(1.4).duration(0.6).repeat(2, yoyo=True)])
scene.render()
```

49.1.33. Canvas.motion_blur

MÉTODO `motion_blur(shutter_angle: Optional[float] = 180.0, samples: int = 8, *, phase: Optional[float] = None) -> None`

Desenfoque de movimiento como el de una cámara de cine en las exportaciones y las capturas: cada cuadro promedia en luz lineal `samples` búsquedas exactas repartidas por el obturador, así que es determinista, y nunca mezcla dos segmentos. La duración del obturador sigue los fps de la exportación (las capturas usan 60). El visor interactivo se ve nítido, y renderizar cuesta `samples` veces más. Un valor fuera de rango lanza `ValueError`. Usa `drawable.motion_blur(False)` para mantener nítido un objeto, y `Segment.motion_blur` para darle a un segmento su propio obturador.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`shutter_angle`	`float | None`	`180.0`	Grados de un cuadro que el obturador está abierto, de 0 a 720: 180 es el estándar del cine y 360 difumina todo el cuadro. `None` lo desactiva.
`samples`	`int`	`8`	Subcuadros promediados por cuadro, de 2 a 64.
`phase`	`float | None`	`None`	Dónde se abre el obturador respecto al tiempo del cuadro, en grados de un cuadro; por defecto se centra en él.

Devuelve: `None`

Ejemplo:

```python
from gaanim import CORAL, GOLD, WHITE, Scene
scene = Scene(frame=(16, 9), background="#101826")
scene.canvas.motion_blur(180, samples=6)
title = scene.text("Motion blur", role="title").fill(WHITE).move_to(0, 3).motion_blur(False)
ball = scene.geometry.circle(0.5).fill(CORAL).move_to(-6, 0)
bar = scene.geometry.rect(4, 0.35).fill(GOLD).move_to(0, -2)
scene.play([ball.animate.move_to(6, 0).duration(0.6),
bar.animate.rotate_by(6.2832).duration(1.0)])
scene.render()
```

49.1.34. Canvas.post

PROPIEDAD `post: PostProcess | list[PostProcess] | None`

Una lista encadena las pasadas en orden. `None` o una lista vacía lo quitan. Un segmento puede cambiarlo mientras está activo: `post=otro` (o una lista) usa otra cadena, `post=False` dibuja el segmento sin postprocesado y `post=None` (el predeterminado) hereda el de la escena. `drawable.post_process(False)` deja un objeto fuera de todas las pasadas. Un uniform que lee un valor de otra escena lanza `ValueError` y cualquier otro valor, `TypeError`.

Devuelve: `PostProcess | list[PostProcess] | None` — El postprocesado de la escena; se puede leer y asignar.

Ejemplo:

```python
from pathlib import Path

vignette = PostProcess.shader(Path("assets/grade.wgsl"))
scene.canvas.post = vignette
scene.segment("intro")                # hereda scene.canvas.post
scene.wait(0.5)
scene.segment("datos", post=False)    # sin postprocesado
scene.wait(0.5)
```

49.1.35. Temas

Un tema reúne colores semánticos, tipografía, reglas de estilo por tipo de objeto, paletas de datos y tokens de espaciado. Los componentes (tarjetas, insignias, bandas, gráficas, tablas, paneles de código)

toman de él sus colores de primer plano, acento, panel y regla. Un `.fill(...)` explícito siempre tiene prioridad.

49.1.36. Canvas.set_theme

MÉTODO `set_theme(theme: ThemeName | Theme | None) -> None`

Equivale a `Scene(theme=...)` después de crear la escena y sustituye al tema `technical` predeterminado. El fondo del tema reemplaza al actual salvo que la escena tenga un fondo explícito. Las reglas se aplican al compilar, así que también alcanzan a los objetos creados antes de la llamada. Un nombre desconocido lanza `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`theme`	`ThemeName | Theme | None`	obligatorio	Nombre o alias de un tema incluido, un `Theme`, o `None` para quitar el tema.

Devuelve: `None` — Instala el tema en la escena, o lo quita.

Ejemplo:

```
from gaanim import Scene

scene = Scene(frame=(16, 9))
scene.canvas.set_theme("presentation")
title = scene.text("Transformada de Fourier", role="title")
subtitle = scene.text("Representación en frecuencia", role="subtitle").move_to(0, -1)
scene.play([title.animate.write(), subtitle.animate.fade_in()])
scene.render()
```

Tema	Alias	Aspecto
`technical` (predeterminado)	`scientific`	Oscuro y sobrio, para matemáticas y documentación técnica: fondo `#121212`, texto `#E6E6E6`, datos en gris `#BDBDBD`, acento ámbar `#F2A541` y rejilla `#707070`.
`presentation`	`deck`	Para proyectar: el mismo fondo gris casi negro que `technical` (`#121212`), títulos y acentos dorados, gráficos en coral `#F4845F` y cuerpo brillante.
`paper`	`light`	Lienzo blanco con tinta oscura.

`dracula`, `nord`, `tokyo-night`	`tokyo`	Paletas oscuras de editores conocidos.
`solarized-dark`, `solarized-light`		Solarized en sus dos variantes.
`gruvbox-dark`	`gruvbox`	Gruvbox oscuro.
`catppuccin-mocha`, `catppuccin-latte`	`catppuccin`, `mocha`, `latte`	Catppuccin oscuro y claro.
`gaanim`		Los colores de la marca: papel `#F7F6FF` sobre grafito `#121212`, acento violeta `#7C6CFF`, títulos en oro `#FFC933`, gráficos en violeta claro `#A99FFF` y reglas en bruma `#3F37A8`.

Los nombres no distinguen mayúsculas. En el stub, `Scene(theme=...)`, `Canvas.set_theme` y `Theme(base)` tipan el nombre como `ThemeName`, un `Literal` con los doce nombres y los ocho alias de la tabla, así que el editor los autocompleta. `Theme.schemes()` devuelve solo los nombres canónicos.

49.1.37. Canvas.theme

PROPIEDAD `theme: str | None`

Es de solo lectura: para cambiarlo usa `set_theme`.

Devuelve: `str | None` — Nombre del tema activo: `"technical"` en una escena nueva, o `None` después de `Scene(theme=None)` o `set_theme(None)`.

49.1.38. Theme.schemes

MÉTODO `schemes() -> list[str]`

Devuelve: `list[str]` — Los nombres canónicos de los temas incluidos, sin alias.

Ejemplo:

```
print(Theme.schemes())

['technical', 'presentation', 'paper', 'dracula', 'nord', 'solarized-dark', 'solarized-light',
'gruvbox-dark', 'tokyo-night', 'catppuccin-mocha', 'catppuccin-latte', 'gaanim']
```

49.1.39. Canvas.set_fonts

MÉTODO `set_fonts(*, font: Optional[str] = None, math_font: Optional[str] = None, code_font: Optional[str] = None) -> None`

Cambia solo las familias, sin crear un `Theme`. Lo omitido conserva su valor anterior. Las familias tienen prioridad sobre el tema activo y siguen vigentes si el tema cambia después; un `font` explícito en un objeto sigue ganando. Una familia vacía lanza `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`font`	`str | None`	`None`	Familia para títulos, cuerpo, subtítulos, encabezados, pies y etiquetas.
`math_font`	`str | None`	`None`	Familia para ecuaciones y fragmentos matemáticos.
`code_font`	`str | None`	`None`	Familia para texto con rol `code`.

Devuelve: `None` — Configura el lienzo; no crea objetos.

Ejemplo:

```
from gaanim import Scene

scene = Scene(frame=(16, 9), theme="technical")
scene.canvas.set_fonts(
    font="Inter",
    math_font="New Computer Modern Math",
    code_font="JetBrains Mono",
)
scene.text("Texto y $x^2$", role="body")
scene.text("fn main() {}", role="code").move_to(0, -1)
```

49.1.40. Canvas.color

MÉTODO `color(role: str) -> Color`

Sirve para que tus propios objetos usen los colores del tema. Sin tema activo, o con un rol desconocido, lanza `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`role`	`str`	obligatorio	Rol o token de color del tema activo.

Devuelve: `Color` — El color resuelto.

Ejemplo:

```
panel = scene.geometry.rounded_rect(5.25, 2.25, 0.3) \
    .fill(scene.canvas.color("panel")) \
    .stroke(scene.canvas.color("accent"), 0.04)
```

49.1.41. Canvas.palette

MÉTODO `palette() -> Palette`

Sirve para colorear tus propias categorías como las colorean los gráficos.

Devuelve: `Palette` — La paleta categórica del tema activo; `gaanim` sin tema.

Ejemplo:

```
teams = ["Norte", "Sur", "Este"]
colors = scene.canvas.palette().colors(len(teams))
dots = [scene.geometry.dot(0.3).fill(color).move_to(index - 1, 0) for index, color in
enumerate(colors)]
```

49.1.42. Canvas.layout_token

MÉTODO `layout_token(name: str) -> float`

Resuelve un espaciado del tema activo. Sin tema usa la escala predeterminada: `space_xs`, `space_sm`, `space_md`, `space_lg`, `page_padding`, `page_padding_wide`, `page_padding_x`, `column_gap`, `vertical_padding`, `vertical_padding_x` y `lower_third_offset`. Un nombre desconocido lanza `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`name`	`str`	obligatorio	Nombre del token.

Devuelve: `float` — El valor en unidades lógicas.

Ejemplo:

```
page = scene.layout.column(
    [title, body],
    padding=scene.canvas.layout_token("page_padding"),
    gap=scene.canvas.layout_token("space_lg"),
)
```

49.1.43. Canvas.validate_theme

MÉTODO `validate_theme() -> list[str]`

Comprueba el contraste de primer plano, título, texto atenuado y panel sobre el fondo, y la tipografía. Es orientativa: no rechaza nada, porque un estado animado puede tener poco contraste a propósito. Sin tema activo lanza `ValueError`.

Devuelve: `list[str]` — Advertencias; vacía si las combinaciones principales son legibles.

Ejemplo:

```
warnings = scene.canvas.validate_theme()
if warnings:
    raise ValueError("\n".join(warnings))
```

49.1.43.1. Temas propios

49.1.44. Theme

CLASE `Theme(base: ThemeName | Theme | None = None, *, name=None, colors=None, fonts=None, sizes=None, text=None, styles=None, series=None, heatmap=None, palette=None, layout=None, font_files=None, font_dir=None, text_markup=None)`

Los archivos de fuente se leen al crear el tema y se incorporan a la escena, así que la exportación no depende de que estén instalados. Con `font_dir`, cada cara se resuelve por la familia, el peso y el estilo que declara su archivo: con `fonts={"text": "Aleo"}`, `weight=700` encuentra la negrita. Selectores, tokens, roles o métricas inválidos, una carpeta sin fuentes o una fuente ilegible lanzan `ValueError`; un archivo o carpeta inexistente, `OSError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`base`	`ThemeName | Theme | None`	`None`	Tema del que se deriva; solo se sustituye lo que pasas. Sin él, se parte de la paleta de `technical` con el nombre `custom`.
`name`	`str | None`	`None`	Nombre del tema nuevo.
`colors`	`dict[str, ColorLike] | None`	`None`	Roles de color y tokens propios para las reglas.
`fonts`	`dict[str, str] | None`	`None`	Familias por rol de fuente. `"text"` nombra todos los roles de texto y `"all"` además `math` y `code`; un rol nombrado aparte, como `"title"`, conserva su familia.
`sizes`	`dict[str, float] | None`	`None`	Tamaños por rol de texto, en unidades lógicas.

`text`	`dict[TextRole, TextStyle] | None`	`None`	`TextStyle` por rol, como capa sobre el rol semántico.
`styles`	`dict[str, Style | AxesStyle] | None`	`None`	Reglas por selector (ver abajo).
`series`	`Sequence[ColorLike] | None`	`None`	Colores de serie: el primero colorea los gráficos sin canal `color`.
`heatmap`	`Sequence[ColorLike] | None`	`None`	Colores continuos de los mapas de calor. Un heatmap cuyo campo `color` lleva `Scale.colors` usa los de su escala.
`palette`	`PaletteLike | None`	`None`	Paleta categórica de la escena: colorea las categorías de todos los gráficos, leyendas, tartas y carreras de barras. Una `Palette`, un nombre (`"okabe_ito"`) o una lista de colores; por defecto, `gaanim`. `Scale.category().colors(...)` la sustituye en un campo concreto.

`layout`	`dict[str, float] | None`	`None`	Tokens de espaciado, nuevos o sustituidos.
`font_files`	`dict[str, str] | None`	`None`	Familia → archivo de fuente.
`font_dir`	`str | os.PathLike | None`	`None`	Carpeta cuyos `.ttf`, `.otf`, `.ttc` y `.otc` se registran (sin subcarpetas).
`text_markup`	`bool | None`	`None`	Con `False`, * y _ son literales por defecto en `scene.text`, `badge` y `chip`; `None` hereda el valor de la base.

Devuelve: `Theme` — Un tema reutilizable para `Scene(theme=...)` o `set_theme`.

Ejemplo:

```
from gaanim import AxesStyle, Scene, StrokeStyle, Style, TextStyle, Theme, colors

theme = Theme(
    "nord",
    name="mis-diapositivas",
    colors={"title": "#A3D9FF", "accent": "#FFB86C", "chart": "#88C0D0"},
    fonts={"text": "Aleo", "code": "Victor Mono"},
    sizes={"title": 0.72, "body": 0.34},
    text={
        "body": TextStyle(size=0.32, letter_spacing=0.1),
        "label": TextStyle(size=0.24, weight=600),
    },
    styles={
        "shape": Style(fill="accent"),
        "line": Style(stroke=StrokeStyle("foreground", 0.04, cap="round")),
        ".warning": Style(fill=colors.tailwind.rose[600]),
        "axes": AxesStyle(grid=StrokeStyle("rule", 0.01), labels=TextStyle(size=0.24)),
    },
    series=[colors.tailwind.blue[600], colors.tailwind.amber[500]],
    layout={"page_padding": 0.7, "column_gap": 0.6},
    font_files={
        "Aleo": "assets/fonts/Aleo-VariableFont.ttf",
        "Victor Mono": "assets/fonts/VictorMono-VariableFont.ttf",
    },
)
scene = Scene(frame=(16, 9), theme=theme)
```

Los roles de color incluidos son `background`, `foreground`, `muted`, `title`, `accent`, `chart`, `panel`, `header`, `rule`, `success`, `warning`, `danger` y `shadow` (la sombra de `shadow=True` en las cajas: negro

suave sobre un fondo claro y casi opaco sobre uno oscuro, salvo que el tema elija otro); `colors` también acepta tokens propios no vacíos para las reglas. Los tokens `code`, `code_keyword`, `code_string`, `code_number`, `code_comment`, `code_function` y `code_type` eligen los colores con los que `scene.text.code` resalta el código; sin ellos, los temas de editores usan su esquema y los demás los derivan de la paleta. Los roles de fuente son `text`, `all`, `title`, `subtitle`, `heading`, `body`, `caption`, `label`, `math` y `code`.

```
brand = Theme(
    name="marca",
    colors={
        "background": "#10131A", "foreground": "#F8FAFC", "muted": "#94A3B8",
        "title": "#FDE68A", "accent": "#38BDF8", "chart": "#22C55E",
        "panel": "#18202E", "header": "#202B3D", "rule": "#475569",
    },
    fonts={"text": "Aptos", "code": "Consolas"},
)
print_theme = Theme(brand, name="marca-impresa", colors={"background": "white", "foreground":
"#172033"})
plain = Theme("paper", font_dir="assets/fonts", fonts={"text": "Aleo"}, text_markup=False)
scene = Scene(theme=plain)
scene.text("tb:dist_comp")          # literal, sin cursiva
scene.text("*Nota*", markup=True)   # negrita
```

49.1.45. Theme.name

PROPIEDAD `name: str`

Devuelve: `str` — Nombre del tema.

49.1.46. Theme.text_markup

PROPIEDAD `text_markup: bool`

Devuelve: `bool` — Modo de markup por defecto del texto creado con este tema.

49.1.47. Theme.color

MÉTODO `color(role: str) -> Color`

Como `Canvas.color`, pero sobre un `Theme` que no tiene por qué estar activo.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`role`	`str`	obligatorio	Rol o token de color.

Devuelve: `Color` — El color del tema.

Ejemplo:

```
theme = Theme("nord")
divider = scene.geometry.line(-5, 0, 5, 0).stroke(theme.color("rule"), 0.025)
```

49.1.48. Theme.layout_token

MÉTODO `layout_token(name: str) -> float`

Un token desconocido lanza `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`name`	`str`	obligatorio	Nombre del token.

Devuelve: `float` — El valor en unidades lógicas.

49.1.49. Theme.validate

MÉTODO `validate() -> list[str]`

Devuelve: `list[str]` — Advertencias de contraste y tipografía; vacía si todo es legible.

49.1.49.1. Reglas por selector

Las claves de `styles` seleccionan una familia (`shape`, `line`, `text`, `glyph` para las llaves, `axes`, `plot`), el nombre exacto de una fábrica (`circle`, `rounded_rect`, `arrow`), una parte semántica como `axes/grid` o `axes/labels`, o una clase propia como `.warning`, que se asigna con `Drawable.style_class`. La precedencia es: tema base, familia, tipo exacto o parte semántica, clases en orden, valores del constructor y, por último, los setters del objeto. Las reglas se aplican al compilar, así que cambiar de tema actualiza también los objetos ya creados. La pintura de los recursos importados sigue la de su archivo salvo que la cambies explícitamente.

49.1.50. Style

CLASE `Style(*, fill: Paint | None = None, stroke: StrokeStyle | None = None, opacity: float | None = None, text: TextStyle | None = None)`

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`fill`	`Paint | None`	`None`	Relleno; una cadena puede nombrar un token del tema.
`stroke`	`StrokeStyle | None`	`None`	Trazo completo.
`opacity`	`float | None`	`None`	Opacidad.
`text`	`TextStyle | None`	`None`	Tipografía.

Devuelve: `Style` — Una regla; lo que omites no cambia.

Ejemplo:

```
theme = Theme(
    "paper",
    colors={"brand": "#2563eb", "danger": "oklch(58% .24 25)"},
    styles={"shape": Style(fill="brand"), ".danger": Style(fill="danger")},
)
```

```
scene = Scene(theme=theme)
ordinary = scene.geometry.circle(0.75)
warning = scene.geometry.square(1.25).style_class("danger")
explicit = scene.geometry.circle(0.5).fill("gold")
```

49.1.51. AxesStyle

CLASE `AxesStyle(*, axis=None, grid=None, minor_grid=None, ticks=None, numbers=None, labels=None)`

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`axis, grid, minor_grid, ticks`	`StrokeStyle | None`	`None`	Trazos de los ejes, la cuadrícula, la cuadrícula secundaria y las marcas.
`numbers, labels`	`TextStyle | None`	`None`	Tipografía de los números y de las etiquetas.

Devuelve: `AxesStyle` — Una regla para el selector `axes`.

49.1.52. Mapas de color

`ColorMap` ofrece 39 mapas canónicos de Matplotlib y 39 de Scientific Colour Maps, incluidos en Gaanim: no hace falta Matplotlib ni ningún archivo externo. Los nombres no distinguen mayúsculas. Las posiciones se normalizan a `[0, 1]`; los mapas continuos interpolan en sRGB y los categóricos conservan escalones. Un nombre, categoría, posición, alfa o lista de colores inválidos lanzan `ValueError`. Donde un parámetro acepta `ColorMapLike`, también vale el nombre como cadena.

49.1.53. ColorMap

CLASE `ColorMap(name: str)`

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`name`	`str`	obligatorio	Nombre de un mapa incluido.

Devuelve: `ColorMap` — Un mapa inmutable.

Ejemplo:

```
viridis = ColorMap("viridis")
midpoint = viridis.sample(0.5)
```

```
swatches = viridis.colors(8)
reverse = viridis.reversed()
```

49.1.54. ColorMap.named

FÁBRICA `named(name: str) -> ColorMap`

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`name`	`str`	obligatorio	Nombre de un mapa incluido.

Devuelve: `ColorMap` — Igual que `ColorMap(name)`.

49.1.55. ColorMap.from_colors

FÁBRICA `from_colors(colors: Sequence[ColorLike], positions: Optional[Sequence[float]] = None) -> ColorMap`

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`colors`	`Sequence[ColorLike]`	obligatorio	Al menos dos colores.
`positions`	`Sequence[float] | None`	`None`	Posiciones estrictamente crecientes entre 0 y 1; por defecto, uniformes.

Devuelve: `ColorMap` — Un mapa continuo propio, sin nombre ni categoría.

Ejemplo:

```
custom = ColorMap.from_colors(["#0f172a", "#06b6d4", "#f8fafc"], positions=[0.0, 0.4, 1.0])
transparent = custom.with_alpha(0.65)
```

49.1.56. ColorMap.names

MÉTODO `names(category: Optional[Literal["matplotlib", "scientific"]] = None) -> list[str]`

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`category`	`"matplotlib" | "scientific" | None`	`None`	Filtra por catálogo.

Devuelve: `list[str]` — Nombres de los mapas incluidos.

Ejemplo:

```
print(ColorMap.names("scientific")[:5])

['acton', 'bam', 'bamako', 'bamO', 'batlow']
```

49.1.57. ColorMap.name

PROPIEDAD `name: Optional[str]`

Devuelve: `str | None` — Nombre del mapa; `None` en uno propio.

49.1.58. ColorMap.category

PROPIEDAD `category: Optional[str]`

Devuelve: `str | None` — `"matplotlib"`, `"scientific"` o `None`.

49.1.59. ColorMap.categorical

PROPIEDAD `categorical: bool`

Devuelve: `bool` — `True` si el mapa es de escalones, como `tab10`.

49.1.60. ColorMap.sample

MÉTODO `sample(position: float) -> Color`

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`position`	`float`	obligatorio	Posición en `[0, 1]`.

Devuelve: `Color` — El color en esa posición.

49.1.61. ColorMap.colors

MÉTODO `colors(count: int) -> list[Color]`

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`count`	`int`	obligatorio	Número de colores.

Devuelve: `list[Color]` — Colores repartidos uniformemente.

49.1.62. ColorMap.reversed

MÉTODO `reversed() -> ColorMap`

Devuelve: `ColorMap` — El mapa invertido.

49.1.63. ColorMap.with_alpha

MÉTODO `with_alpha(alpha: float) -> ColorMap`

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`alpha`	`float`	obligatorio	Opacidad en `[0, 1]`.

Devuelve: `ColorMap` — El mapa con esa opacidad.

49.1.64. Paletas categóricas

Una `Palette` es una lista ordenada de colores que se reparten entre las categorías por turno: las de un campo `color` de texto, las porciones de una tarta o las barras de una carrera. La del tema (`Theme(palette=...)`) colorea toda la escena, y `Scale.category().colors(...)` la sustituye en un campo concreto. Cuando hay más categorías que colores, la paleta vuelve a empezar y cada vuelta aclara u oscurece un poco sus colores, así que dos categorías nunca comparten color. Donde un parámetro acepta `PaletteLike`, también vale el nombre de una paleta o una lista de colores.

Incluidas: `gaanim` (la predeterminada, de diez colores), `tableau10`, `okabe_ito` (siete colores distinguibles con daltonismo; sin el negro, que desaparece sobre un fondo oscuro), `set2`, `dark2`, `paired` y los mapas categóricos `tab10`, `tab20`, `tab20b` y `tab20c`. Los nombres no distinguen mayúsculas.

49.1.65. Palette

CLASE `Palette(colors: PaletteLike)`

Un nombre desconocido o una lista vacía lanzan `ValueError`.

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`colors`	`Palette | str | Sequence[ColorLike]`	obligatorio	Una paleta, el nombre de una incluida o al menos un color.

Devuelve: `Palette` — Una paleta inmutable.

Ejemplo:

```
okabe = Palette("okabe_ito")
brand = Palette(["#7c6cff", GOLD, (40, 180, 160)])
scene = Scene(frame=(16, 9), theme=Theme("gaanim", palette=okabe.rotate(1)))
```

49.1.66. Palette.names

MÉTODO `names() -> list[str]`

Devuelve: `list[str]` — Nombres de las paletas incluidas.

49.1.67. Palette.from_colormap

FÁBRICA `from_colormap(colormap: ColorMapLike, count: int) -> Palette`

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`colormap`	`ColorMapLike`	obligatorio	Mapa del que se toman los colores.
`count`	`int`	obligatorio	Número de colores, al menos uno.

Devuelve: `Palette` — `count` colores repartidos de un extremo al otro del mapa.

Ejemplo:

```
ramp = Palette.from_colormap("viridis", 5)
```

49.1.68. Palette.name

PROPIEDAD `name: Optional[str]`

Devuelve: `str | None` — Nombre de una paleta incluida; `None` en una propia o derivada.

49.1.69. Palette.colors

MÉTODO `colors(count: Optional[int] = None) -> list[Color]`

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`count`	`int | None`	`None`	Número de categorías; sin él, los colores propios de la paleta.

Devuelve: `list[Color]` — Los colores de las primeras `count` categorías, con las vueltas incluidas.

49.1.70. Palette.color

MÉTODO `color(index: int) -> Color`

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`index`	`int`	obligatorio	Posición de la categoría, desde 0.

Devuelve: `Color` — Su color; `palette[index]` hace lo mismo y `len(palette)` cuenta los colores propios.

49.1.71. Palette.reversed

MÉTODO `reversed() -> Palette`

Devuelve: `Palette` — Los mismos colores en orden inverso.

49.1.72. Palette.rotate

MÉTODO `rotate(steps: int) -> Palette`

Parámetros:

Nombre	Tipo	Predeterminado	Descripción
`steps`	`int`	obligatorio	Posiciones que avanza; negativo, hacia atrás.

Devuelve: `Palette` — Los mismos colores empezando `steps` más tarde: `rotate(1)` pone el segundo primero.

49.1.73. Efectos visuales

`glow`, `blur` y `shadow` se aplican a cualquier objeto y funcionan sobre rellenos y trazos, incluidos los gradientes; `no_effects()` los quita. Están en Drawable y sus versiones animadas en Animaciones.

CAPÍTULO

50. Línea de comandos

El comando gaanim: previsualizar, crear proyectos, validar, exportar, presentar y comparar capturas

El ejecutable `gaanim` abre el editor, crea proyectos y exporta escenas. Todos los comandos que reciben `<SCRIPT_O_PROYECTO>` aceptan la ruta de un script Python o la carpeta de un proyecto: en una carpeta se ejecuta el `entry` de su `gaanim.toml`.

```
gaanim                                  # abre el Inicio
gaanim mi-video                         # previsualiza con hot reload
gaanim init video mi-video              # crea un proyecto
gaanim check mi-video                   # valida sin abrir ventana
gaanim inspect mi-charla --json         # la línea de tiempo y sus líneas
gaanim export mi-video --output exports/video.mp4
gaanim export mi-charla --output mi-charla.gaanim   # graba un paquete
gaanim --present --monitor 1 mi-charla  # presenta a pantalla completa
gaanim --present mi-charla.gaanim       # presenta un paquete, sin Python
gaanim --diff --example mi-video        # compara capturas con un baseline
gaanim update                           # instala la versión más reciente
gaanim --version
```

`gaanim --help` resume el uso y `gaanim <comando> --help` lista las opciones de `init`, `check`, `inspect`, `export` y `--diff`. `--help`, `--version` y `update` funcionan aunque no haya un Python compatible instalado, igual que reproducir, presentar y exportar un paquete `.gaanim` (ver Compartir sin Python); el resto de comandos necesita el entorno de Python que Gaanim detecta junto al proyecto (ver Proyectos).

50.1.1. Códigos de salida

Código	Significado
`0`	El comando terminó bien.
`1`	`check` encontró errores (o advertencias con `--strict`), la exportación falló o `update` no pudo actualizar.
`2`	Argumentos inválidos, proyecto o manifiesto inválido, entorno de Python no disponible, o un script que falla o no llama a `scene.render()`.

50.2. Previsualizar

```
gaanim [--present] [--monitor <ÍNDICE>] [--duration <TIEMPO>] [--sections <LISTA>]
[--from <NOMBRE>] [--listen <DIRECCIÓN>] <SCRIPT_O_PROYECTO>
gaanim [--present] [--monitor <ÍNDICE>] [--duration <TIEMPO>] [--sections <LISTA>]
[--from <NOMBRE>] [--listen <DIRECCIÓN>] <PAQUETE.gaanim>
```

Sin argumentos, `gaanim` abre el Inicio: crear un proyecto, abrir una carpeta o volver a uno de los diez proyectos recientes. Con un script o proyecto abre el editor, que vuelve a ejecutar el script al guardar (hot reload) y recarga la escena cuando cambia un asset del proyecto. Con un paquete `.gaanim`, `gaanim` reproduce los fotogramas grabados con las mismas opciones, sin cargar Python.

Opción	Efecto
`--present`	Abre la escena en modo presentación: pantalla completa sin bordes para la audiencia y Presenter View para quien presenta. Requiere `<SCRIPT_O_PROYECTO>`.
`--monitor <ÍNDICE>`	Monitor de la presentación, contado desde `0`. Sin él se usa el monitor principal. Solo es válido junto con `--present`.
`--duration <TIEMPO>`	Duración prevista de la charla: `20m`, `1h30m`, `90s`, `20:00` o un número de minutos. Presenter View la cuenta hacia atrás y muestra si vas adelantado o retrasado (ver El tiempo de la charla).
`--sections <LISTA>`	Reproduce solo los segmentos indicados, separados por comas, por ejemplo `resultados,cierre`.
`--from <NOMBRE>`	Empieza en ese segmento y sigue hasta el final. Combinado con `--sections`, recorta la lista a partir de ese nombre.
`--listen <stdio|HOST:PUERTO>`	Deja que un editor de código mueva la vista previa y sepa dónde está, con JSON por líneas (ver Controlar la vista previa desde un editor).
`--message-format <FORMATO>`	`json`: mensajes, recargas y errores del script con su línea en JSON por líneas (ver Mensajes para herramientas).

Cada nombre de `--sections` y `--from` puede ser el nombre completo de un segmento, la clave de un `Section` (selecciona todos sus pasos) o el nombre de un `SectionStep`. No distingue mayúsculas de minúsculas. Para un nombre con comas, pásalo solo (`--sections "Tiempo, lugar y orientación"`: si el valor completo nombra un segmento, se usa tal cual) o escribe sus comas como `\,` dentro de una lista (`--sections "Tiempo\, lugar y orientación,Cierre"`). Un nombre que no selecciona nada muestra un error con los nombres disponibles y se reproduce la escena completa. La escena se construye entera igualmente, así que objetos, cámara y tema llegan al primer segmento elegido en el mismo estado que en una reproducción completa; solo se limitan la reproducción y la navegación entre pausas.

```
gaanim mi-charla --sections portada,resultados
gaanim --present --monitor 1 mi-charla --from resultados
```

Los atajos de teclado del modo presentación y de Presenter View están en Presentaciones.

50.2.1. Comandos y atajos del editor

La barra de reproducción se esconde mientras no la usas. Al abrir un proyecto y tras cada recarga se queda unos segundos a la vista; después, una línea corta en el borde inferior indica dónde está, y basta con bajar el puntero para que vuelva.

Atajo	Qué abre
`Ctrl+Shift+P`	La paleta de comandos: todas las acciones del editor en una lista que se filtra al escribir (reproducir, exportar, presentar, pantalla completa, overlays, consola, guardar el fotograma, cada sección de Settings, volver al Inicio, abrir un proyecto reciente). Cada fila muestra su atajo. Las flechas eligen, `Enter` ejecuta y `Esc` cierra.

`Ctrl+P`	Ir a un segmento o a una pausa por su nombre. Las pausas aparecen bajo su segmento y se encuentran también por el nombre de este. Al lado se ve la diapositiva tal como queda en ese punto, la misma vista previa de Presenter View, y las notas que se dicen desde allí. `Enter` salta y deja la reproducción en pausa allí.
`?`	La hoja de atajos: todos los atajos del editor agrupados, sacados de la misma lista que la paleta.

`Cmd` sustituye a `Ctrl` en macOS. Mientras escribes en un campo, los atajos de una sola tecla no actúan.

50.2.2. Controlar la vista previa desde un editor

Con `--listen`, un editor de código mueve la vista previa y sabe dónde está: lleva la vista previa a la diapositiva del cursor sin reiniciarla y muestra en qué segmento y pausa está. Cada mensaje es un objeto JSON en una línea.

- `--listen stdio`: las órdenes llegan por la entrada estándar y los eventos salen por la salida estándar, para un editor que lanza `gaanim` como proceso hijo. Lo que el script escriba con `print` también sale por ahí: descarta las líneas que no sean un objeto JSON.
- `--listen 127.0.0.1:7801` (o `localhost:7801`, `[::1]:7801`): escucha en ese puerto de este ordenador; cada editor que se conecta envía órdenes y recibe los eventos, empezando por el último `ready` y el último `playhead`. Solo acepta direcciones locales.

Órdenes:

```
{"cmd": "seek", "time": 12.5}
{"cmd": "seek", "segment": "Resultados"}
{"cmd": "seek", "segment": "resultados", "stop": "grafico"}
{"cmd": "seek", "stop": "grafico"}
{"cmd": "follow", "file": "C:/charla/main.py", "line": 42}
{"cmd": "play"}
{"cmd": "pause"}
{"cmd": "loop", "segment": "Problema"}
{"cmd": "loop", "start": 1.2, "end": 3.4}
{"cmd": "unloop"}
```

`seek` se detiene en un instante, en un segmento o un `Section` (por su clave), o en una pausa por su nombre, dentro del segmento indicado si lo hay; los nombres no distinguen mayúsculas. Un segmento se muestra donde descansa por primera vez: en su primera pausa (en un `Section`, la primera de sus pasos), como su primera página del PDF, o en su final si no tiene pausas. Su inicio todavía puede mostrar la diapositiva anterior. `follow` se detiene en el instante que representa esa línea del script, con las mismas reglas que `gaanim inspect`: lo creado en esa línea o, si no, en la línea más cercana por encima en el mismo archivo. Un segmento cuenta como en `seek`, una pausa o un marcador por su instante, un voiceover por su inicio y un `play` por su final, cuando lo que anima ya llegó.

`loop` repite un segmento o un `Section` (desde el inicio de su primer segmento hasta el final del último, con los mismos nombres que `seek`) o el tramo de `start` a `end` en segundos, como la tecla `L` y el botón de bucle de la barra de reproducción, que lo muestran encendido. Empieza desde el principio del tramo y reproduce, sin detenerse en las pausas. A diferencia de la tecla, no alterna: repetir el bucle que ya está activo lo deja encendido. Sobrevive a las recargas: un segmento se vuelve a buscar por su nombre y un tramo conserva sus tiempos, hasta que el bucle se cambia o se apaga

desde la vista previa. `unloop` lo apaga y devuelve el tramo que había antes. Un segmento que no existe o un tramo inválido responde con `error`, y una presentación (`--present`) no repite tramos.

Eventos:

```
{"event": "reload"}
{"event": "ready", "duration": 42.0, "timeline": {"schema": 1, "segments":
[...], ...}}
{"event": "playhead", "time": 12.5, "segment": "Resultados", "stop": "grafico",
"playing": false, "loop": null}
{"event": "failed", "message": "NameError: name 'cirlce' is not defined", "file":
"C:/charla/main.py", "line": 17}
{"event": "error", "message": "seek: no segment or section named \"Cierre\""}
```

`reload` llega cuando el script se volvió a ejecutar y la escena se reconstruye, y `ready` cuando ya se ve. `ready` trae en `timeline` la línea de tiempo de la escena que se ve, el mismo objeto que imprime `gaanim inspect --json`, así que un editor sabe sus segmentos, pausas y líneas sin volver a ejecutar el script; al reproducir un paquete `.gaanim` es `null`. `playhead` llega al moverse: enseguida tras un salto, una pausa o al cambiar de segmento o de pausa, y como mucho diez veces por segundo mientras se reproduce. `stop` es `null` antes de la primera pausa del segmento y `""` en una pausa sin nombre. `loop` es `[inicio, final]` mientras un bucle está encendido, lo haya puesto un editor o la vista previa, y `null` si no. `failed` trae la línea del error del script cuando el traceback la da, y `error`, por qué no se pudo cumplir una orden.

```
gaanim mi-charla --listen stdio
gaanim --present mi-charla --listen 127.0.0.1:7801
```

50.3. `gaanim init`

```
gaanim init <video|slides> [DIRECTORIO] [--force]
```

Crea un proyecto ejecutable con un `main.py` mínimo: `video` genera una escena de dos segundos (un título y un círculo) y `slides`, dos diapositivas con notas y pausas para Presenter View. Sin `DIRECTORIO` se usa `gaanim-video` o `gaanim-slides`.

Archivo	Contenido
`main.py`	Escena inicial del tipo elegido.
`gaanim.toml`	Manifiesto del proyecto (ver `gaanim.toml`).
`pyproject.toml`	Proyecto uv sin paquete propio (`requires-python = ">=3.14"`) que declara `gaanim` como dependencia.
`.python-version`	`3.14`.
`assets/`, `exports/`	Carpetas de recursos y de salidas, con un `.gitkeep`.
`README.md`, `AGENTS.md`	Instrucciones de uso y guía para asistentes de código.
`.gitignore`	Ignora exportaciones, capturas y vídeos generados.

Después de escribir los archivos, `init` prepara `.venv` con uv e instala el paquete de autoría de Gaanim; si no lo consigue, avisa y el proyecto sigue siendo válido. Si algún archivo del scaffold ya existe, falla. `--force` reescribe solo esos archivos y no borra nada de `assets/` ni de otras carpetas.

50.4. `gaanim check`

```
gaanim check <SCRIPT_O_PROYECTO> [--strict] [--message-format json]
```

Ejecuta el script sin abrir ventana y revisa la escena resultante. Imprime la duración y el tamaño del marco, una línea por problema (`error` o `warning`) y termina con `pass` o `fail`:

```
▸ check     charla/main.py · presentation
            3 segments · 5 stops · 42.0 seconds · 16×9
! warning   segment `intro` contains unnamed stops; names improve Presenter View
✓ pass      with 1 warning
```

- En toda escena: que la línea de tiempo dure algo, que no queden textos de plantilla que empiecen por `[` y, si quitaste el tema (`theme=None` o `set_theme(None)`), que el fondo no sea blanco o casi blanco, porque el texto, las formas y los ejes sin color propio también son blancos y no se verían.
- Que ningún texto quede tapado por completo por una figura opaca dibujada encima en una pausa o al final de un segmento. El aviso dice el texto, la figura y dónde se tapa por primera vez; súbelo con `z_layer` o `z_index`. Un texto oculto a propósito (con opacidad 0) o detrás de una figura translúcida, de una imagen o de una figura marcada con `covers_on_purpose()` no cuenta.
- Si la escena usa segmentos (una presentación): que exista al menos un segmento, que ninguno dure cero segundos, que cada uno tenga notas y que las pausas tengan nombre; avisa si el marco no es 16:9 o si la escena dura menos de un segundo.

Con un paquete `.gaanim`, `check` no necesita Python: describe el paquete y vuelve a componer cada fotograma para compararlo con el resumen grabado (ver Compartir sin Python).

Los errores devuelven el código `1`. `--strict` también devuelve `1` cuando solo hay advertencias. Si el script lanza una excepción o no termina con `scene.render()`, `check` lo informa y devuelve `2`.

```
gaanim check mi-charla --strict
```

Con `--message-format json`, el informe sale en JSON por líneas, cada diagnóstico con su archivo y línea cuando los tiene (ver Mensajes para herramientas).

50.5. `gaanim inspect`

```
gaanim inspect <SCRIPT_O_PROYECTO> [--json]
```

Ejecuta el script sin abrir ventana y lista su línea de tiempo: cada segmento con su inicio y su final, sus pausas, los marcadores, los voiceovers y la toma en vivo, cada uno con el archivo y la línea del script que lo creó. Sirve para que un editor de código sepa qué diapositiva tiene el cursor sin leer el Python.

```
charla/main.py · 3 segments · 4 stops · 1 markers · 42.00 s
   0.00–12.00   Portada                main.py:8
   4.50           stop titulo          main.py:14
  12.00–30.00   resultados · 1 · 1 · Datos
  20.00           marker climax        main.py:31
```

La línea es la del script que hizo la llamada: si una función tuya crea el segmento, la de la llamada dentro de esa función; un segmento que crea un `Section` lleva la línea del `Section`. Lo que no sale de un script, como el segmento implícito, no tiene línea.

Con `--json` imprime un solo objeto en una línea, la última de la salida estándar: lo que el script escriba con `print` sale antes. `schema` es `1` y solo crecerá con campos nuevos. Los tiempos van en segundos, y `file` y `line` son `null` cuando no hay línea:


```
{
  "schema": 1,
  "script": "C:/charla/main.py",
  "duration": 42.0,
  "segments": [
    {"name": "Portada", "section": null, "start": 0.0, "end": 12.0,
     "file": "C:/charla/main.py", "line": 8,
     "stops": [{"name": "titulo", "time": 4.5, "file": "C:/charla/main.py", "line":
14}]}
  ],
  "sections": [{"key": "resultados", "start": 12.0, "end": 30.0}],
  "markers": [{"name": "climax", "time": 20.0, "file": "C:/charla/main.py", "line":
31}],
  "voiceovers": [{"key": "intro", "start": 0.0, "end": 6.2, "file": "C:/charla/
main.py", "line": 10}],
  "live_take": null,
  "plays": [{"start": 0.0, "end": 1.5, "file": "C:/charla/main.py", "line": 11}]
}
```


`section` es la clave del `Section` que generó el segmento, y `sections` da cada clave una vez, del inicio de su primer segmento al final del último. `plays` tiene cada `play` con su inicio y su final: con ellos un editor pasa de una línea al instante que muestra. Un paquete `.gaanim` no guarda líneas, así que `inspect` solo acepta scripts y proyectos.

50.6. `gaanim export`

```
gaanim export <SCRIPT_O_PROYECTO> --output <ARCHIVO> [OPCIONES]
gaanim export <PAQUETE.gaanim> --output <VÍDEO_O_PDF> [OPCIONES]
```

Renderiza la escena completa, o un tramo, a un archivo. El script no cambia: la resolución y la calidad se eligen aquí, no en `Scene`. Las pausas (`scene.stop()`) no detienen la exportación de un vídeo; un PDF tiene una página por pausa (ver Diapositivas en PDF).

50.6.1. Formatos

La extensión de `--output` elige el formato.

Extensión	Formato	Transparencia
`.mp4`	H.264. Necesita FFmpeg.	No
`.webm`	VP9. Necesita FFmpeg.	Sí
`.mov`	Apple ProRes con audio PCM, para editores de vídeo: 422 HQ, o 4444 con `--transparent`. Necesita FFmpeg.	Sí
`.webp`	WebP animado. Necesita FFmpeg.	Sí
`.gif`	GIF. Necesita FFmpeg.	No
`.png`	Secuencia de PNG, un archivo por fotograma. No necesita FFmpeg.	Sí

`.pdf`	Diapositivas: una página por cada pausa de la presentación y otra para el final, con las notas del orador si se piden con `--notes`. No necesita FFmpeg.	No
`.gaanim`	Paquete de reproducción: todos los fotogramas que dibuja la escena, con sus segmentos, pausas, marcadores y audio, en un archivo que se reproduce, presenta y exporta sin Python.	No

En una secuencia PNG, un patrón `%d` o `%0Nd` en el nombre se sustituye por el número de fotograma desde 0 (`frames/f_%04d.png` escribe `f_0000.png`, `f_0001.png`…). Sin patrón, el número se añade al nombre: `frame.png` escribe `frame_00000.png`. El audio de la escena se mezcla en MP4, WebM y MOV.

50.6.2. Opciones

Opción	Predeterminado	Efecto
`-o, --output <ARCHIVO>`	obligatorio	Archivo de salida; su extensión elige el formato.
`--quality <PRESET>`	`standard`	`draft`: 30 fps y codificación rápida. `standard`: 60 fps y equilibrio. `production`: 60 fps y la mejor compresión, más lenta. GIF y WebP se quedan en 30 fps.
`--fps <TASA>`	la de `--quality`	Fotogramas por segundo, de 1 a 240: un entero (`24`), una fracción (`24000/1001`) o un decimal (`23.976`). Sustituye a la tasa de la calidad.
`--width <PX>`	`1920`	Ancho de salida en píxeles.
`--height <PX>`	`1080`	Alto de salida en píxeles.
`--fit <MODO>`	`error`	Qué hacer si la proporción de salida no coincide con el marco de la escena: `error` falla, `contain` añade bandas y `cover` recorta los bordes.
`--encoder <CODIFICADOR>`	`auto`	Solo MP4: `auto`, `libx264`, `nvenc`, `amf`, `qsv` o `vaapi`.
`--transparent`	desactivado	Conserva el canal alfa en WebM, MOV, WebP y PNG. MP4 y GIF lo rechazan.
`--from <SEGUNDOS|NOMBRE>`	`0`	Inicio del tramo exportado: segundos, un marcador, un segmento (su inicio) o una pausa.
`--to <SEGUNDOS|NOMBRE>`	fin de la escena	Fin del tramo exportado; un segmento lo termina en su final.
`--frame <SEGUNDOS|NOMBRE>[,…]`	—	Escribe exactamente un PNG en cada instante, en lugar de un tramo; un

		segmento da su primera pausa. Necesita una salida `.png` y no se combina con `--from` ni `--to`.
`--notes`	desactivado	Solo PDF: añade tras la última página de cada diapositiva una página con sus notas (`scene.segment(notes=...)`).
`--message-format <FORMATO>`	`human`	`json`: mensajes y progreso con su porcentaje en JSON por líneas, sin barra (ver Mensajes para herramientas).

La composición se define en unidades lógicas, así que `--width` y `--height` cambian la nitidez, no la disposición. Con una proporción distinta de la del marco (por ejemplo 1080×1920 para una escena 16×9) hace falta `--fit contain` o `--fit cover`.

Con `--encoder auto`, MP4 prueba NVENC, AMF y QSV con una codificación real y usa el primero que funciona; si ninguno funciona, usa `libx264`. Un codificador explícito no tiene alternativa: si falla, la exportación falla. VAAPI nunca se prueba automáticamente, porque un driver defectuoso puede bloquear la GPU.

`--transparent` solo tiene efecto si la escena tiene un fondo con alfa, por ejemplo `Scene(background="#00000000")`.

50.6.3. Fotogramas por segundo

Sin `--fps`, un vídeo sale a 30 fps con `--quality draft` y a 60 con `standard` o `production`; GIF y WebP no pasan de 30. `--fps` elige otra tasa para cualquier formato: 24 para cine y animación, 25 o 50 para PAL, y las fracciones NTSC 24000/1001, 30000/1001 y 60000/1001. Un decimal que redondea una de ellas (`23.976`, `29.97`, `59.94`) significa esa fracción exacta, que es la que lleva el archivo. La duración y el audio no cambian: solo cuántos fotogramas la cubren. El desenfoque de movimiento reparte sus muestras en el intervalo de cada fotograma de esa tasa.

```
gaanim export mi-video --output exports/cine.mp4 --fps 24
gaanim export mi-video --output exports/ntsc.mp4 --fps 24000/1001
```

50.6.4. Editores de vídeo

Un `.mov` es Apple ProRes, el formato de intercambio que Premiere, DaVinci Resolve y Final Cut importan sin conversión. Sin `--transparent` es ProRes 422 HQ; con `--transparent`, ProRes 4444 con canal alfa, para poner rótulos, *lower thirds* y gráficos de Gaanim encima de un montaje. El color se guarda etiquetado como BT.709 y el audio, sin comprimir (PCM de 24 bits). ProRes no tiene niveles de compresión: `--quality` solo elige los fps, y los archivos pesan mucho más que un MP4.

```
gaanim export rotulo.py --output rotulo.mov --transparent
gaanim export mi-video --output master.mov --fps 25
```

50.6.5. Tramos y marcadores

`--from` y `--to` aceptan segundos o un nombre, y se pueden combinar. Los nombres se resuelven después de ejecutar el script, en este orden:

1. un `scene.marker`, con su nombre exacto: su instante;
2. un segmento, la clave de un `Section` (todos sus pasos) o el nombre de un `SectionStep`, sin distinguir mayúsculas, como en `--from` al previsualizar: `--from` empieza en su inicio y `--to` termina en su final;

3. el nombre de una pausa (`scene.stop("nombre")`), sin distinguir mayúsculas: el instante en que descansa.

Un número son siempre segundos. Un nombre que no existe produce un error con los marcadores, segmentos y pausas definidos, y el de una pausa que se repite en varios segmentos es ambiguo: el error dice dónde está cada una, para pasar sus segundos. El audio se recorta al mismo tramo, una secuencia PNG numera sus fotogramas desde 0 y un `--to` posterior al final se limita a la duración de la escena. `--to` debe quedar después de `--from`.

```
gaanim export mi-video --output exports/video.mp4 --quality production
gaanim export mi-video --output exports/vertical.mp4 --width 1080 --height 1920 --fit
cover
gaanim export overlay.py --output overlay.webm --transparent
gaanim export rotulo.py --output rotulo.mov --transparent
gaanim export mi-video --output exports/tramo.mp4 --from 12 --to 15
gaanim export mi-video --output exports/climax.mp4 --from climax --to fin
gaanim export mi-charla --output exports/resultados.mp4 --from resultados --to
resultados
gaanim export mi-video --output frames/f_%04d.png --quality draft
```

50.6.6. Fotogramas sueltos

Para revisar la composición en un instante, `--frame` escribe exactamente un PNG con lo que se ve en ese momento, sin exportar un tramo. Acepta segundos y nombres separados por comas, con las mismas reglas que `--from` una coma dentro de un nombre se escribe `\,`. Un segmento o `Section` da el instante de su primera pausa, el mismo de su primera página en el PDF, o su final si no tiene pausas; una pausa, el instante en que descansa, el mismo que captura `--diff --capture-stops`. Con un solo instante escribe el archivo de `--output`; con varios añade su posición al nombre (`fotos/f.png` escribe `f_1.png`, `f_2.png`…). Respeta `--width`, `--height`, `--fit` y `--transparent`.

```
gaanim export mi-video --output revision.png --frame 11.4
gaanim export mi-video --output fotos/f.png --frame 3,11.4,climax
gaanim export mi-charla --output problema.png --frame Problema
```

50.6.7. Diapositivas en PDF

Con `--output` terminado en `.pdf`, `export` escribe una página por cada punto en el que se detiene una presentación: cada `scene.stop()`, en orden, y el final de la escena si la última pausa no está ahí. Cada página es el fotograma que muestra el Presenter en esa pausa, al tamaño de `--width` y `--height` (con `--fit` si la proporción no coincide), sin pérdida. Las páginas miden 960 puntos de ancho, como una diapositiva panorámica, con la proporción de la escena.

Con `--notes`, tras la última página de cada diapositiva (cada `scene.segment(...)`) va una página con su nombre y sus notas, en texto que se puede buscar y copiar. Una diapositiva sin notas no añade página. `--from`, `--to`, `--frame`, `--fps`, `--transparent` y `--encoder` no se aplican.

```
gaanim export mi-charla --output mi-charla.pdf
gaanim export mi-charla --output material.pdf --notes
gaanim export mi-charla.gaanim --output mi-charla.pdf   # sin Python
```

Un paquete se exporta a PDF igual, sin ejecutar Python. El diálogo **Export** del editor también ofrece el formato PDF, con una casilla para las notas.

50.6.8. Paquetes

Con `--output` terminado en `.gaanim`, `export` graba un paquete de la escena completa; `--from`, `--to`, `--transparent`, `--notes` y `--encoder` no se aplican, y `--fps` (60 por defecto) debe ser un entero. Con un paquete como entrada, `export` lo renderiza a vídeo o a PDF sin ejecutar Python: acepta los demás formatos y opciones, incluidos los tramos con marcadores, y el vídeo tiene la frecuencia con

la que se grabó el paquete; un `--fps` distinto es un error. El resultado es idéntico, píxel a píxel, a exportar el script con esa frecuencia y el mismo tamaño.

```
gaanim export mi-charla --output mi-charla.gaanim --fps 30
gaanim export mi-charla.gaanim --output mi-charla.mp4 --quality production
```

`gaanim pack` vuelve a escribir un paquete en el formato actual, sin Python ni GPU. Un paquete grabado con una versión anterior pasa así al formato que los reproductores abren con su primer bloque. Con `--web`, además, guarda los trazados en una rejilla y deja fuera lo que solo usan `gaanim check` y el desenfoque de movimiento al exportar vídeo. Queda más ligero para insertarlo en otras páginas:

```
gaanim pack mi-charla.gaanim mi-charla.gaanim       # al formato actual
gaanim pack mi-charla.gaanim web/charla.gaanim --web
```

La guía Compartir sin Python explica qué guarda un paquete, cómo se reproducen los callbacks y sus límites.

50.7. `gaanim relay`

Configura el relay que lleva los votos de las encuestas (`scene.poll`) a la presentación. No necesita Python. La guía Tu primera encuesta explica el despliegue.

```
gaanim relay                          # muestra el relay en uso y de dónde sale
gaanim relay init [CARPETA] [--force] # escribe el relay (por defecto ./gaanim-relay)
gaanim relay use https://…workers.dev # lo guarda para todas tus presentaciones
(también en Settings)
gaanim relay forget                   # olvida el relay guardado
gaanim relay reset [RUTA]             # partida nueva en la sesión del proyecto de
RUTA
gaanim relay results [RUTA] [--output CARPETA]  # guarda los resultados de la partida
gaanim relay key <ARCHIVO.gaanim|CÓDIGO> [--src URL]  # clave para presentar en la
web
```

El orden de prioridad es `GAANIM_POLL_RELAY`, luego `[polls] relay` del proyecto y por último el guardado con `gaanim relay use`. La dirección debe empezar por `https://` (o `http://` para probar con `npx wrangler dev`).

`gaanim relay reset` empieza una partida nueva, como **New game** en la vista del presentador: borra votos, respuestas y jugadores de la sesión del proyecto al que pertenece `RUTA`, un script o una carpeta (por defecto la actual).

`gaanim relay results` descarga la partida que el relay guarda para ese proyecto y escribe sus planillas (`jugadores.csv`, `respuestas.csv` y `preguntas.csv`) en `resultados/` dentro de la carpeta de salida del proyecto, o en `--output`. Sirve si la presentación se cerró antes de guardarlas; el relay olvida una partida doce horas después de su última actividad. Ver Guardar los resultados.

`gaanim relay key` imprime la clave de presentador de la sesión de un paquete (o de un código de sesión), para presentarla desde el reproductor web en vez de esta computadora. Con `--src` imprime el enlace completo al reproductor, terminado en `#clave=…`. Ver En el reproductor web.

50.8. `gaanim update`

Instala la versión más reciente de Gaanim sobre la carpeta del `gaanim` que lo ejecuta. No necesita Python.

```
gaanim update --check   # dice si hay una versión nueva y dónde están sus novedades
gaanim update           # la descarga y la instala
```

Busca entre las releases de GitHub la más reciente con el paquete de tu sistema (Windows o Ubuntu x64) y, si es más nueva que la que usas, descarga el paquete, comprueba su suma SHA-256, lo descomprime en `.gaanim-update/` dentro de la instalación y comprueba que el `gaanim` nuevo arranca. Solo entonces cambia los archivos: los que reemplaza pasan a `.gaanim-update/old/`, porque Windows no deja sobrescribir un programa abierto pero sí moverlo, y se borran la próxima vez que se abre Gaanim. Si un paso falla, la instalación queda como estaba. Cada proyecto recibe el paquete de autoría nuevo la próxima vez que se abre.

Un `gaanim` compilado desde el código (en la carpeta `target/` de Cargo) no se actualiza así: actualízalo con `git pull` y vuelve a compilar.

La pantalla de inicio consulta GitHub como mucho una vez al día y, si hay una versión nueva, ofrece instalarla con el mismo proceso. Para no esperarla, **Check now** en **Settings → Updates** consulta en el momento, también desde el editor. Desactiva esa consulta en **Settings → Updates** o con la variable `GAANIM_NO_UPDATE_CHECK=1`.

50.9. `gaanim --diff`

```
gaanim --diff --example <SCRIPT_O_PROYECTO> [OPCIONES]
gaanim --diff --baseline <DIR> --current <DIR> [OPCIONES]
```

Captura fotogramas exactos de una escena y los compara con un baseline aprobado y guarda un informe JSON y HTML (`report/index.html`) que se revisa en el navegador. Las capturas de `--example escenas/intro.py` viven en `tests/visual/escenas/intro/` (`baseline/`, `current/` y `report/`). El flujo completo está en Capturas y comparación visual.

Opción	Efecto
`-e, --example <SCRIPT_O_PROYECTO>`	Captura la escena y la compara con su baseline. Con un paquete `.gaanim` captura sus pausas sin Python, como `--capture-stops`.
`--bless`	Guarda la captura como baseline y termina. Sobrescribe el baseline anterior.
`--capture-only`	Escribe solo `current/` y termina, sin comparar.
`--no-capture`	Compara los PNG que ya están en `current/`.
`--capture-stops`	Captura el fotograma de cada `scene.stop()` en lugar de los tiempos de `scene.snapshots`; el script no necesita cambios.
`--stops <LISTA>`	Con `--capture-stops`, solo esas pausas: números y rangos contados desde 1, como `3-7,12`.

`--sections <LISTA>`, `--from <NOMBRE>`	Con `--capture-stops`, solo las pausas de esos segmentos, con las mismas reglas que al previsualizar.
`--height <PX>`	Captura los fotogramas con esa altura en píxeles (de 16 a 8192); el ancho sigue la proporción de la escena. Por defecto, la altura de la baseline que ya existe, o 960 en el lado largo si no hay ninguna. Una captura más chica es más rápida para revisar muchas pausas; si su tamaño no coincide con el de la baseline, `--diff` lo dice en vez de comparar. No se aplica a paquetes `.gaanim`.
`--tests-root <DIR>`	Carpeta raíz de las capturas; por defecto `tests/visual`.
`-b`, `--baseline <DIR>`, `-c`, `--current <DIR>`	Modo manual: compara dos carpetas cualesquiera.
`-o`, `--output <DIR>`	Carpeta del informe.
`--pixel-threshold <0..255>`	Diferencia por canal que se ignora.
`--max-changed-ratio <0..1>`	Fracción de píxeles distintos que se tolera.

Sin `--capture-stops`, el script elige los instantes: cuando existe la variable de entorno `GAANIM_SNAPSHOTS`, debe llamar a `scene.snapshots` con esa ruta.

```
gaanim --diff --example escenas/intro.py --bless
gaanim --diff --example escenas/intro.py --pixel-threshold 4
gaanim --diff --example mi-charla --capture-stops --stops 3-7 --capture-only
```

50.10. Mensajes para herramientas: `--message-format json`

`gaanim check`, `gaanim export` y la vista previa aceptan `--message-format json`: cada línea que escribirían para una persona sale como un objeto JSON en una línea de la salida estándar, para que un editor de código o un script la lea sin interpretar texto. Sin la opción, o con `--message-format human`, la salida es la de siempre. Los tracebacks de Python siguen saliendo por la salida de errores, y lo que el script escriba con `print` sale tal cual: descarta las líneas que no sean un objeto JSON.

Cada objeto tiene un `type`:

- `message`: una línea de estado, con `level` (`debug`, `info`, `success`, `warning` o `error`), `label`, `message` y el `file`, `line` y `column` del script a los que se refiere, o `null`. Un error del script trae la línea que nombra su traceback; `column` todavía es siempre `null`.
- `detail` (`key`, `value`) y `hint` (`message`): las líneas que acompañan a un mensaje.
- `diagnostic`: un error o aviso de `check`, con `level`, `message`, `file`, `line` y `column`. Un aviso sobre un segmento trae la línea que lo creó.
- `check`: el veredicto de `check`, con `result` (`pass` o `fail`), `errors`, `warnings`, `segments`, `stops`, `duration`, `width` y `height`. El código de salida es el de siempre.
- `progress`: el avance de una exportación, con `current`, `total` y `percent`, una vez por cada punto porcentual. La barra de progreso no se dibuja.

- `reload`: en la vista previa, `phase` `start` cuando un cambio vuelve a ejecutar el script (con `cause`, `source` o `assets`) y `end` cuando la escena nueva ya se ve, con `seconds` y lo que tardaron `python`, `replay` y `seek`.

```
gaanim check mi-charla --message-format json
gaanim export mi-charla -o charla.mp4 --message-format json
gaanim mi-charla --message-format json --listen stdio
```

```
{"type":"diagnostic","level":"warning","message":"segment `Problema` has no speaker
notes","file":"C:/charla/main.py","line":31,"column":null}
{"type":"check","script":"C:/charla/
main.py","kind":"presentation","result":"pass","errors":0,"warnings":1,"segments":6,"stops":9,"dura
{"type":"progress","current":1530,"total":5070,"percent":30}
{"type":"message","level":"error","label":"python","message":"NameError: name
'cirlce' is not defined","file":"C:/charla/main.py","line":17,"column":null}
```

Con `--listen stdio`, los eventos del canal de control comparten la salida estándar; se distinguen porque llevan `event` en lugar de `type`.

50.11. Ajustes

Settings reúne las preferencias que Gaanim guarda para ti, las mismas en todos tus proyectos. Se abre igual en la pantalla de inicio (**Settings** en la barra lateral) y en el editor (el botón de ajustes a la derecha de la barra de reproducción, o **More controls → Settings** si la ventana es estrecha), y en los dos con `Ctrl+,`; `Esc` la cierra. Cada cambio se guarda al momento. A su lado, el botón de la casa (**Back to Home**) cierra el proyecto y vuelve a la pantalla de inicio: Gaanim se abre de nuevo sin él.

Sección	Qué ajusta
Recording	El nivelado de cada toma nueva y su volumen final en LUFS (−16 por defecto), la cuenta atrás de 3 s antes de grabar y si se guarda el audio de cada grabación de la charla para volver a escucharlo.
Transcription	whisper.cpp: el ejecutable `whisper-cli` (vacío: lo busca en el `PATH`), el modelo `ggml-*.bin`, el idioma y si cada toma nueva se transcribe al guardarla.
Dictation	Zoom Scribe: el token (pegado para la sesión, o guardado en el Administrador de credenciales de Windows), el idioma, que en el editor se puede fijar en el `gaanim.toml` del proyecto, el máximo de minutos por sesión y si se envían los nombres de diapositivas y stops como vocabulario.
Prompter	El ritmo al que se ilumina el guion: **Automatic** (el de tus grabaciones, si no tu ritmo medido al dictar, si no 150 palabras por minuto) o **Fixed**, el mismo para todos los tramos, entre 80 y 240 palabras por minuto. También el tamaño del texto. Ver El prompter.
Preview	La resolución de la previsualización del editor: **Automatic** la baja mientras la reproducción no llega a 60 fps, **Full** dibuja siempre todos los píxeles y **Fixed** dibuja una fracción. Se aplica al momento y nunca afecta a las exportaciones.
Integrations	El comando del editor de código que abre una línea del script, con `{file}` y `{line}` (vacío: prueba `code`, `cursor`, `codium`, `zed` y `subl`), y el relay de las encuestas, el mismo que guarda `gaanim relay use`.

Updates	**Check now** consulta GitHub en el momento, aunque la consulta diaria esté desactivada, y si hay una versión nueva la instala y reinicia Gaanim como `gaanim update`. Debajo, si la pantalla de inicio consulta GitHub una vez al día para avisar de versiones nuevas.

Todo se guarda en `settings.json`, en la carpeta de datos de Gaanim (`%LOCALAPPDATA%\Gaanim\Gaanim\data` en Windows, `~/.local/share/gaanim` en Linux), con una sección por área: `narration`, `prompter`, `preview`, `integrations`, `polls`, `updates` y `home` (la vista de los recientes en el Inicio). El pie de la ventana muestra la ruta y **Show folder** la abre. Las versiones anteriores guardaban parte de esto en `narration.json` y `poll-relay.txt`; la primera vez que Gaanim lee los ajustes los pasa a `settings.json` y borra esos archivos.

Las variables de entorno siguen mandando sobre lo guardado: si `GAANIM_EDITOR`, `GAANIM_PREVIEW_RESOLUTION`, `GAANIM_POLL_RELAY` o `GAANIM_NO_UPDATE_CHECK` están definidas, su ajuste aparece bloqueado y dice qué variable lo fija.

50.12. Variables de entorno

Variable	Efecto
`GAANIM_SNAPSHOTS`	La define `gaanim --diff` con la carpeta donde `scene.snapshots` debe capturar. No la definas a mano.
`GAANIM_EDITOR`	Comando con el que el inspector, la consola, las animaciones, el panel de error y el icono del guion de **Script & narration** abren una línea del script o del guion, con `{file}` y `{line}` donde van la ruta y el número de línea: `code --goto {file}:{line}`, `zed {file}:{line}`, `idea --line {line} {file}`. Las comillas agrupan palabras con espacios. Sin `{file}`, la ruta se añade al final. Sin la variable, Gaanim usa el comando de **Settings → Integrations** y, si está vacío, prueba `code`, `cursor`, `codium`, `zed` y `subl` en el `PATH`; si no encuentra ninguno abre el archivo con la aplicación predeterminada.
`GAANIM_INCREMENTAL=0`	Desactiva la recompilación incremental del hot reload: cada guardado recompila la escena entera.
`GAANIM_RELOAD_PROFILE=1`	Escribe en la terminal qué costó cada recarga: el tiempo del script y de sus segmentos más lentos, las mediciones con `bounds()` (cuántas se compilaron aparte, cuántas compilaron la escena y en qué línea del script, cuántas reanudaron una compilación anterior, cuántas la reutilizaron), las fases del replay (las huellas que deciden qué segmentos se reutilizan, la limpieza o la restauración de la escena anterior, la compilación), los segmentos que más tardaron en compilarse y la captura del fotograma de t=0. Al abrir un script, también cuánto tardó cada fase del arranque: el entorno de Python, el

	intérprete, la preparación de la app y de la GPU, el primer fotograma y la primera escena lista.
`GAANIM_PREVIEW_RESOLUTION`	Resolución de la previsualización del editor. Con `auto` (por defecto), si al reproducir no llega a 60 fps porque dibujar cuesta demasiado, baja al 75 % y luego al 50 % de los píxeles, y vuelve a la resolución completa al pausar. `full` la mantiene siempre completa; una fracción como `0.5` la fija. Nunca afecta a las exportaciones. Sin la variable manda **Settings → Preview**.
`GAANIM_FRAME_PROFILE=1`	Reproduce la línea de tiempo entera sin detenerse en las pausas y escribe en la terminal, una vez por segundo, el coste del seek, de la composición y del render, la resolución de la previsualización, el tamaño de la escena de Vello (caminos, segmentos y recortes, el mayor del segundo; un número alto de recortes suele venir de sombras o máscaras por glifo) y cuántos objetos están visibles. Al terminar lista los segundos más lentos y cierra el editor.
`GAANIM_SEGMENT_CHECKPOINTS=0`	Desactiva los puntos de control por segmento. Por defecto, al reproducir o recorrer un segmento, Gaanim guarda el estado de la escena en su inicio y cada fotograma parte de ahí, así que el coste de un fotograma no crece a lo largo de la presentación. Con `0`, cada fotograma se reconstruye desde el principio de la escena. El resultado es idéntico en ambos casos.
`GAANIM_CHECKPOINT_TIMINGS=1`	Escribe en la terminal cuánto tarda cada punto de control por segmento en guardarse.
`GAANIM_POLL_RELAY`	Dirección del relay de las encuestas. Tiene prioridad sobre `[polls] relay` y sobre el relay guardado con `gaanim relay use` o en **Settings**.
`GAANIM_NO_UPDATE_CHECK=1`	La pantalla de inicio no consulta GitHub para avisar de versiones nuevas, como al desmarcarlo en **Settings → Updates**. `gaanim update` sigue funcionando.
`GAANIM_POLL_SESSION`	Código de seis caracteres que usan las encuestas en lugar del que Gaanim guarda para el proyecto, por ejemplo para que varias personas presenten el mismo proyecto o para capturas reproducibles.

CAPÍTULO

51. Manifiesto gaanim.toml

Campos, valores predeterminados y validación del manifiesto de proyecto

51.1. Manifiesto `gaanim.toml`

Una carpeta es un proyecto de Gaanim cuando contiene un `gaanim.toml`. El manifiesto dice qué script ejecutar y dónde están los recursos y las salidas. `gaanim init` lo genera así:

```
name = "mi-charla"
kind = "slides"
entry = "main.py"
assets_dir = "assets"
output_dir = "exports"
```

51.1.1. Campos

Campo	Tipo	Predeterminado	Significado
`kind`	cadena	obligatorio	Tipo de proyecto: `"video"` o `"slides"`.
`entry`	ruta	obligatorio	Script que ejecutan `gaanim <carpeta>`, `check` y `export`.
`name`	cadena	nombre de la carpeta	Nombre que muestran el Inicio y el título de la ventana.
`assets_dir`	ruta	`"assets"`	Carpeta de recursos, relativa al manifiesto. La lee `scene.assets.load_project()`.
`output_dir`	ruta	`"exports"`	Carpeta donde el diálogo de exportación del editor propone guardar, y donde van las capturas del fotograma.
`language`	cadena	ninguno	Idioma en que se habla la charla, como etiqueta BCP 47 (`"es-ES"`, `"es"`). Lo usa el dictado del guion.
`polls.relay`	URL	ninguno	Relay de las encuestas de este proyecto, en una tabla `[polls]`.

Las claves desconocidas se ignoran. Las que parecen credenciales, porque su nombre contiene `token`, `secret`, `password` o `api_key`, también, pero `gaanim check` y el panel **Script & narration** avisan de ellas: el manifiesto viaja con el proyecto, también a un repositorio, así que los tokens no van ahí (ver El token).

51.1.1.1. `kind`

`"video"` y `"slides"` distinguen los dos tipos de proyecto en el Inicio y en la plantilla de `gaanim init`. No cambian cómo se ejecuta ni cómo se exporta la escena: `gaanim check` aplica las comprobaciones de presentación cuando la escena usa segmentos, sea cual sea su `kind`. Los valores

antiguos `"presentation"` y `"thesis"` producen un error que pide cambiarlos por `"slides"`; cualquier otro valor, un error con los tipos disponibles.

51.1.1.2. `entry`

Ruta relativa a la carpeta del proyecto. Debe apuntar a un archivo que exista y no puede ser absoluta ni salir de la carpeta con `..`. Así, `gaanim .`, `gaanim check .` y `gaanim export . --output …` no necesitan la ruta del script.

51.1.1.3. `assets_dir`

El editor y la CLI no aplican `assets_dir` por su cuenta: la escena lo carga al llamar a `scene.assets.load_project()`, que busca el `gaanim.toml` en la carpeta del script que la llama y, si no está, en sus carpetas superiores, como hace `git` con `.git`: un módulo dentro de un paquete del proyecto encuentra el manifiesto del proyecto. Después, las rutas relativas de `scene.media.image`, `svg`, `lottie`, `video` y demás se resuelven desde esa carpeta, sin depender del directorio de trabajo desde el que se lanzó Gaanim.

```python
from gaanim import Scene

scene = Scene()
scene.assets.load_project()             # lee assets_dir del gaanim.toml del proyecto
logo = scene.media.image("logo.webp")   # es assets/logo.webp
```

`load_project` lee solo la clave `assets_dir` de nivel superior, escrita como cadena entre comillas; si el manifiesto no la tiene, usa `"assets"`, igual que la CLI y el editor. Consulta `AssetManager.load_project`.

51.1.1.4. `output_dir`

Carpeta relativa al proyecto, o absoluta. El diálogo de exportación del editor propone guardar ahí, con el nombre del proyecto (`<output_dir>/mi-video.mp4`), y las capturas del fotograma actual van a `<output_dir>/captures/`. `gaanim export` no la usa: escribe en la ruta de `--output`, relativa al directorio actual.

51.1.1.5. `language`

```toml
language = "es-ES"
```

El idioma en que se habla la charla del proyecto, como etiqueta BCP 47: `"es-ES"`, `"es"`, `"es-PE"`, `"en-US"`… Al dictar el guion con Scribe manda sobre el idioma elegido en el editor, así que la charla se dicta igual en cualquier equipo. Scribe usa la misma etiqueta si la tiene y, si no, la primera de ese idioma: `"es"` y `"es-PE"` se dictan como `es-ES`, y `"pt"` como `pt-BR`. Si Scribe no transcribe ese idioma, se usa el del editor y el panel lo avisa.

En **Settings → Dictation**, con el proyecto abierto en el editor, **Pin in gaanim.toml** escribe el idioma elegido y **Unpin** lo quita; el resto del archivo, comentarios incluidos, queda como estaba.

51.1.1.6. `[polls]`

```toml
[polls]
relay = "https://gaanim-relay.<tú>.workers.dev"
```

Relay al que apuntan los códigos QR de las encuestas (`scene.poll`) de este proyecto, en lugar del guardado con `gaanim relay use`. `GAANIM_POLL_RELAY` tiene prioridad sobre ambos. Ver Tu primera encuesta.

51.1.2. Cómo se encuentra el proyecto

- `gaanim <carpeta>` exige un `gaanim.toml` en esa carpeta.
- `gaanim <script.py>` ejecuta ese script y busca un `gaanim.toml` en su carpeta y en hasta cinco carpetas superiores para mostrar el nombre del proyecto y proponer la carpeta de exportación.

Al abrir una carpeta, un manifiesto que no es TOML válido, sin `kind` o `entry`, o con valores inválidos detiene `gaanim`, `check` y `export` con un mensaje que incluye la ruta del manifiesto y el código de salida 2. Al abrir un script, un manifiesto inválido en una carpeta superior se ignora.

PARTE VII

Apéndices

Novedades de cada versión, glosario y solución de problemas

CAPÍTULO

52. Novedades

Cambios visibles de cada versión de Gaanim y cómo migrar entre ellas

Qué cambió en cada versión, de la más reciente a la más antigua. Cada sección indica también qué tienes que ajustar en tus escenas al actualizar. Para instalar una versión nueva, sigue Instalación.

52.1. 0.15.0

Sin publicar. Los gráficos declarativos dibujan tartas y donuts con la marca `arc`, que se abren porción a porción al entrar y reparten sus porciones por clave al cambiar de datos, y la paleta categórica se elige: el tema la fija para toda la escena y cada campo puede tener la suya.

52.1.1. Gráficos

- `ChartSpec.mark("arc")` dibuja una porción por fila según su parte del total del canal `theta`, en el sentido del reloj desde las 12. Sus opciones son `inner_radius` (un donut), `pad_angle` (la separación entre porciones), `start_angle`, `radius` y `label_color`; el canal `label` escribe cada fila en el centro de su porción y un campo `radius` dibuja un gráfico de rosa.
- `chart.animate.grow()` hace crecer la tarta: una manecilla gira desde el ángulo inicial, cada porción se abre en su sitio cuando la manecilla la cruza y su etiqueta aparece al completarse.
- `chart.animate.to(...)` entre dos gráficos de arcos lleva cada porción a la de su clave barriendo sus ángulos, en lugar de deformar su contorno; las porciones que salen se cierran y las que llegan se abren.
- `Palette` reparte colores entre categorías: una incluida por nombre (`gaanim`, `tableau10`, `okabe_ito`, `set2`, `dark2`, `paired`, `tab10`…), tus propios colores o los de un `ColorMap`, con `reversed()` y `rotate()`.
- `Theme(palette=...)` colorea las categorías de todos los gráficos, leyendas, tartas y carreras de barras, y `scene.canvas.palette()` la devuelve para tus propios objetos. `Scale.category().colors(...)` acepta también una `Palette` o un nombre.
- Con más categorías que colores, la paleta vuelve a empezar algo más clara u oscura en cada vuelta, en lugar de repetir colores.

52.1.2. Al actualizar

- Un gráfico de más de cuatro categorías sin colores propios usa ahora diez colores en lugar de repetir los cuatro primeros, y una carrera de barras sin `colors` toma la paleta del tema (`gaanim`) en lugar de `tableau10`. Para recuperar sus colores, pasa `colors=Palette("tableau10").colors()` o usa `Theme(palette="tableau10")`.

52.2. 0.14.0

Publicada el 15 de octubre de 2026. El editor reúne tus preferencias en una ventana **Settings**, se maneja entero desde el teclado con una paleta de comandos y una lista de saltos con vista previa, y redibuja la línea de tiempo con secciones, rombos en las pausas y un cabezal dorado. Home muestra los recientes como tarjetas con portada. El panel **Script & narration** se acopla a la derecha con un objetivo para la charla, compartido con Presenter View, y prácticas cronometradas contra él (ver

Ensayar y grabar una charla). Corrige las exportaciones que se quedaban en 0 % al abrir Gaanim desde Neovim o VS Code y el bloom que se convertía en una mancha blanca con la ventana grande.

52.2.1. Ajustes

- **Settings** (`Ctrl+,`, la barra lateral de Home o **More controls**) reúne en una ventana lo que antes estaba repartido: **Recording**, **Transcription** y **Dictation** (antes plegables del panel de narración), **Prompter**, **Preview**, **Integrations** y **Updates**. Cada cambio se guarda al momento en un solo `settings.json` en la carpeta de datos de Gaanim, y la ventana reabre en la última sección. **Reset section** devuelve la sección a sus valores por defecto; el token de Scribe y el relay guardado conservan sus botones.
- El prompter lee a un ritmo automático (el de la grabación, si no el medido al dictar, si no 150 palabras por minuto) o a uno fijo, de 80 a 240, y recuerda el tamaño del texto.
- La resolución de la vista previa, el comando del editor de código y la comprobación diaria de actualizaciones, que solo eran variables de entorno, tienen su ajuste. Las variables siguen mandando, y **Settings** muestra el ajuste bloqueado con su nombre. El comando del editor avisa si su programa no está en el `PATH`, y las rutas de Whisper que no llevan a nada salen en rojo.
- **Updates → Check now** busca una versión nueva al instante, aunque la comprobación diaria esté apagada, y actualiza también desde el editor.
- **Preview** elige qué cuenta la insignia de la esquina tras cada recarga (avisos y errores, solo errores o nada) y si aparece el aviso «Scene ready». La × de la insignia la oculta desde entonces.

52.2.2. Editor

- `Ctrl+Shift+P` abre una paleta de comandos: cada acción del editor, cada sección de **Settings**, **Back to Home** y los proyectos recientes, filtrados al escribir y con su atajo. `Ctrl+P` lista los segmentos y sus stops por nombre, con la diapositiva tal como se ve allí y sus notas, y `Enter` salta. `?` muestra todos los atajos.
- La barra de reproducción ancha trae **Settings** y **Back to Home**, que vuelve a Home sin cerrar Gaanim; la intermedia muestra tantos botones como caben. Al abrir o recargar un proyecto se queda visible unos segundos.
- La línea de tiempo nombra las secciones de los segmentos llamados `Sección · parte` en una fila encima, alterna dos grises entre segmentos, dibuja los stops como rombos (huecos hasta llegar a ellos) y los marcadores de `scene.marker` en violeta. Una línea sobre cada diapositiva dice cómo va su guion, y el cabezal es dorado.
- La ventana reabre con el tamaño y la posición que tenía, y **Back to Home** o cambiar de proyecto preguntan antes si hay una grabación en curso.
- La interfaz sigue la paleta de la marca: grafito, violeta para acentos y selección, dorado para las pausas, casillas, deslizadores y desplegables a juego, y JetBrains Mono en el texto monoespaciado. Home muestra el logo.
- Una exportación que falla ofrece **Try again**, **Copy error** y, si falta FFmpeg, cómo instalarlo; un error de Home que nombra Python, uv o FFmpeg lleva a la página del entorno.

52.2.3. Home

- **Recent** alterna entre la lista y una cuadrícula de tarjetas con la portada de cada entrada, y recuerda la elección. El editor guarda la portada de un proyecto al cargar la escena: `scene.thumbnail(t)`, si no el primer stop, si no el instante del primer segmento que más muestra; siempre en la carpeta de datos de Gaanim, nunca en el proyecto.
- Un paquete `.gaanim` exportado dentro de la carpeta de un proyecto reciente va con él, con un botón que lo reproduce. Un campo busca por nombre o ruta, y una chincheta fija una entrada arriba, también tras **Clear list**.

52.2.4. Script & narration y prácticas

- El panel se acopla a la derecha y la escena se encoge a su lado. Muestra el objetivo de la charla (**TARGET**), que es el mismo que fija Presenter View, lo que suman las tomas y la última práctica frente a él, las últimas prácticas, un mapa de la charla, el guion de la diapositiva en pantalla con sus palabras y su tiempo de lectura, una lista de diapositivas filtrable (**No time**, **Over**, por nombre) y los controles para grabar o practicar (ver El panel Script & narration).
- Cada diapositiva recibe su parte del objetivo: su `target=`, si no lo que dejan los objetivos, repartido según cuánto dice su guion.
- **Practice** presenta la charla cronometrada sin grabar nada y la guarda como un ensayo, el mismo registro que llena Presenter View. El prompter dice en cada momento si vas adelantado o retrasado (ver Practicar contra un objetivo).
- **Test level** abre el micrófono para ver su nivel sin grabar, y el medidor y el guion de la diapositiva actual están siempre a mano.
- Nueva guía, Ensayar y grabar una charla, con un proyecto de ejemplo y capturas: del guion a una charla que cabe en su tiempo, y de ahí al vídeo o a la presentación.

52.2.5. Correcciones

- Las exportaciones desde el editor abierto por un plugin de Neovim o VS Code (`--listen stdio`) se quedaban en 0 % y fallaban a los 120 segundos: el proceso de exportación heredaba la tubería de entrada del plugin. Ya no hereda la entrada estándar.
- Con la ventana grande, una escena con muchos objetos y post-proceso (bloom, aberración cromática) se llenaba de una mancha blanca: Vello se saltaba el fotograma por falta de memoria en la GPU y el post-proceso se aplicaba una y otra vez sobre el anterior. La vista previa detecta ahora esos fotogramas y los repite con más memoria, como ya hacía la exportación.
- Una grabación `.gaanim` que falla o se cancela ya no deja `nombre.gaanim.partial`.
- Las flechas (→) se dibujan en el texto de la interfaz, y la línea de estado del prompter recorta el nombre de la diapositiva en lugar de montarse sobre el aviso de avance.

52.2.6. Al actualizar

- Las preferencias de `narration.json` y `poll-relay.txt` pasan solas a `settings.json` la primera vez, y esos archivos se borran.
- `narration/` (el guion y las tomas) vive en la carpeta de assets del proyecto aunque el script no llame a `scene.assets.load_project(...)`. Un proyecto que ya guardaba `narration/` junto al script, y no en sus assets, sigue igual. Una ruta relativa de `scene.narration_script(...)` se busca entonces en esos assets.

52.3. 0.13.1

Publicada el 14 de octubre de 2026. Corrige el reproductor de los paquetes `.gaanim` en formato 6: uno largo, con las tablas en varios segmentos, se quedaba congelado en la primera pausa o en el primer salto de la línea de tiempo. No hace falta volver a exportar nada.

52.3.1. Cambios

- Los paquetes en formato 6 se reproducen enteros y saltan a cualquier instante. Un segmento guarda las recetas de dibujo de todos los bloques que las usaron primero, y al cargarlo el reproductor las leía todas, también las que remiten a segmentos que el bloque abierto no carga; fallaba con «a frame refers to a path of a segment it does not list» y se quedaba en el último fotograma. Ahora cada receta se lee la primera vez que un fotograma la dibuja. `gaanim check` no lo detectaba porque recorre los fotogramas en orden.

52.3.2. Al actualizar

- Los paquetes grabados con 0.13.0 son correctos: 0.13.1 los reproduce tal cual, sin `gaanim pack` ni una nueva exportación.

52.4. 0.13.0

Publicada el 14 de octubre de 2026. El guion de la charla pasa a un archivo Markdown con una sección por diapositiva y marcas donde avanzas, que Presenter View y el editor muestran en un prompter. La charla se dicta con Scribe y se graba diapositiva a diapositiva para ajustar sus tiempos, y Presenter View cuenta hacia atrás la duración prevista y dice si vas adelantado o retrasado. Un editor de código puede manejar la vista previa (`--listen`) y leer los mensajes de Gaanim en JSON, y `gaanim inspect` da la línea del script de cada parte de la escena. Los paquetes `.gaanim` pasan al formato 6, que se abre con mucho menos y guarda el 3D como mallas, y `gaanim pack --web` los aligera para publicarlos. El reproductor ligero se maneja desde la página que lo inserta, con el elemento `<gaanim-player>`, y el playground descarga una página HTML con la animación dentro. Toda la interfaz del editor está en inglés. Lee «Al actualizar»: un `scene.stop()` dentro de la transición de entrada de su diapositiva cambia de momento, y el guion manda sobre las `notes=`.

52.4.1. Guion y prompter

- El guion de la charla vive en un archivo Markdown, el mismo de la narración (`narration/script.md`, o el que nombra `scene.narration_script(...)`): una sección `## nombre` por diapositiva, con el nombre de su segmento, y una línea `[stop]` o `[stop nombre]` (también `[stop: nombre]`) donde avanzas. El texto antes de la n-ésima marca se dice mientras la diapositiva espera en su n-ésimo stop. La sección de una diapositiva sustituye a sus `notes=`; las diapositivas sin sección conservan las suyas, que también admiten marcas. Los encabezados pueden ser cualquier nombre de diapositiva (antes debían ser una clave de toma válida), los `###` son notas que no se muestran y los comentarios HTML se ignoran. Ver El guion de la charla.
- **Script & narration → Create script** escribe el archivo con una sección por diapositiva, a partir de sus `notes=`, y una marca por stop. Cuando el código gana diapositivas o stops, **Open and complete** añade lo que falta sin tocar lo escrito (los archivos CRLF siguen en CRLF) y abre el guion en tu editor de código (`GAANIM_EDITOR`, o uno conocido en el `PATH`) en lo primero que añadió.
- `gaanim check` avisa cuando una diapositiva tiene más o menos marcas que stops, cuando una marca con nombre no coincide con su stop y cuando una sección no corresponde a ninguna diapositiva ni bloque de voz.
- Al guardar el guion, el editor actualiza las notas sin volver a compilar la escena, y los paquetes `.gaanim` lo llevan dentro con sus marcas. La exportación a PDF con `--notes` imprime cada marca como una flecha (`→ grafico`).
- El prompter muestra el tramo que toca decir como un subtítulo, palabra a palabra y siguiendo la línea de tiempo, con el aviso de avanzar (`▶ stop`, o el nombre de la siguiente diapositiva), que late cuando la diapositiva ya espera allí. En Presenter View, `T` llena con él la columna del orador; en el editor va sobre la barra de reproducción, con `Shift+T` o **More controls → Prompter**. Lee al ritmo con que grabaste ese tramo, o al tuyo medido en tus dictados, o a 150 palabras por minuto. Ver El prompter.
- En la barra de tiempo del editor, un cuadrado en la esquina de cada diapositiva dice cómo va su guion: verde si todos sus tramos tienen texto, ámbar si solo algunos.

52.4.2. Dictar y cronometrar la charla

- **Script & narration → Record the talk** presenta la charla desde la primera diapositiva y guarda el tiempo de cada una y de cada tramo. **TIMING** los compara con sus objetivos y el total con la

duración prevista, con un veredicto: cuánto queda, cuánto sobra o que cabe. **Re-record** vuelve a grabar una diapositiva o un solo stop, y **Keep the audio, to listen back** guarda el audio de cada tramo para escucharlo desde su fila. Los tiempos y el audio se guardan en la carpeta de datos de Gaanim, nunca en el proyecto. Ver Grabar la charla y ajustar sus tiempos.

- Mientras grabas, el micrófono puede ir a Scribe Live, de Zoom, y cada frase que transcribe pasa al tramo del guion que estabas diciendo; el texto anterior queda debajo en un comentario `<!-- previous: … -->`. El token se escribe en el panel, se guarda en el Administrador de credenciales de Windows o se lee de `GAANIM_SCRIBE_TOKEN`, y nunca entra en el proyecto. A Zoom solo le llegan el audio y los ajustes de la sesión. Una sesión a la vez, sin reconectar tras un fallo, y se cierra sola en su límite de minutos (60 por defecto), tras 2 minutos sin voz o al pulsar `Esc`. Ver Dictar el guion con Scribe.
- La clave `language` de `gaanim.toml` (`es-ES`, `es`, `es-PE`…) dice en qué idioma se da la charla, para Scribe y otras herramientas de voz; **Pin in gaanim.toml** la escribe sin tocar el resto del archivo. `gaanim check` avisa de las claves que parecen credenciales (`token`, `secret`, `password`, `api_key`), que Gaanim nunca lee. Ver `language`.

52.4.3. Presenter View

- `scene.slides.target(20 * 60)` fija la duración de la charla y `scene.segment(..., target=120)` la de cada diapositiva, en segundos. La de la charla también se elige en Presenter View, con el botón junto a `LEFT`, o con `gaanim --present --duration 20m` (`1h30m`, `90s`, `20:00` o minutos). `LEFT` cuenta hacia atrás lo que queda, `PACE` dice cuánto vas adelantado o retrasado según los objetivos, y bajo el nombre de la diapositiva se lee `0:45 of 1:30 on this slide` y lo que tardaste en el ensayo anterior. Cada presentación de más de 30 segundos se guarda como ensayo (los 20 últimos por archivo; en el reproductor web, durante la sesión). Los paquetes `.gaanim` guardan los objetivos. Ver El tiempo de la charla.

52.4.4. Editores de código

- `gaanim --listen stdio` (o `--listen 127.0.0.1:PUERTO`) deja que un editor maneje la vista previa con JSON por líneas: `seek` a un tiempo, segmento, `Section` o stop; `follow` a una línea del script; `play`, `pause`, `loop` de un segmento o un tramo y `unloop`. Gaanim responde con los eventos `ready` (con la línea de tiempo de la escena), `reload`, `playhead` (tiempo, segmento, stop, si reproduce y el bucle), `failed` (el error del script con su línea) y `error`. Solo escucha en la propia máquina. Ver Controlar la vista previa desde un editor.
- `gaanim inspect mi-charla [--json]` lista los segmentos, `Section`, stops, marcadores, bloques de voz, la toma en directo y cada `play` con su tiempo y la línea del script que los crea. Ver `gaanim inspect`.
- `--message-format json` en `gaanim check`, `gaanim export` y la vista previa escribe cada mensaje como un objeto JSON por línea: mensajes con su archivo y línea, diagnósticos y veredicto de `check`, progreso por porcentaje y recargas con sus tiempos. Ver Mensajes para herramientas.
- Todo es opcional: sin estas opciones, la vista previa y la CLI se comportan como antes, y `gaanim init` no añade nada propio de ningún editor.

52.4.5. Exportación

- `gaanim export --frame`, `--from` y `--to` aceptan, además de segundos y marcadores, el nombre de un segmento, una clave de `Section` o el nombre de un stop. `--frame` de una diapositiva toma su primer stop, el mismo fotograma que su página del PDF; `--from` empieza en su comienzo y `--to` acaba en su final. Un nombre compartido por varios stops es ambiguo y el error los lista con su tiempo. Ver Tramos y marcadores.

52.4.6. Paquetes

- Formato 6: las tablas (trazados, imágenes, recetas, textos, shaders y mallas 3D) van en segmentos de unos 2 MB, y abrir un paquete solo lee su manifiesto, la escena, el primer bloque y los

segmentos que usa. En los 170 ejemplos, lo que hace falta para abrirlos baja un 74 %. Gaanim y los reproductores siguen abriendo los paquetes en formato 5.
- Las escenas 3D se guardan como mallas, matrices y luces, una sola vez, y se proyectan de nuevo al leerlas: `gallery/surface_plot` pasa de 54 MB a 1,1 MB y `3d/camera_3d` de 15 MB a 0,16 MB.
- `gaanim pack entrada.gaanim salida.gaanim` reescribe un paquete en el formato actual, fotograma a fotograma. Con `--web`, guarda los trazados en una rejilla de menos de 1/32 de píxel en 4K y deja fuera lo que solo usan `gaanim check` y el desenfoque de movimiento: los ejemplos ocupan un 39 % menos y abrirlos necesita un 89 % menos. Ver Paquetes.
- Abrir un paquete ya no espera a las huellas de los fotogramas, que solo usa `gaanim check`, y la portada se guarda con la mejor compresión (la mitad de tamaño).

52.4.7. Reproductor web

- El reproductor ligero se maneja desde la página que lo inserta, en cualquier sitio: recibe `gaanim:command` (`play`, `pause`, `toggle`, `seek`, `loop`, `mute`) y envía `gaanim:status` con el tiempo, la duración y su estado; el reproductor completo responde a los mismos mensajes. Ver Controlarlo desde la página.
- El elemento `<gaanim-player>` (`insertar/gaanim-player.js`) inserta el reproductor con la proporción de la escena, sin escribir `aspect-ratio`, y ofrece `play()`, `pause()`, `seek()`, `currentTime`, `duration`, `paused`, `ended`, `loop` y `muted`, con sus eventos.
- Arranca antes: descarga el archivo mientras compila el módulo, y un archivo grande que se lee por partes solo baja el resto mientras se reproduce; `preload=auto` lo descarga entero como antes.
- Cuando el reproductor no tiene la forma de la escena, oscurece lo que la rodea y marca su borde, como la vista previa del editor.
- El playground descarga la grabación como una página `.html` con el reproductor ligero dentro, que se abre con doble clic, sin servidor, o va en un `<iframe>`: en **Insertar** o en el menú **Descargar**. Ver Una página HTML.

52.4.8. Editor

- Toda la interfaz del editor está en inglés, como ya lo estaba Presenter View: la pantalla de inicio, la barra de reproducción, la exportación, el inspector, la consola, la línea de tiempo de animaciones, las encuestas, **Script & narration** y los mensajes del reproductor web. Las claves de los enlaces (`segmento=`, `pausa=`) y los nombres de los archivos de resultados no cambian.

52.4.9. Correcciones

- Un `scene.stop()` escrito mientras la transición de entrada de su diapositiva aún corre espera a que termine. Antes, Presenter View, el PDF y `--capture-stops` mostraban la diapositiva a medio entrar.
- `gaanim check` verifica los paquetes con grupos que comparten un brillo, un desenfoque o una sombra; antes rechazaba sus fotogramas (92 de 307 en `effects/glow_blur_shadow`).

52.4.10. Al actualizar

- Un `scene.stop()` dentro de la transición de entrada de su diapositiva se mueve al final de la transición, y lo que viene después se retrasa lo mismo. Los enlaces y las capturas tomadas en ese stop cambian de momento.
- Si tu proyecto ya tiene `narration/script.md`, una sección con el nombre de una diapositiva sustituye a sus `notes=`. Un guion que no se puede leer (un encabezado vacío o repetido) hace que `scene.segment` lance `ValueError`, como ya hacía `scene.voiceover`.
- Los paquetes grabados con 0.13.0 están en formato 6, que las versiones anteriores de Gaanim no abren. `gaanim pack` pasa los de formato 5 al actual, más rápidos de abrir en la web.
- Vuelve a grabar los paquetes con efectos compartidos (brillo, desenfoque o sombra en varios objetos iguales) grabados antes para que `gaanim check` los verifique.

- Los menús y botones del editor tienen sus nombres en inglés: el panel **Narración** es ahora **Script & narration**, y **Más controles**, **More controls**.

52.5. 0.12.0

Publicada el 13 de octubre de 2026. Los ejemplos se ordenan por temas, con una sección en la web donde cada uno se reproduce sobre su captura, y una galería de escenas cortas en los colores de la marca sustituye a los ejemplos básicos y avanzados. Las diapositivas se exportan a PDF con las notas del orador, y los vídeos a cualquier frecuencia y a MOV con Apple ProRes, con canal alfa, para montarlos en un editor de vídeo. Llegan luces 3D que se colocan y se animan, el tema `gaanim`, el resaltado de sintaxis de `scene.text.code` con JetBrains Mono y un reproductor ligero que marca las pausas en su barra. Corrige los paquetes `.gaanim` de escenas 3D, cuatro fallos de `animate.transform_to` con SVG y degradados, la cámara que sigue o se acerca a un objeto reactivo y la selección de fracciones en las ecuaciones, y los rastros se dibujan como curvas suaves. Lee «Al actualizar»: varios cambios alteran cómo se ven escenas existentes (énfasis, `write`, bloques de código, cajas escaladas, ecuaciones de varias filas).

52.5.1. Exportación

- `gaanim export --fps` elige los fotogramas por segundo de cualquier vídeo, de 1 a 240, en lugar de los 30 o 60 de la calidad: un entero (`24`), una fracción (`24000/1001`) o un decimal (`23.976`, que significa esa fracción exacta). Antes solo se aplicaba al grabar un paquete, y la forma de lograr un vídeo a 24 fps era grabar un paquete a 24 y exportarlo. La duración y el audio no cambian, y el desenfoque de movimiento usa la tasa elegida. El diálogo de exportación del editor tiene el mismo selector. Ver Fotogramas por segundo.
- Un `--output` terminado en `.mov` exporta Apple ProRes, que Premiere, DaVinci Resolve y Final Cut importan sin conversión: 422 HQ, o 4444 con canal alfa con `--transparent`, con el color etiquetado como BT.709 y el audio sin comprimir. Sirve para poner rótulos y *lower thirds* de Gaanim encima de un montaje; WebM también lleva alfa, pero los editores lo leen mal. El diálogo del editor suma MOV a sus formatos y una casilla **Conservar la transparencia** para MOV, WebM, WebP y PNG.
- Un `--output` terminado en `.pdf` exporta las diapositivas: una página por cada `scene.stop()` y otra para el final, con el fotograma que muestra el Presenter en esa pausa, sin pérdida y al tamaño de `--width` y `--height`. Con `--notes`, tras cada diapositiva va una página con sus notas, en texto que se puede buscar. Sirve para repartir la charla a quien no tiene Gaanim. Un paquete `.gaanim` también se exporta a PDF, sin Python, y el diálogo del editor suma PDF a sus formatos con una casilla para las notas. Ver Diapositivas en PDF.

52.5.2. Actualizaciones

- La pantalla de inicio avisa cuando hay una versión nueva de Gaanim (consulta GitHub como mucho una vez al día) y la instala con **Actualizar**; **Reiniciar** abre la versión nueva. `gaanim update` hace lo mismo desde la terminal y `gaanim update --check` solo lo comprueba. Gaanim verifica la suma SHA-256 del paquete y que el programa nuevo arranca antes de reemplazar nada, y deja la instalación como estaba si algo falla. `GAANIM_NO_UPDATE_CHECK=1` desactiva el aviso. Ver `gaanim update`.

52.5.3. Animación

- `write()` y `unwrite()` animan como el `Write` y el `Unwrite` de Manim: cada pieza traza su contorno en la primera mitad de su parte, un tramo de Bézier tras otro, fino y centrado, en el color de su trazo, y en la segunda mitad aparece su relleno y el trazo toma su ancho. La curva se aplica al avance de cada pieza, y sin duración duran 1 s con menos de 15 piezas y 2 s con más. Se conservan `by`, `order`, `seed`, `stagger`, la tiza y la punta de lápiz. Ver `Anim.write`.

52.5.4. Texto

- `scene.text.code` resalta la sintaxis: las palabras clave, las cadenas, los números, los comentarios, las funciones y los tipos toman los colores de código del tema. Los temas de editores (`dracula`, `nord`, `solarized-*`, `gruvbox-dark`, `tokyo-night`, `catppuccin-*`) usan su esquema original; `technical`, `presentation`, `paper` y los temas propios lo derivan de la paleta, y los tokens `code_keyword`, `code_string`, `code_number`, `code_comment`, `code_function`, `code_type` y `code` de `Theme(colors=...)` lo cambian. `theme="esquema.tmTheme"` usa un esquema de TextMate o Sublime Text, `highlight=False` lo deja monocromo y `font` elige la fuente, con JetBrains Mono como respaldo. Con un `background` en el que los colores del tema no se leen, como un panel oscuro en `paper`, el texto pasa al color de fondo del tema y el acento y los tokens se ajustan hasta leerse. Antes, `language` solo se mostraba en la cabecera. Ver `Typography.code`.
- El código usa JetBrains Mono, incluida en Gaanim con todos sus pesos y su cursiva: el rol `code` (que antes pedía Consolas y, sin ella, caía en otra fuente) y los bloques de `scene.text.code` (antes DejaVu Sans Mono). Así el código se ve igual en Windows, macOS, Linux, el playground y los paquetes `.gaanim`. Un tema con `fonts={"code": ...}` sigue eligiendo la suya.

52.5.5. Matemáticas

- `scene.viz.matrix(..., fractions="inline")` escribe los racionales exactos en una línea (`3/2`), tan altos como las demás entradas; antes había que rodearlo con un `cell_factory`. Ver `Visualization.matrix`.
- `TextStyle.size` devuelve el tamaño que fija el estilo.

52.5.6. Layout

- Una porción de un `ZoneSet`, como `zonas[1:]`, es otro `ZoneSet`: se recorre, se indexa y tiene `fill()`. Ver `ZoneSet`.
- El tema tiene un color `shadow`, el de `shadow=True` en las cajas: negro suave sobre fondos claros y casi opaco sobre fondos oscuros. Cámbialo con `Theme(colors={"shadow": ...})`.

52.5.7. Estilo y 3D

- Tema integrado `gaanim`, con los colores de la marca: papel sobre grafito, acentos violeta, títulos en oro y reglas en bruma. Sus series empiezan por el violeta y el oro, y su mapa de calor va del grafito al oro pasando por la bruma y el violeta. Ver Temas.
- `scene.geometry.light_3d()` crea luces puntuales o direccionales para las primitivas con material: se colocan con `move_to_3d`, se animan y siguen a otros objetos como cualquier dibujable, y alumbran en su color. Mientras la escena muestra luces, sustituyen a la luz principal fija; `fade_in` y `fade_out` las encienden y las apagan. Ver Geometría.
- `space.surface(..., color_map=...)` colorea la superficie por altura con un `ColorMap` o con cualquier mapa integrado por su nombre (`"viridis"`, `"plasma"`…). Sin él conserva la rampa de siempre.

52.5.8. Ejemplos

- Una galería, `examples/gallery/`, con 23 escenas cortas en los colores de la marca (el tema `gaanim`), agrupadas en conceptos básicos, animaciones, gráficas, cámara y 3D, y avanzadas. Va desde el logo construido con primitivas hasta los epiciclos de Fourier, pasando por un vector que barre el plano, sumas de Riemann que se refinan, una cámara que sigue un punto, una lupa sobre un copo de Koch y una superficie 3D. Abre la sección Gallery de los ejemplos y sustituye a «Ejemplos básicos» y «Ejemplos avanzados», cuyas direcciones llevan ahora a la galería. El playground ofrece estas escenas en su menú de ejemplos.
- Los ejemplos del repositorio se ordenan por temas: `examples/animation/easings.py`, `examples/camera/camera_moves.py`, `examples/3d/primitives_3d.py`… Cada uno recorre las variantes que ofrece la API de su tema, con el tema por defecto y una etiqueta con el código que construye cada variante. `examples/examples.toml` los cataloga y `just run easings` abre uno por su nombre. Los

scripts sueltos de antes se reescribieron en inglés dentro de su tema, o pasaron a `tests/` los que eran pruebas.
- La documentación tiene una sección Examples con todos los ejemplos por tema: una portada con los temas en cinco bloques, Showcase destacado y un filtro, y una página por tema con cada ejemplo, su captura, su código y su enlace al repositorio. Sale del catálogo al compilar la documentación, así que un ejemplo nuevo aparece sin tocarla.

52.5.9. Editor

- El título de la ventana dice qué versión de Gaanim se está ejecutando: «Gaanim 0.12.0 — proyecto», con «(dev)» en una build de desarrollo, para distinguirla del release instalado. También en la Presenter View y al reproducir un paquete.

52.5.10. Reproductor web

- La barra del reproductor ligero (`reproductor/insertar/`) marca cada pausa de la escena (`scene.stop()`) en su línea de tiempo; las ya pasadas se ven en violeta, como el progreso.
- La opción `full=click` del reproductor ligero espera a un clic antes de abrir el reproductor completo para una escena que lo necesita, y avisa a la página que lo contiene para que ofrezca su propio botón (ver Insertar en un blog o sitio web).

52.5.11. Correcciones

- Las partes de una ecuación que contienen una fracción (`frac(a, b)` o `a/b`) o una función que solo compone sus argumentos (`sqrt`, `abs`, `bold`, `binom`…) se pueden seleccionar: `surrounding_rect`, `fill`, `write(by="part")` y las transformaciones las alcanzan. Antes la selección quedaba vacía y el marco o el color no aparecían. Seleccionar la fracción entera incluye su barra.
- `transform_to` entre marcas de un plano (`riemann_sum`, `plot_data`, `scatter_data`…) transforma su forma; antes solo cambiaba el color, porque la marca se redibujaba desde sus datos en cada fotograma.
- `scene.camera.animate.pan_to(dibujable)` centra el dibujable donde se dibuja, también cuando sigue a un punto reactivo, y al terminar la cámara se queda allí; antes iba a la posición con la que se declaró y la siguiente animación de cámara partía de la pose anterior a la panorámica.
- `scene.camera.animate.orbit` gira alrededor del eje `up` de la cámara: con `look_at(..., up=(0, 0, 1))`, como en los ejes 3D, rodea la escena en vez de pasar por encima. Una órbita tras otra parte de donde terminó la anterior.
- Un dibujable con `hidden()` dentro de un grupo, o una marca de un plano, espera a su propia animación: el `create` del plano o la entrada del grupo ya no lo muestran.
- Un plano con el eje X abajo (`crossing("min")`) numera el cero de su eje Y. Solo se omite el número del eje Y donde los dos ejes se cruzan en el mismo valor, como en el origen.
- Un paquete `.gaanim` guarda bien las escenas 3D. Antes, en cada fotograma salvo el primero de cada bloque, todas las piezas de una malla se dibujaban como la última (un cubo quedaba en una cara, una esfera en un triángulo) y las rejillas de unos ejes 3D en una sola línea, al reproducir, presentar o exportar el paquete. Hay que volver a grabar los paquetes con 3D.
- El reproductor ligero (`reproductor/insertar/`) abre en el reproductor completo las escenas con cristal; antes las dibujaba sin el efecto. Al pasarle una escena le da la dirección completa del archivo, que antes se perdía en un sitio bajo una ruta base, como GitHub Pages.
- Los temas `solarized-light` y `catppuccin-latte` pasan su propio `validate()`. El título de `solarized-light` (`#B08500`) y el color atenuado de `catppuccin-latte` (Overlay2, `#7C7F93`) son algo más oscuros y llegan al contraste de 3:1 sobre el fondo; antes se quedaban en 2,98:1 y 2,83:1.
- `scene.camera.animate.orthographic()` desde una cámara en perspectiva dura lo que pide `scene.play` (un segundo por defecto): el ojo se aleja por su eje de visión mientras el campo de visión se cierra, el objetivo sigue centrado y la proyección pasa a ortográfica al terminar. Antes cambiaba de golpe y no avanzaba el cursor. Desde una cámara ortográfica anima el zoom.

- Los docstrings de `Brush.solid`, `linear`, `radial` y `sweep` explican sus parámetros: las coordenadas están en unidades locales del objeto, y un grupo pasa el pincel a cada miembro, así que un gradiente no cruza el grupo. La referencia de los pinceles lo explica también, con cómo hacer que un gradiente recorra varias formas.
- El nombre del lenguaje de `scene.text.code` va en monoespaciada dentro de la cabecera, y el código empieza bajo la regla, desde el margen izquierdo del panel. Antes el nombre, con la letra del cuerpo, se salía por la esquina, y el código se centraba en la mitad izquierda: un fragmento ancho se salía del panel.
- Un relleno o un trazo con degradado se interpola durante el morph hacia el del destino (posición y paradas) y sigue siendo un degradado al terminar; dos degradados de tipos distintos pasan por un color liso a mitad. Antes se pintaba del color de la primera parada desde el inicio del morph. Vale para `transform_to`, `replacement_transform_to`, `magic_move` y `transform_matching_shapes`.
- `transform_to` de un SVG importado, de una de sus partes `<g>` o de un grupo se transforma pieza a pieza en lugar de desaparecer: las partes con el mismo nombre o `id` de SVG se emparejan, a cualquier profundidad, y las demás hojas se emparejan por forma dentro de su pareja. Las hojas que sobran del objeto se desvanecen y las del destino aparecen donde está el destino. El objeto conserva su identidad, así que sus `part()` siguen valiendo después, y el destino sigue siendo una plantilla oculta.
- Un contorno se empareja con el de destino en el sentido de recorrido que menos mueve sus puntos. Una forma y su espejo, escritos en sentidos opuestos, se reducían a una línea a mitad del morph; ahora conservan su área, y el resultado ya no depende de cómo se escribió el camino.
- `transform_to` deja libre la opacidad del objeto, como indicaba la referencia: un `animate.opacity` del mismo objeto en el mismo `play` lo funde mientras se transforma. Antes el `play` fallaba por usar los dos el canal `opacity`, y los mensajes de error se remitían el uno al otro. El error de dos animaciones que se pisan ya solo propone combinarlas en una cuando las dos son de propiedades.
- `PostProcess.lut`, `PostProcess.shader`, `Background.shader`, `Transition.shader`, la imagen de `Transition.preset("luma")` y las fuentes de `Theme(font_dir=..., font_files=...)` buscan una ruta relativa como los demás archivos de una escena: en el directorio de trabajo, luego en la carpeta `assets` del script y luego junto a él. Antes solo la buscaban en el directorio de trabajo, así que un script lanzado desde otra carpeta no encontraba su `.cube`, su `.wgsl`, su imagen o sus fuentes.
- `gaanim --diff` captura por defecto a 960 píxeles en el lado largo, o a la altura de la baseline que ya existe, y guarda los fotogramas opacos en RGB con la máxima compresión de PNG: una baseline pesa unas cinco veces menos que a 1920 en RGBA. Las baselines que ya tenías se siguen comparando a su tamaño; `--height` elige otro.
- `create`, `write` y los demás trazos que se dibujan de a poco ya no dejan franjas, barras ni gotas en sus primeros fotogramas. Una curva hecha de muchos trozos, como `implicit` o `contour`, se llenaba de franjas horizontales al empezar a dibujarse, y un `plot` mostraba un rectángulo en su origen: al recortar el trazo quedaban segmentos tan cortos que el trazador de Vello los convertía en esas manchas. Ahora los trozos visibles de menos de una diezmilésima de unidad se omiten.
- `scene.slides.bullets` alinea cada punto a la izquierda, justo después de su viñeta. Antes cada texto se centraba en la columna, así que los puntos cortos quedaban separados de su viñeta y desalineados entre sí.
- La vista previa gira la cámara como las exportaciones. Con `rotate_to`, `bind_2d(rotation=...)` o `follow_path(orient=True)` giraba la escena al revés, el `hud()` giraba el doble en vez de quedar fijo y, con la cámara lejos del origen, su centro se desplazaba de la pantalla hasta dejar la curva seguida fuera de vista. Las exportaciones y `--diff` no cambian.
- Las etiquetas `billboard` de una cámara ortográfica girada quedan derechas también al exportar; antes solo lo estaban en la vista previa equivocada.

- `camera.frame_to` y `camera.animate.frame_to` sin `margin` dejan media unidad por lado. Antes usaban 40 unidades, un valor de cuando el margen se medía en píxeles, y la vista se alejaba hasta un zoom de 0,1.
- Al terminar, `camera.animate.follow` y `frame_to(..., dynamic=True)` dejan la cámara donde la llevaron, y el siguiente movimiento parte de ahí. Antes volvía de golpe a la pose de antes de la toma. La pose final sale de dónde acaba el objetivo al final del `play`; si lo mueve otra cosa en tiempo de ejecución, como un updater, la cámara salta a esa pose al terminar.
- `move_to`, `align_to`, `next_to(aligned_edge=...)`, `to_corner`, `anchor_point` y `with_port` aceptan el nombre de un anclaje (`anchor="top_left"`), como ya hacía `place`; antes lanzaban `TypeError`.
- `CameraView.follow` y `Drawable.follow` sin `delay` empiezan en el cursor, y `no_camera_view()` quita la vista desde el cursor. Antes un `follow` llamado después de `view.animate.pan_to(...)` se aplicaba desde la creación de la vista y anulaba ese movimiento, y `no_camera_view()` al final de un segmento quitaba la vista en todo el segmento.
- `camera.animate.to(state)` entre una pose 2D y una en perspectiva, y `camera.animate.reset()` desde una en perspectiva, se mueven en perspectiva desde o hacia la vista equivalente a la 2D, y solo pasan a ortográfica en su extremo 2D. Antes la cámara salía del plano de la escena, así que las primitivas 3D desaparecían la primera mitad del movimiento, y un `reset` pasaba a ortográfica de golpe con la cámara aún inclinada, de modo que el contenido 2D se salía de la vista.
- `animate.flash()` dispara rayos alrededor del objeto (`lines`, `length`, `gap`, `color`, `width`) y `animate.circumscribe()` dibuja un marco a su alrededor y lo borra, como `flash_around(time_width=1.0)`. Antes el primero fundía y agrandaba el objeto y el segundo solo lo agrandaba un 10 %.
- `with_pen_tip()` dibuja una punta en el extremo de cada trazo mientras `write`, `create` o `draw_border_then_fill` lo trazan; antes no hacía nada.
- `grow_from_edge`, `grow_from_point`, `spin_in_from_nothing`, `indicate`, `wiggle`, `blink`, `broadcast` y `scene.spotlight` respetan su lugar en `sequence`, `stagger` o con `delay`; antes empezaban todos a la vez.
- `animate.indicate()` crece 1,2 veces y tiñe el objeto del `"accent"` del tema, de su `"foreground"` si el objeto ya tiene el color del acento, o de `color=`; antes solo crecía un 10 %. `animate.wiggle()` acepta `amplitude` y `count`, y sacude 0,1 unidades a cada lado en lugar de 0,05.
- En una selección de texto, `pulse()` late dos veces, `wave()` levanta los glifos uno tras otro y `highlight()` los tiñe del acento sin moverlos; `indicate()` crece y tiñe. Antes los cuatro eran casi el mismo salto.
- `magic_move(...)` sin `duration` y los movimientos de `scene.camera.animate` toman la `duration` de `scene.play` cuando no fijan la suya; antes duraban siempre un segundo.
- `clear_drive()` deja el objeto girado como lo tenía su `look_at`; antes volvía a su rotación propia.
- `scene.geometry.cross()` y `right_angle()` toman el estilo de línea del tema y se ven sin `stroke`; antes eran invisibles con el tema por defecto.
- Los trazos discontinuos de las formas cerradas conservan sus huecos con remates `"round"` o `"square"` (el trazo interior por defecto los agrandaba), y un guion de longitud cero con esos remates dibuja un punto.
- `stroke(paint, width)` después de `stroke_style(...)` conserva los guiones, remates y uniones, y `StrokeStyle` acepta `align`.
- El estilo raíz de un texto subraya, tacha y contornea todo el texto, como ya hacía en una parte, y `word_spacing` separa las palabras en la raíz. Un `stroke` sin `stroke_width` contornea con 0,02. Una parte con `baseline` sube o baja dentro de su línea; antes saltaba a otra o desaparecía.
- `max_lines` limita las líneas: `overflow="clip"` corta el texto tras la última palabra que cabe y `"ellipsis"` lo termina con «…». Antes no recortaba nada y, en una caja, ponía una palabra por línea.

- Un `wrap` numérico ajusta el texto dentro de una caja; antes lo dejaba en una sola línea.
- `text.tracking(...)` espacia bien los glifos que siguen a una coma.
- `text["parte"].fill(...)` después de un `play` cambia el color de la parte en el cursor, como una selección literal; antes no hacía nada.
- `role="code"` muestra * y _ tal cual (antes `"stagger(*grows)"` lanzaba `ValueError`).
- `write(order="random", seed=n)` cambia el orden con la semilla, y `write()` sobre un grupo, como un bloque de `scene.text.code`, escribe cada texto en orden de lectura en lugar de empezar por su última línea. Ya no deja recuadros alrededor de las etiquetas `content(...)` de CeTZ.
- `text.become(...)` después de un `play` cambia el texto en el cursor, y `animate.fade_transform_to(otro_texto)` deja visible el texto de destino.
- Los trazos negros de los documentos Typst (tablas, `rect`, `line`, CeTZ) toman el color del texto del documento y se ven sobre fondos oscuros.
- Los documentos Typst escriben su texto en el color del rol `body` del tema, como `scene.text`, en lugar de blanco o negro puro, y empiezan con `theme`, un diccionario con los colores del tema (`theme.accent`, `theme.rule`…) para dárselos a sus paquetes, que piensan en papel blanco y eligen negro y blanco por su cuenta. Ver Colores del tema en Typst.
- `write()` traza de un tirón las curvas que un paquete de Typst dibuja muestra a muestra (simple--plot, CeTZ); antes se veían punteadas.
- Las ediciones de una matriz (`insert_row`, `remove_column`, `swap_rows`, `reorder_columns`, `become`…) la dejan donde estaba, con su escala y sus animaciones; antes la reconstruían en el origen.
- `matrix.morph_to(destino)` termina en el destino: las entradas viajan hasta las suyas, los delimitadores se transforman en los del destino y aparecen sus entradas sin pareja. Antes se reordenaban dentro de la matriz de origen y el destino se quedaba con los delimitadores vacíos.
- `matrix.place(zona)` devuelve la matriz, no la caja de su layout.
- `order="simultaneous"` empieza todas las entradas a la vez aunque haya `stagger`; antes las escalonaba por filas. Sin `order`, un `stagger` las sigue escalonando fila a fila.
- `matrix.rank()` dibuja `rank = 2`; antes Typst no conocía `rank` y no se veía nada.
- Escalar una matriz, o cualquier caja, agranda también lo que contiene; antes solo separaba las entradas. Los delimitadores automáticos abarcan las filas según el tamaño de `entry_style`, y un `delimiter_size` explícito sigue al número de filas tras `become()`.
- Una ecuación con matemática inválida, como `a dot.circle b`, lanza `ValueError` al crearla, como dice la documentación; antes solo escribía el error de Typst y no dibujaba nada.
- Las filas de una ecuación partida con `\` ya no se pisan: guardan la separación de Typst entre filas.
- Las selecciones de partes encuentran sus glifos: `eq["inertia"]` con partes anidadas (antes no seleccionaba nada al animarla, y `highlight()` no se veía) y una parte cuyo texto aparece antes en la ecuación (antes seleccionaba la primera aparición). El color de `part(..., color=...)` tiñe solo la parte, no cada aparición de su texto.
- `write(by="part")` escribe las partes una tras otra; antes empezaban casi a la vez.
- `reveal("wipe")` deja el término oculto hasta su turno, como `"fade"`; antes se veía mientras se escribía el resto y luego se barría.
- `item(basis=...)` acepta `"N%"`, `"hug"` y `"fill"`, como dice la documentación; antes solo aceptaba longitudes.
- `fit="cover"` recorta la capa donde está su caja aunque la caja se haya movido con `move_to` o `place`; antes recortaba en el sitio donde se resolvió y la capa desaparecía.
- `box.detach(hijo)` en una caja movida deja al hijo donde estaba; antes se dibujaba en el sitio de la caja sin mover, desde el principio de la escena. Cambiar de padre ocurre en la línea de tiempo: el hijo sigue a la caja hasta el `detach` y desde ahí se mueve solo.

- Pasar un hijo de una caja a otra (`a.detach(hijo)` y luego `b.add(hijo, duration=...)`) lo desliza desde donde está; antes no se veía desde el principio de la escena hasta entrar en la segunda caja, y su posición anterior se leía respecto a ella.
- `point_on_curve`, `tangent_on_curve`, `normal_on_curve` y `curvature_on_curve` siguen la curva tal como se dibuja: movida, girada, escalada, animada o dentro de unos ejes. Antes usaban los puntos declarados y se quedaban atrás.
- `curvature_on_curve` dibuja el círculo también en los extremos de una curva abierta y en la costura de una cerrada; antes no dibujaba nada, o un círculo de radio 1 en el origen.
- `bind_rotation_from` ve el giro de `Updater.rotate`, y `bind_translation_from_rotation` sigue a `Updater.rotate` y a `rotate_to(Parameter)` más allá de media vuelta; antes la cremallera retrocedía al pasar de media vuelta.
- `fill` colorea entera una flecha sólida: su contorno sigue al relleno salvo que `stroke` fije otro. Antes `always_redraw_arc(...).fill(color)` solo coloreaba el interior de la punta.
- Los trackers de curvas, `always_redraw_arc` y `point_between(alpha=...)` aceptan un `Computed`.
- El ejemplo de `Parameter.add_updater_fn` en la referencia calcula el valor a partir del tiempo transcurrido; el anterior volvía a 0 al hacer seek.
- `transform_matching(..., mode="tex")` y `mode="text"` terminan mostrando el destino; antes dejaban ocultos el origen y el destino. Un `mode` desconocido lanza `ValueError` en vez de usar `"shapes"`.
- `transform_matching_shapes` y `transform_matching` muestran las piezas nuevas del destino, que antes no aparecían, y al final muestran el destino y ocultan el origen: las animaciones siguientes del destino mueven lo que se ve.
- `with_pivot(x, y)` en una parte de un SVG (`svg.part(...)`) lee el punto en coordenadas de escena, también en `animate.rotate_by`, aunque el SVG esté escalado o movido; antes lo leía en el marco del documento SVG.
- `Transition.morph` y `Transition.magic_move` con un resorte (`Easing.BOUNCY`, `Easing.SNAPPY`…) dejan que el movimiento se pase del destino y vuelva, y no parpadean: antes el sobrepaso mostraba el estado final de golpe y el rebote volvía a mostrar el contenido saliente. Con un easing explícito, el morph ya no lo suaviza otra vez. `cross_fade` y `fade_through` tampoco vuelven a mostrar el segmento saliente.
- `fx.spotlight([a, b])` abre un hueco alrededor de cada objetivo; antes abría uno solo alrededor de la caja conjunta.
- `drawable.post_process(False)` deja un objeto, con sus miembros, fuera del postprocesado de la escena y del de su segmento: se dibuja encima del cuadro procesado, así que un HUD o una etiqueta se leen bajo `pixelate`, `glitch` o `halftone`. Funciona en la vista previa, al exportar y en los `.gaanim`; un reproductor anterior dibuja esos objetos procesados.
- `segment.motion_blur(...)` da a un segmento su propio desenfoque de movimiento, o ninguno con `None`, en lugar del de `scene.canvas.motion_blur`.
- El mensaje de un `.cube` que `PostProcess.lut` no puede leer habla de una LUT, no de un «WGSL asset».
- Los ejemplos de `PostProcess.glitch` y de los pulsos con `Parameter` en la referencia y en la guía de efectos usan `.repeat(2, yoyo=True)`, que sube y vuelve; `.repeat(1, yoyo=True)` solo subía.
- La referencia de `scene.persist` explica que un objeto nuevo persistido justo después de `scene.segment(...)` entra con la transición de ese segmento.
- `text.animate.move_to(x, y)` deja el texto donde lo pone `text.move_to(x, y)`: la primera línea base centrada en el punto (el centro visual si tiene varias líneas). Antes centraba la caja visual, así que un texto colocado con `move_to` y luego animado a otro punto bajaba además de desplazarse.

- `scene.media.image(ruta, width=…, height=…, fit="cover")` y su equivalente en `video` recortan el sobrante por igual a ambos lados, como `.frame(…, fit="cover")`; antes recortaban desde la esquina superior izquierda.
- Una cuadrícula con columnas `"auto"` dentro de otra caja mide lo mismo que sola. Antes sus columnas se repartían el ancho por igual: la cuadrícula quedaba más ancha y, con textos que pueden partirse, el más ancho se partía en dos líneas.
- `box.reflow()` hace sitio a un hijo escalado o girado desde que se colocó, como dice la documentación; antes lo medía con su tamaño original y se solapaba con sus hermanos. `scale_to`, `rotate_to` y los demás ajustes de forma inmediatos funcionan sobre un hijo de una caja; antes lanzaban `LayoutOwnershipError`. Una caja anidada escalada ocupa el sitio de su tamaño escalado, como cualquier otro hijo; antes ocupaba el de su tamaño sin escalar.
- `obj.animate.place(zona, fit="stretch")` estira el objeto a la zona; antes lo escalaba como `"contain"`.
- El botón de descarga de la portada baja el paquete más reciente que exista para tu sistema y, si GitHub no responde, abre la página de la última release. Antes podía apuntar al de una versión aún sin publicar y dar un
  404. Las releases se publican cuando ya tienen sus paquetes.
- Las restricciones entre posiciones ya no encogen objetos: dos relaciones en conflicto sobre los bordes de una caja, como `(sq.center_x == caja.left).weak()` y `(sq.center_x == caja.right).strong()`, dejan la caja de su tamaño; antes la reducían a cero. `(texto.width <= 0.5).weak()` ya no aplasta los glifos.
- `group.animate.fill(color)` y `group.animate.stroke(color, ancho)` animan a todos los miembros del grupo, también en grupos anidados, como ya hacían `group.fill()` y `group.stroke()`; antes no cambiaban nada.
- `glow`, `blur`, `shadow` y `no_effects` después del primer `play` o `wait` son cortes en el cursor, como `fill`; antes se ignoraban y el objeto seguía con los efectos de su declaración. `blur(0)` quita el desenfoque y sirve de inicio nítido para `animate.blur`; antes lanzaba `ValueError`.
- `L.column`, `L.row`, `L.grid` y `L.stack` mantienen su dirección aunque un `style=` o una clase traigan otro `direction`.
- El stub de `Zone.arrange` incluye `direction="wrap"`, que ya funcionaba.
- Exportar a MP4, WebM, GIF, WebP o MOV crea la carpeta de salida si no existe, como ya hacían la secuencia PNG y el paquete. Antes fallaba con «FFmpeg stopped accepting frames»: por ejemplo al exportar desde el editor un script suelto, cuya carpeta por defecto es `exports/` junto al script.
- `blend("hue")`, `"saturation"`, `"color"` y `"luminosity"` ya no pintan columnas negras sobre un degradado gris: ciertos niveles de gris salían negros en toda la caja del objeto mezclado, también fuera de él, en la vista previa, al exportar y en los reproductores web.
- `box.set(color=…)` recolorea los textos que la caja creó a partir de cadenas, como al crearla, y con `duration` lo anima; antes solo valía para las cadenas añadidas después. Un texto de una caja recoloreado (con `set` o con `fill`) conserva el color cuando la caja cambia de ancho y lo vuelve a partir; antes recuperaba el de su declaración.
- `box.replace(viejo, nuevo)` y `box.remove(viejo)` en una caja con fondo dejan ver al hijo viejo hasta el cambio; antes, si se creó antes que sus hermanos, el fondo lo tapaba desde el principio de la escena.
- Un texto oculto con `opacity(0)`, o que salió con un fundido, sigue oculto al colocarlo en una caja; antes la caja lo mostraba opaco hasta que se animaba su opacidad.
- `box.reveal()` y `box.cascade(…)` hacen aparecer un `rolling_number` o un `progress_ring` de la caja como cualquier otra pieza; antes pasaban por el `animate` de su valor y el contador se veía desde el principio.

- `scene.geometry.traced_path(objeto)` de un objeto simulado con `add_updater_fn(…, fixed_dt=…)` que además tiene animaciones propias, como un `fade_in`, sigue la simulación al buscar un instante (en el editor y en las capturas). Antes el objeto volvía a su posición inicial en la mayoría de los cuadros y el rastro acababa con una recta hacia ella; había que trazar un punto con `follow(objeto)`.
- Un `traced_path` con `dissipating_time` ya no deja en algunos cuadros un bloque relleno que parte de su cola: cuando la cola estaba a punto de alcanzar su siguiente muestra, el primer tramo daba media vuelta y el trazo quedaba abierto. Los dos extremos de un rastro siguen su recorrido; antes podían acabar en un pequeño gancho.
- `race.bar(nombre).animate.opacity(…)` atenúa una fila de una carrera de barras, y el desvanecimiento de las filas que salen del top sigue funcionando encima. Antes lanzaba un error sobre un `Parameter` inaccesible.
- `chart.layer(nombre)` con una capa que el gráfico no tiene (`grid`, `guides` o `labels`) dice por qué falta y qué capas tiene. Antes respondía con la lista de nombres válidos, que incluía el pedido.

52.5.12. Al actualizar

- Un `hidden()` dentro de un grupo, o en una marca de un plano, ya no aparece con la entrada del grupo o el `create` del plano. Si contabas con ello, anima el objeto (`fade_in`, `create`…) o quita el `hidden()`.
- Una órbita de cámara con `up=(0, 0, 1)` rodea la escena en vez de pasar por encima; si buscabas ese giro, usa `look_at` con el ojo que quieras.
- Los planos con el eje X abajo muestran el cero del eje Y.
- `scene.camera.animate.orthographic()` ya no es instantáneo: dura un segundo, o la `duration` de `scene.play`, y avanza el cursor. Para el salto de antes, usa `scene.camera.orthographic()` o `.duration(0)`.
- El título de `solarized-light` y el color `muted` de `catppuccin-latte` son un poco más oscuros.
- `text.animate.move_to(x, y)` sin `anchor` lleva el texto a su línea base, como `text.move_to`. Un texto que se animaba a un punto pensando en su centro queda más arriba; pasa `anchor="center"` para conservar la posición anterior.
- `image(…, width=…, height=…, fit="cover")` y `video(…, fit="cover")` muestran el centro de la imagen en lugar de su esquina superior izquierda. Para encuadrar otra parte, usa `crop`.
- Las formas con degradado que se transforman, o que entran o salen en un `magic_move`, se ven con su degradado durante y después del morph en lugar del color de su primera parada. Si contabas con ese color liso, dale un `fill` liso al objeto y al destino.
- Si transformabas un SVG hoja a hoja (con un `id` en cada camino) para esquivar el fallo, sigue funcionando; ahora también puedes transformar el SVG o el grupo entero.
- Los fotogramas intermedios de un morph entre contornos de sentidos opuestos cambian (los extremos son los mismos). Si normalizabas el sentido de los contornos para evitar el colapso, ya no hace falta.
- Los bloques de `scene.text.code` salen coloreados y con el código alineado arriba a la izquierda. Para recuperar el aspecto monocromo, pasa `highlight=False`; si colocabas otros objetos respecto al código, revisa su posición.
- El texto con `role="code"` y los bloques de código cambian de fuente a JetBrains Mono, un poco más ancha que Consolas: revisa lo que alineabas con ellos. Para volver a la anterior, nómbrala en el tema, como `Theme(fonts={"code": "Consolas"})`, o en `font=` del bloque.
- En el editor, un WebM exportado desde un paquete abierto o desde un script fuera de un proyecto ya no conserva el canal alfa por defecto, como ocurre con `gaanim export`: marca **Conservar la transparencia** si la escena tiene un fondo transparente.

- `indicate()` crece más (1,2 en lugar de 1,1) y tiñe el objeto: para el salto de antes, usa `indicate(color=..., scale_factor=1.1)` con el color del propio objeto. `wiggle()` sacude el doble; `wiggle(amplitude=0.05)` recupera la sacudida anterior.
- Las escenas que usaban `flash()`, `circumscribe()` o los énfasis de texto `pulse`, `wave` y `highlight` se ven distintas: revisa sus tomas.
- Un `play(..., duration=...)` que incluye `magic_move` sin `duration` o movimientos de cámara sin `.duration()` ahora los alarga o acorta a esa duración. Fija `.duration(1)` si contabas con el segundo de antes.
- Los efectos que antes ignoraban su lugar en un `sequence` o `stagger` ahora esperan su turno, así que la escena puede durar más.
- `fill_level(direction=...)` nombra siempre hacia dónde crece el relleno: `"right"` crece desde el borde izquierdo y `"left"` desde el derecho, como `"up"` crece desde abajo. Antes `"left"` y `"right"` nombraban el borde de partida: cámbialos entre sí.
- `cross()` y `right_angle()` se ven con el trazo del tema; si les dabas un `stroke`, no cambia nada.
- Con `role="code"`, un `\$` muestra la barra invertida y el dólar, como el resto del código; antes mostraba solo el dólar. Quita la barra si la escribías para escapar el dólar.
- Un `word_spacing` en la raíz, que antes no tenía efecto, ahora separa las palabras: quítalo si no lo querías.
- Los textos con `max_lines` se cortan; con `overflow="visible"` se ven todas las líneas.
- `write()` sobre un grupo con textos (ejes, matrices, bloques de código) escribe cada texto desde su primer carácter: los fotogramas intermedios cambian, el resultado final no.
- `write()` y `unwrite()` sin duración duran 2 s en textos y grupos de 15 piezas o más, así que la escena puede alargarse: fija `.duration(1)` si contabas con el segundo de antes. Sus fotogramas intermedios cambian (la mitad del tiempo de cada pieza traza el contorno y la otra mitad lo rellena), y un objeto con trazo lo dibuja fino y luego lo engrosa.
- Los documentos de `scene.text.typst` usan el color de texto del tema en lugar de blanco o negro puro. Si querías blanco, ponlo en el documento con `#set text(fill: white)`. Un documento que definía su propio `theme` sigue usando el suyo.
- Una caja escalada o girada escala y gira también a sus hijos. Si la escalabas para separarlos, usa `gap` o `row_gap`/`column_gap`.
- `write()` agrupado con `by="word"`, `"line"` o `"part"` escribe hasta cuatro grupos uno tras otro y más con solape, así que sus fotogramas intermedios cambian; `stagger=0.2` recupera el solape anterior.
- Las ecuaciones con varias filas ocupan más alto; `line_spacing` por debajo de 1 las vuelve a juntar.
- `scene.text.equation` lanza `ValueError` con matemática inválida: corrige la fuente que antes no se dibujaba.
- `matrix.morph_to(destino)` muestra el destino y oculta el origen al terminar; antes el origen tomaba el aspecto del destino y el destino seguía oculto.
- Un objeto conserva su ancho y su alto frente a las restricciones medias y débiles: si una `.weak()` o `.medium()` tenía que redimensionarlo, ahora aparece en `check_layout()` sin cumplirse. Hazla `.strong()` para que lo redimensione.
- `shadow=True` sobre un fondo oscuro dibuja una sombra casi opaca en lugar de la suave que no se veía. Para la de antes, pasa `shadow={"color": "#00000055"}`.
- `box.reflow()` mide a los hijos con su escala actual: si escalabas un hijo con `.animate` y luego llamabas a `reflow()`, sus hermanos se apartan. Una caja escalada antes de meterla en otra también les hace sitio.
- Un `L.column(..., style=estilo)` cuyo estilo trae `direction="row"` vuelve a ser una columna: pasa `direction="row"` en la llamada o usa `L.row`.

- Si construías una curva ya en su sitio para que `point_on_curve` y los demás trackers coincidieran con ella, sigue funcionando; ahora también puedes moverla después.
- Una flecha con `fill(color)` y sin `stroke` tiene ahora el contorno del mismo color; antes era el del tema. Si querías el contorno del tema, fíjalo con `stroke(...)`.
- Para iluminar la caja conjunta de varios objetivos con `fx.spotlight`, pásale un grupo: `fx.spotlight(scene.geometry.group([a, b]))`.
- Si compensabas `with_pivot` en las partes de un SVG pasando coordenadas del documento, pasa ahora el punto de la escena. La fórmula de la guía Personajes ya lo hacía.
- `Transition.morph(..., easing=Easing.SMOOTH)` se ve algo más rápido al principio y al final: el easing ya no se aplica dos veces.
- Después de `transform_matching_shapes`, el origen queda oculto y el destino visible: anima el destino para seguir moviendo lo que se ve.

52.6. 0.11.0

Publicada el 10 de octubre de 2026. Gaanim llega al navegador: el playground ejecuta escenas de Python sin instalar nada, y el reproductor web se inserta en un blog o un sitio web como una figura más. Trae capas de parallax con profundidad de campo, texto que sigue una trayectoria, notaciones a mano alzada, movimientos de cámara (seguir una ruta, barridos y dolly zoom) y extrusión 3D, y el editor suma un inspector, una consola, un panel de animaciones, capturas de fotogramas y secuencias PNG. Lee «Al actualizar» si alguna escena mueve objetos por una trayectoria.

52.6.1. Playground y reproductor web

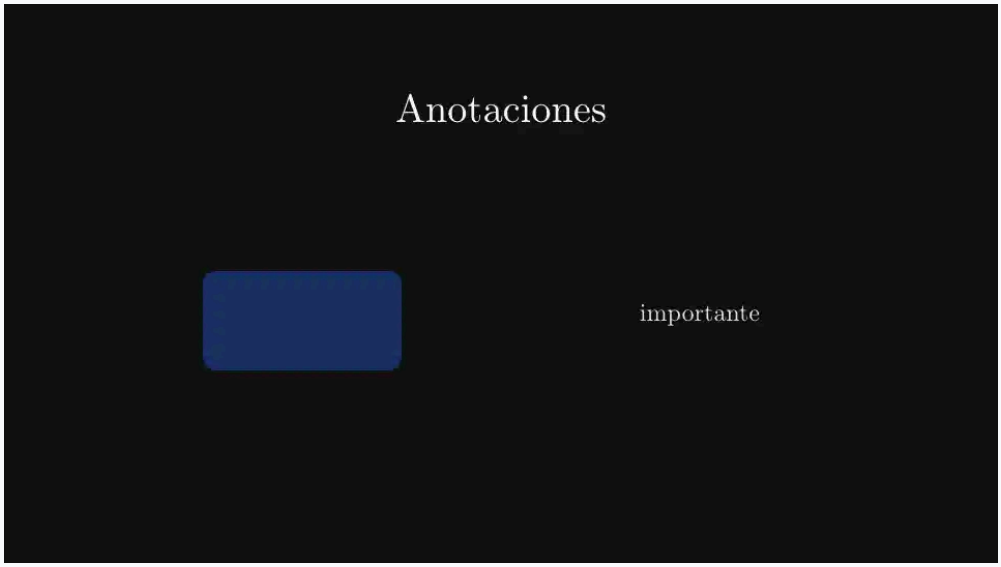
- El playground ejecuta escenas en el navegador. Escribes Python en un editor con autocompletado, firmas y la documentación de la API al pasar el cursor, y la escena se graba en un `.gaanim` que el reproductor web muestra al lado; un error marca su línea, también dentro de un módulo. Cada trabajo es un proyecto, como en un editor de código: el explorador crea, sube (archivos o carpetas enteras), renombra, mueve y elimina archivos, `main.py` importa los módulos de sus carpetas, y las imágenes, la música y las fuentes que subes (`Theme(font_dir="assets/fonts", …)`) se quedan en tu navegador. Hay varios proyectos: se crean vacíos o desde un ejemplo de la galería, se duplican y se abren desde un `.zip`, también la carpeta de un proyecto de la aplicación. Los paquetes de PyPI se declaran en `pyproject.toml` (la lista de `uv add`) o en `requirements.txt` y se instalan con micropip. `Ctrl+S` guarda y ejecuta, **En vivo** ejecuta al dejar de escribir, la grabación muestra su avance y su tiempo, y **Detener** cambia a un Python que ya se estaba cargando. **Compartir** copia un enlace con los archivos de texto, y **Descargar** convierte la grabación en vídeo MP4 o WebM, con su audio, o en GIF, en el navegador, o guarda el `.gaanim` o el proyecto en un `.zip`. Tiene tema claro y oscuro, el mismo que esta documentación. Usa Pyodide (Python 3.14 en WebAssembly); lo que necesita FFmpeg, como los vídeos como recurso, solo funciona en la aplicación. Ver El playground.
- **Insertar**, en el playground, prepara la escena para un blog o un sitio web: descarga el `.gaanim` con el nombre que elijas, compone el `<iframe>` con la dirección donde lo publicarás y la proporción de la escena, y muestra cómo se verá.
- Un reproductor ligero, `reproductor/insertar/?src=`, pone una animación en otra página: descarga unos 3 MB en lugar de los 24 del reproductor completo, empieza cuando está a la vista y se pausa al salir de ella, para que quien lee la entrada vea la animación al llegar a ella. `loop` la repite, `controls=0` oculta su barra fina (un clic pausa o reanuda) y `autoplay=0` espera un clic. Se detiene en cada `scene.stop()` hasta un clic y repite el bucle de la pausa (`loop=`) mientras espera, como al presentar. Una escena con posprocesado o con fondos, transiciones o efectos de shader se abre en el reproductor completo. El playground también lo usa. Ver Insertar en un blog o sitio web.

52.6.2. Motion design

- **Capas de parallax.** `scene.layer(depth)` devuelve una `ParallaxLayer`, y lo que le añades con `.add(...)` se mueve con `1/depth` del paneo de la cámara y se acerca con `zoom ** (1/depth)`: el fondo lejano apenas se mueve y el primer plano se mueve más. Cada capa se ordena por su profundidad; `depth=math.inf` queda fija como un HUD. Ver Capas de parallax.
- **Profundidad de campo.** `scene.camera.depth_of_field(focus, aperture=..., max_blur=...)` desenfoca cada capa según lo lejos que esté de la distancia de enfoque, también en exportaciones, capturas y paquetes; lo que no está en una capa, y los HUD, nunca se desenfoca. `scene.camera.animate.focus_to` cambia el foco de una capa a otra (rack focus) y puede ir a la vez que un movimiento de cámara. Ver Profundidad de campo.
- **Texto en una trayectoria.** `scene.text.on_path(texto, ruta, align=..., orient=..., reverse=..., offset=...)` coloca las letras a lo largo de un trazado, y `text.path_offset(...)` o `animate.path_offset(...)` las hacen correr por él. Ver Texto sobre una trayectoria.
- **Notaciones a mano alzada.** `objeto.annotate` (y `seleccion.annotate` sobre parte de un texto) subraya, encierra en una caja o un círculo, tacha, cruza o pone corchetes con un trazo irregular, como dibujado a mano. Cada marca sigue a su objeto y se dibuja con `animate.create()`. Ver Anotaciones a mano alzada.
- **Movimientos de cámara.** `scene.camera.animate.follow_path(ruta, orient=...)` lleva la cámara por un trazado, `whip_pan(destino)` es un barrido rápido que en exportaciones y capturas lleva desenfoque de movimiento, y `dolly_zoom(factor)` acerca la cámara en perspectiva mientras el sujeto conserva su tamaño. Ver Cámara.
- **Extrusión 3D.** `scene.geometry.extrude(objeto, depth, bevel=...)` convierte una figura, un texto, una ecuación, un grupo o un SVG en un sólido 3D. Ver Geometría.
- `Axonometric` se importa desde `gaanim`; en 0.10.2 solo estaba en `gaanim.gaanim_core`.

En la web, estas animaciones se reproducen en el reproductor ligero, el mismo que se inserta en un blog: pasa el cursor por encima para ver su barra.

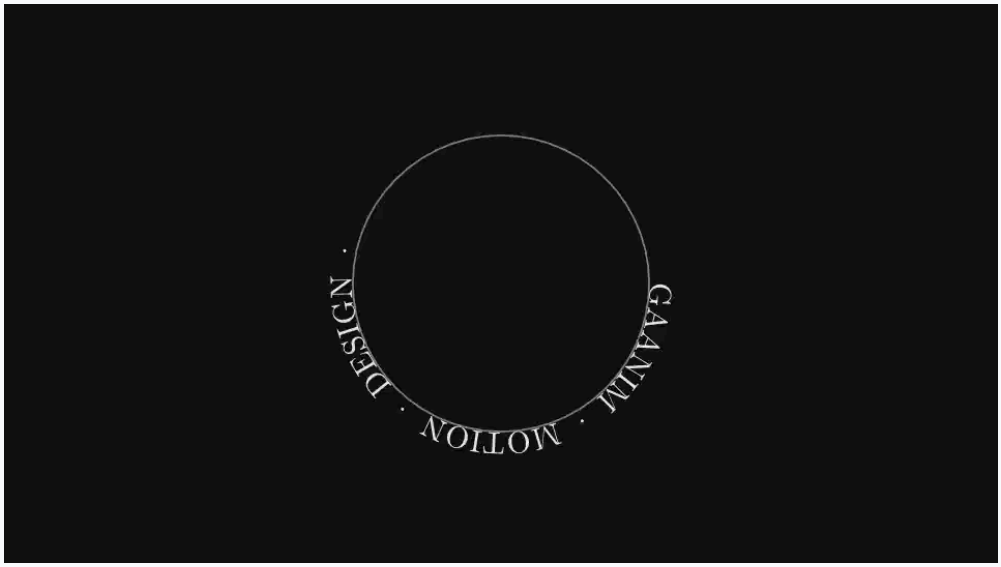
```
from gaanim import *

scene = Scene(frame=(16, 9))
title = scene.text("Anotaciones", role="title").move_to(0, 2.6)
card = scene.geometry.rounded_rect(3.2, 1.6, 0.2).fill(BLUE).move_to(-3.2, -0.6)
word = scene.text("importante").move_to(3.2, -0.6)
ul = title.annotate.underline(color=RED, seed=1)
box = card.annotate.box(color=GOLD, seed=2)
circle = word.annotate.circle(color=RED, seed=3)
scene.play([ul.animate.create(), box.animate.create()], duration=1.2)
scene.play([circle.animate.create()], duration=0.9)
scene.render()
```




```
import math
from gaanim import *

scene = Scene(frame=(16, 9), background="#1d2b53")
sky = scene.layer(depth=math.inf).add(scene.geometry.circle(0.9).fill(GOLD).move_to(4.5, 2.6))
far = scene.layer(depth=3.0)
for i in range(6):
    far.add(scene.geometry.polygon([(-2.4, 0), (0, 2.6), (2.4, 0)]).fill("#3b4a7a").move_to(-8 + i *
4.2, -1.4))
for i in range(8):
    scene.geometry.rect(0.5, 1.2).fill("#8a5a3b").move_to(-6 + i * 3, -2.0)
near = scene.layer(depth=0.6)
for i in range(5):
    near.add(scene.geometry.circle(0.9).fill("#1f4d33").move_to(-7 + i * 5, -3.3))
scene.play([scene.camera.animate.pan_to(6, 0).duration(3)])
scene.render()
```

```
from gaanim import *

scene = Scene(frame=(16, 9))
circle = scene.geometry.circle(2.4).no_fill().stroke(GRAY, 0.03)
```

```
ring = scene.text.on_path("GAANIM · MOTION · DESIGN · ", circle, reverse=True, size=0.5)
scene.play([ring.animate.path_offset(1.0).duration(4).easing(Easing.LINEAR)])
scene.render()
```



52.6.3. Editor

- **Inspector.** Un clic en la vista previa selecciona un objeto, y el inspector muestra su tipo y su nombre, la línea del script que lo creó (con **Abrir** y **Copiar**, y las líneas que la llamaron), su posición, rotación, escala, tamaño, relleno, trazo y opacidad, su capa y su profundidad de parallax. **Dentro de…** sube al grupo que lo contiene. Sin selección muestra la cámara, profundidad de campo incluida.
- **Consola** (`J`): avisos de Gaanim y de Python, lo que imprime `print`, los errores con su traza y las recargas, cada fila con un enlace a su línea, con filtros por gravedad y texto. Mientras está oculta, una marca arriba a la izquierda cuenta los problemas.
- **Animaciones** (`T`): una línea de tiempo sobre la barra de reproducción, con una fila por objeto animado y la cámara y un bloque por cada `play`. Un clic en un bloque va a su inicio y selecciona el objeto; un doble clic abre la línea del `play`.
- `Ctrl+Mayús+S` (o el botón de captura) guarda el fotograma actual como PNG, sin overlays, en `<output_dir>/captures/`.
- El diálogo de exportación graba secuencias PNG, nombra el archivo como el proyecto (o el script, si no es `main.py`), elige la carpeta con el selector del sistema y avisa antes de reemplazar un archivo.
- **Abrir** usa el editor de `GAANIM_EDITOR` (con `{file}` y `{line}`) o, sin esa variable, el primero de `code`, `cursor`, `codium`, `zed` o `subl` que encuentre. Ver CLI.

52.6.4. Correcciones

- Seguir una trayectoria (`move_along`, `path_arc`, rotar alrededor de un pivote externo) usa la longitud de arco exacta: el objeto ya no da saltos de hasta 0,15 unidades ni retrocede, y con `orient=True` no gira 180° un instante. Con orientación es de dos a cuatro veces más rápido.

52.6.5. Al actualizar

- Los objetos que siguen una trayectoria pueden quedar unos píxeles más allá o más acá que en 0.10.2 donde el muestreo anterior saltaba, y su orientación final usa la tangente exacta. No hay forma de volver al muestreo anterior, que era un error; si una composición dependía de esa posición, ajusta la fracción de la trayectoria.

- En el editor, un clic en un texto selecciona el texto entero, no un glifo, y entre objetos con el mismo `z_index` gana el más pequeño.

52.7. 0.10.2

Publicada el 6 de octubre de 2026. Recargar y abrir presentaciones largas es más rápido: medir con `bounds()` reutiliza lo ya compilado, un `traced_path` ya no encarece los saltos, guardar sin cambios conserva la escena, cambiar un asset recompila solo lo que lo usa, el script arranca mientras se prepara la ventana y `GAANIM_RELOAD_PROFILE=1` dice en qué se va el tiempo. Trae capas de dibujo con nombre, un aviso de textos tapados en `gaanim check` y dibujos axonométricos. Corrige un fallo de Typst en sesiones largas, el ancho de los trazos de contornos cerrados, las recargas tras guardar con herramientas que escriben un temporal y tres detalles de la API. Lee «Al actualizar» si alguna escena traza contornos cerrados o entra con `fade_in` a un objeto con opacidad declarada.

52.7.1. Cambios

- Las mediciones con `bounds()` (y `Anim.target_bounds`, `scatter` y `avoid_boxes`) comparten una compilación mientras no escribas nada entre ellas, y los miembros de un mismo layout recién creado se miden con una sola. Junto con el cambio siguiente, el Python de una recarga de la presentación de prueba (43 segmentos) bajó de 3,5 a 1,8 s.
- Un `traced_path` ya no obliga a cada salto en la línea de tiempo a recorrer la escena entera desde t=0, ni desactiva los puntos de control de los segmentos: en esa presentación, capturar sus 95 pausas pasó de casi tres minutos a 13 s. El rastro sale idéntico.
- Si guardas sin cambiar nada de lo que se compila (por ejemplo, solo un comentario), la recarga conserva la escena y sus puntos de control, y el aviso dice `(unchanged, N segments)`.
- Las recargas ya no recompilan los módulos de Python del proyecto que no cambiaron.
- El aviso de recarga incluye `seek`: lo que tardó volver a la posición donde estabas, que antes no contaba en el total.
- `GAANIM_RELOAD_PROFILE=1` escribe en la terminal qué costó cada recarga (el script por segmento, las mediciones y las líneas que compilaron la escena, las fases de la recompilación) y el arranque, fase por fase. Sin esa variable, Gaanim avisa en la terminal y en `gaanim check` cuando medir con `bounds()` compiló la escena a un coste notable, con las líneas del script responsables.
- La vista previa arranca en Vulkan (Metal en macOS) cuando la GPU lo admite, sin iniciar también DX12: en Windows llega al primer fotograma unos 0,3 s antes y sin las esperas de varios segundos que a veces causaba DX12.
- Al abrir un script, este empieza a ejecutarse mientras se preparan la ventana y la GPU, y encontrar su entorno de Python ya no arranca procesos: se leen el `pyvenv.cfg` y los metadatos del paquete instalado. En la presentación de prueba, la primera escena llega unos 1,3 s antes.
- Guardar un asset (una imagen, un Lottie, un archivo de datos) ya no vacía todas las cachés ni recompila la escena entera: se olvida solo lo que las cachés guardaban de ese archivo, y la recarga sigue siendo incremental desde el primer segmento que lo usa. Si la escena no lo lee, la recarga queda en `unchanged`. El aviso de recarga dice qué archivo cambió.
- Editar una diapositiva recompila con menos coste fijo: las huellas que deciden qué segmentos se reutilizan se calculan en paralelo, las transiciones entre segmentos y las capas procedurales ya no recorren todos los clips una vez cada una, y el punto de control se copia mientras se restaura la escena. Al editar la última diapositiva de la presentación de prueba, el replay pasó de 0,45 a 0,25 s (las huellas, de 100 a 17 ms). Un cambio inmediato sobre un objeto de un segmento anterior ya no impide reutilizar el segmento que lo declaró.
- Una medición con `bounds()` que compila la escena reanuda la compilación de la anterior desde el segmento donde cambió la escena, y la siguiente ejecución del script reanuda la última, salvo

desde un segmento con funciones de Python (reactivas, updaters), que son nuevas en cada ejecución. En la presentación de prueba, la segunda medición de cada recarga pasó de unos 0,3 a 0,16 s.
- Capas de dibujo con nombre: `scene.z_layers("modelo", "overlay")` y `drawable.z_layer("overlay")` dejan lo que va encima (ventanas, lupas, etiquetas) sobre cualquier `z_index` del resto, que pasa a ordenar solo dentro de cada capa. Sin capas, todo se dibuja como antes.
- `gaanim check` avisa cuando una figura opaca dibujada encima tapa por completo un texto en una pausa o al final de un segmento, y dice cuál, detrás de qué y dónde. `covers_on_purpose()` marca una figura, o un grupo, que tapa textos a propósito, como un diálogo sobre el modelo, para que no avise ni haga fallar `--strict`.
- Dibujos axonométricos: `scene.geometry.axonometric("isometric", ...)` dibuja puntos de un modelo 3D (con `z` hacia arriba) como polígonos y polilíneas planos, con el mismo estilo que el resto de la escena. `depth_sort` ordena las caras como un pintor y `animate_to` pasa de una vista a otra (de planta a isométrica, por ejemplo) moviendo los vértices. Ver Geometría.

52.7.2. Correcciones

- Compilar Typst en una sesión larga ya no falla con «out of file ids». Cada compilación gastaba uno de los 65 000 identificadores de archivo de Typst; al agotarse, las miniaturas del presentador dejaban de generarse y fallaba toda compilación Typst posterior.
- `bounds()` justo después de un cambio inmediato sobre un objeto ya animado, como un `move_to`, mide la posición nueva. Según cómo sumaran las duraciones de la escena, medía un instante antes del cambio.
- `with_pivot` sobre un objeto ya mostrado cambia el pivote en el cursor sin moverlo, y las rotaciones, escalas y sesgos siguientes lo usan. Antes se ignoraba sin aviso.
- `points()` y `animate.points()` aceptan una polilínea con `closed=True`, que sigue cerrada. Antes lanzaban `ValueError`.
- `fade_in()` termina en la opacidad declarada o fijada del objeto, o en la que tenía antes de un `fade_out`, en lugar de 1. También las entradas por trazo (`create`, `write`) tras un `fade_out`.
- `stroke(color, ancho)` en un contorno cerrado, con el trazo hacia dentro (el valor por defecto), dibuja el ancho entero dentro de la figura, como `"outside"` lo dibuja entero fuera. Antes se veía la mitad: el trazo centrado, recortado por el contorno.
- Guardar un `.py` escribiendo un temporal y renombrándolo sobre el archivo (la herramienta Edit de Claude Code, `sed -i`) o restaurarlo con `git checkout` recarga como un cambio de código, con el replay incremental. Antes contaba como un cambio de asset: vaciaba las cachés y recompilaba la escena entera. Ni los temporales de guardado ni la carpeta del archivo, que Windows también da por cambiada, cuentan ya como cambios.

52.7.3. Al actualizar

- Los trazos de contornos cerrados hacia dentro (sin `align` o con `align="inside"`), incluidos los de los glifos, se ven con el ancho declarado: el doble que en 0.10.1. Para conservar el aspecto anterior, divide entre dos su ancho. Los trazos centrados, hacia fuera o de caminos abiertos no cambian.
- Un objeto declarado con una opacidad menor que 1 que entra con `fade_in` (o con `create` o `write` tras un `fade_out`) termina en esa opacidad, no en 1. Si contabas con llegar a 1, quita la opacidad de la declaración.

52.8. 0.10.1

Publicada el 4 de octubre de 2026. Corrige una caída de fotogramas al mostrar la barra de reproducción en presentaciones largas abiertas desde su script.

52.8.1. Correcciones

- La barra de reproducción ya no frena el editor en presentaciones con muchas diapositivas. Calculaba los capítulos recorriendo todos los clips una vez por escena en cada fotograma, y la vista previa bajaba a unos 30 fps mientras la barra estaba visible. Los paquetes `.gaanim` no se veían afectados.

52.9. 0.10.0

Publicada el 4 de octubre de 2026. Rendimiento de la vista previa y de la exportación: exportar es entre un 13 y un 49 % más rápido, saltar en la línea de tiempo cuesta entre un 7 y un 35 % menos, el editor descansa cuando no hay nada que cambiar y GIF, WebP, WebM y secuencias PNG se exportan más rápido. Lee «Al actualizar»: los GIF y WebP salen a 30 fps como mucho y las exportaciones respetan los fps que pidas.

52.9.1. Cambios

- La exportación prepara el fotograma siguiente mientras la GPU dibuja el actual, y los fotogramas que repiten el anterior (un `wait()`, una pausa) no se vuelven a dibujar.
- La exportación y las capturas reutilizan el dibujo de cada objeto que no cambió; los efectos por objeto independientes se dibujan con un solo render, y las mallas 3D con sombreado suave ya no recalculan sus degradados en cada fotograma.
- Las mallas 3D se proyectan y se preparan en varios núcleos: la escena 3D del benchmark exporta en la mitad de tiempo que en la 0.9 y se reproduce en el editor a 144 fps en lugar de 90.
- Los ecos de un grupo se evalúan una vez por nivel de eco en lugar de una vez por copia, y los efectos por objeto que cambian de tamaño ya no recompilan sus pases en cada fotograma.
- El motion blur prepara cada subfotograma mientras la GPU dibuja el anterior y los promedia en varios núcleos: exportar con motion blur tarda la mitad.
- Las sombras difuminadas, los brillos y el desenfoque se difuminan en la GPU una sola vez por objeto: un texto con sombra que se mueve exporta cinco veces más rápido.
- Los MP4 y WebM pasan a YUV en la GPU, que envía a FFmpeg menos de la mitad de bytes por fotograma.
- Los morph de una figura, un texto o una fórmula a otra emparejan sus contornos una vez por animación en lugar de en cada fotograma.
- Una escena sin segmentos que encadena cientos de animaciones salta desde el último `play` en el que todas terminaron, en lugar de repetirlas desde el principio.
- Las capturas de `--diff` codifican los PNG en varios hilos mientras se dibujan los fotogramas siguientes.
- Exportar arranca entre 0,3 y 0,5 s antes en Windows, porque la GPU se busca primero en Vulkan.
- Saltar en la línea de tiempo reutiliza lo ya dibujado: la línea de tiempo ya no reescribe valores que no cambian.
- Con la vista previa en pausa o en el Inicio, el editor espera a la siguiente entrada en lugar de recalcular la escena en cada refresco.
- WebM usa varios núcleos y se exporta un tercio más rápido; las secuencias PNG se codifican en varios hilos y el borrador de MP4 usa `veryfast`.

52.9.2. Correcciones

- Los GIF y WebP en calidad estándar o de producción salían a 60 fps y se veían a cámara lenta en los navegadores. Ahora salen a 30 fps.
- `scene.export(..., fps=...)` y los ajustes del diálogo de exportación ya no se sustituyen por los del preset.
- Las imágenes de los efectos por objeto que dejan de dibujarse se liberan.

- Un FFmpeg que escribe mucho en su salida de error ya no puede bloquear la exportación.
- Las escenas 3D con más de unos 8000 triángulos visibles con sombreado suave ya no fallan al exportar.
- La recarga en caliente ya no vigila `.venv`, `.git` ni `target`.
- Un contador animado con Typst ya no hace crecer la memoria sin fin.
- Al arrastrar la barra de tiempo, el vídeo ya no parpadea con fotogramas anteriores.
- `Updater.bob`, `Updater.pulse` y `Updater.rotate` con un número dependen solo del tiempo: ya no varían con la cadencia de fotogramas, oscilan alrededor de la posición animada del objeto en lugar de la que tenía al añadirlos y se combinan en un mismo objeto (antes el último sustituía a los anteriores).

52.9.3. Al actualizar

- Los GIF y WebP salen a 30 fps como mucho en cualquier calidad.
- Los fps que pases a `scene.export` se respetan; antes se usaban los del preset.
- Los MP4 con `libx264` usan `-tune animation`, y los WebM salen algo más grandes con la misma CRF.
- Las sombras difuminadas, los brillos y el desenfoque son un difuminado gaussiano real: sus bordes cambian levemente y un objeto desenfocado muestra su color exacto.
- Las aristas de las mallas 3D opacas se sellan con un trazo centrado: la silueta sobresale medio píxel más que antes.
- Un objeto con `Updater.pulse` y `Updater.rotate` ahora late y gira; antes solo giraba. `remove_updater()` devuelve un objeto con `Updater.bob` o `Updater.pulse` a su posición y escala.

52.10. 0.9.0

Publicada el 3 de octubre de 2026. Transiciones por shader: cada segmento se dibuja por separado y la GPU los mezcla con una función WGSL, como en gl-transitions. Llegan cinco listas para usar y la posibilidad de escribir las tuyas. Y el audio anima la escena: el volumen, las bandas de frecuencia y los golpes de una pista mueven, giran y tiñen cualquier objeto. Cualquier objeto puede ser de vidrio, esmerilado o líquido como el *Liquid Glass* de iOS, pasar por sus propios shaders, mostrarse a través de un mate o fundirse como gotas con los metaballs. Además, los SVG con transformaciones conservan sus degradados, recortes y trazos, los grupos se funden como una capa y las sombras de texto cuestan una por objeto. Lee «Al actualizar»: los paquetes `.gaanim` de 0.8 deben grabarse de nuevo y el interlineado por defecto es más compacto.

52.10.1. Cambios

- `Transition.preset(nombre, duración, **ajustes)` aplica una transición incorporada: `cross_zoom`, `directional_warp`, `ripple`, `glitch_displace` o `luma`, que revela el segmento entrante siguiendo una imagen en escala de grises (`image=`). Lee Transiciones.
- `Transition.shader(wgsl, duración, uniforms={...})` acepta tu propia función `transition(uv)`, que lee `gaanim_from(uv)`, `gaanim_to(uv)` y `progress`.
- Cada segmento se dibuja con su fondo; los objetos persistentes y los overlays quedan nítidos por encima de la mezcla. Funcionan en la vista previa, la exportación, los paquetes y el reproductor web, y un seek cae en el mismo fotograma que la reproducción.
- Una pista de audio da señales de 0 a 1 que animan cualquier cosa que acepte un número, y siguen a la pista donde suena: `level()` (el volumen), `band(low, high)` (una banda de frecuencias) y `pulse(decay)` (los golpes). `beats()` da los instantes de los golpes y `duration`, lo que dura la pista. `pulse` y `beats` aceptan una banda (`low=40, high=120` sigue el bombo).

- `Audio.tempo()` estima el BPM y la fase del primer pulso para `scene.tempo`, y `Audio.analysis()` da el análisis como listas de Python para colocar cortes a mano. `Audio.level(range_db=24)` mide el volumen en decibelios.
- `Updater.rotate`, `Updater.wiggle` y el nuevo `Falloff.source` aceptan una señal del tiempo, así que el audio también gira, hace temblar y tiñe con `drive("fill", ...)`. Lee Audio.
- `Audio.spectrum(bandas, normalize=)`, `Audio.waveform(puntos, span=)` y `Audio.samples(puntos, span=)` (la onda con signo, para un osciloscopio) dan una señal por banda o por instante reciente, y `scene.viz.equalizer` las dibuja con la forma (`"bar"`, `"capsule"`, `"dot"`, `"line"`, `"area"` o la tuya), la disposición (`"row"`, `"mirror"`, `"radial"` o la tuya) y los colores que elijas. También sirven como uniforms de un shader, y `PostProcess.shader(..., audio=)` le da el espectrograma entero (`gaanim_audio_spectrum(x, ago)`). `Background.shader` y `Transition.shader` aceptan uniforms que siguen señales (y el fondo, `audio=`). Lee Audio.
- `scene.geometry.polyline` acepta señales en sus coordenadas y `closed=True`.
- `Drawable.shader_effect(efecto, margin=)` aplica pases de `PostProcess` (presets, tu WGSL o una cadena) a un solo objeto y sus hijos, en su sitio y con uniforms animados por `Parameter` o por el audio. Lee Drawable.
- Los paquetes `.gaanim` graban la tiza y los efectos de shader por objeto.
- `Drawable.liquid_glass()` convierte un objeto en vidrio líquido como el *Liquid Glass* de iOS: el centro deja ver lo que hay detrás sin deformarlo y el borde lo curva como una lente, lo desliza a lo largo del contorno (`twist`), lo separa en colores (aberración cromática, `dispersion`) y brilla con una línea fina de luz. Sus ajustes (`bevel`, `refraction`, `blur`, `transparency`…) son los de `Drawable.glass`, que hace vidrio esmerilado; `backdrop_blur` solo desenfoca lo que hay detrás. El borde sigue la forma real del objeto, así que unos metaballs de vidrio se funden como gotas. Lee Drawable.
- `Drawable.matte(fuente, mode=)` muestra un objeto solo a través de otro (alpha o luma, normales o invertidos), y `scene.geometry.metaballs` funde bolas como gotas. Lee Drawable y Geometría.
- `echo(..., start=, end=)` limita la estela a un intervalo de la escena.
- `drawable.copy()` duplica un objeto; en un SVG copia también sus partes con nombre y sus recortes (`copia.part("cabeza")`).
- `keyframes(offset=[...])` mueve el objeto relativo a donde está al empezar. `position` fija su origen, no su centro visual como `move_to`.
- Sin carpeta de recursos, las rutas relativas se buscan también en la carpeta `assets/` del script y en la del script.
- `GAANIM_FRAME_PROFILE=1` informa de caminos, segmentos y recortes.
- El fondo llena la vista aunque la cámara salga del cuadro de la escena.
- `readout` y `variable` aceptan `font` y `weight`.

52.10.2. Correcciones

- `uniforms=` de `Background.shader`, `PostProcess.shader` y `Transition.shader` acepta cualquier `Mapping`, y los stubs lo tipan así: un diccionario de señales de audio ya no da error en el comprobador de tipos.
- Los alias de tipo de los stubs (`Paint`, `ColorLike`, `Padding`…) existen también al ejecutar la escena: `from gaanim.gaanim_core import Paint` ya no lanza `ImportError`. `Paint`, `ColorLike`, `ColorMapLike` y `BackgroundLike` se importan también desde `gaanim`.
- Un `blur_in` con `.delay(...)` mantiene sus glifos transparentes durante el retardo; antes dejaba una trama tenue donde iba a entrar el texto.
- Un `Background.shader` sigue animándose y siguiendo el audio durante una transición por shader o con máscara; antes se congelaba hasta que acababa.
- Los degradados y los `clip-path` de un SVG bajo `<g transform>` o un `viewBox` desplazado siguen la transformación, y el trazo de una figura cerrada va centrado, como en SVG.

- Un grupo o un SVG que se funde se compone como una sola capa: lo que una pieza tapa sigue tapado.
- La sombra de un texto o un grupo se dibuja una vez, desde la silueta de sus miembros, en lugar de una por glifo.
- `echo` en reposo no apila sus copias sobre el objeto.
- `quantity(1600, "kg/m^3")` escribe «kg/m³» sin hueco antes de la barra.
- La vista previa ya no desborda los búferes de Vello con muchas curvas (decenas de fórmulas o tiza), que dejaban una mancha blanca.
- Con fuentes variables, `write()` y los trazos de texto ya no muestran los contornos que se solapan dentro de cada glifo.

52.10.3. Al actualizar

- Los paquetes `.gaanim` pasan a la versión 5 del formato: Gaanim 0.9 no abre los grabados con 0.8 ni anteriores. Vuelve a grabarlos con `gaanim export <proyecto> --output charla.gaanim`.
- `line_spacing` vale 1.0 por defecto en lugar de 1.2: las líneas quedan a un tamaño de letra, más cerca de como se veían en 0.7. Pasa `line_spacing=1.2` para el espaciado de 0.8.
- Los trazos de los SVG importados van centrados; usa `.stroke_align("inside")` si un dibujo dependía del trazo hacia dentro.
- La sombra de un grupo es una sola, bajo todos sus miembros.
- Un emisor de partículas con `opacity()` se funde como una sola capa: las partículas solapadas ya no suman su brillo.
- Con el mismo `z_index`, un objeto de `scene.persist(...)` queda encima del contenido de los segmentos, también al terminar una transición por shader.
- `format` en `readout`, `variable` y las cotas sigue la especificación de Python: `"05.2f"` rellena con ceros y un formato no admitido (`d`, `s`…) lanza `ValueError`.

52.11. 0.8.1

Publicada el 3 de octubre de 2026. Corrige lo que se vio al hacer los primeros videos con 0.8.0: sombras de curvas sin relleno, cotas demasiado gruesas, un `stagger` que cambiaba de orden y documentos Typst que se ensanchaban hasta la página. Llegan `gaanim export --frame`, contadores con ancho reservado sin ceros, trazos que escalan con su figura, updaters con velocidad o amplitud animable, escritura con tiza, arcos de elipse, magnitudes con unidades, reparto automático sin solapes y una guía de personajes por partes. Lee «Al actualizar» si usas `scene.text.typst` con párrafos largos.

52.11.1. Cambios

- `scene.geometry.double_arrow(..., body_width=…)` acepta el grosor del cuerpo, como `arrow`, para cotas finas. Para una cota completa con líneas de extensión, usa `scene.mechanics.dimension`. Lee Geometría.
- `gaanim export <escena> --output f.png --frame 11.4` escribe exactamente un PNG con lo que se ve en ese instante. Acepta varios instantes o marcadores separados por comas, que se escriben como `f_1.png`, `f_2.png`… Lee CLI.
- `scene.viz.rolling_number(..., pad=" ")` deja vacías las posiciones que reserva `min_digits` en lugar de dibujar ceros: el contador ocupa siempre lo mismo y el prefijo no se desplaza.
- `drawable.scale_stroke_with_object()` o `stroke(..., scale_with_object=True)` hacen que el ancho del trazo siga la escala de la figura y de sus grupos; en un grupo o un SVG se aplica a todos sus trazos.

- `Updater.rotate(velocidad)` y las amplitudes y la frecuencia de `Updater.wiggle` aceptan un `Parameter`: animarlo acelera un giro o calma un temblor sin saltos, y un seek cae en el mismo fotograma que la reproducción. Lee Movimiento.
- `write(brush="chalk", seed=…)` y `drawable.chalk(seed, roughness)` dibujan como con tiza: un contorno que tiembla un poco y un grano que rompe la pintura, iguales para la misma semilla. Los paquetes `.gaanim` aún no graban tiza. Lee Drawable.
- `scene.geometry.ellipse_arc(cx, cy, rx, ry, inicio, barrido)` dibuja arcos de elipse, como la base de un cilindro en perspectiva.
- `quantity(valor, unidad)` escribe magnitudes con unidades como `siunitx`: `quantity(181, "L")` y `quantity(1, "m^3")` dejan el mismo espacio fino. Lee Texto.
- `scene.layout.scatter(items, region, avoid=[...], gap=..., seed=...)` reparte objetos en una región sin solaparse entre sí ni con `avoid`, igual para la misma semilla. Lee Layout.
- La guía Personajes con SVG por partes anima un personaje dibujado en Inkscape con `part`, `with_pivot` y `look_at`.

52.11.2. Al actualizar

- Un documento de `scene.text.typst(...)` sin `width=` ya no reparte un párrafo largo en líneas al ancho de una página A4: corta línea solo donde lo indica el texto. Pasa `width="16cm"` (o el ancho que quieras) para que se ajuste, o para columnas `1fr`.
- Un `stagger` con `total` o `easing` pero sin `origin` sigue el orden de la lista. Si querías el orden por distancia, pasa `origin="start"`.

52.11.3. Correcciones

- `shadow()` sobre un camino abierto o sin relleno proyecta la sombra de su trazo; antes rellenaba la región entre el trazo y la cuerda que une sus extremos, también en un `group`. Los objetos con relleno conservan su sombra.
- `scene.text.typst(...)` ajusta la página a su contenido, así que un `#align(center)` centra dentro del contenido y `move_to` deja el documento centrado.
- `stagger(..., total=…, easing=…)` sin `origin` ni `grid` escalona en el orden de la lista, y `distribute` también.

52.12. 0.8.0

Publicada el 2 de octubre de 2026. El público entra a la presentación: desde su teléfono vota encuestas, responde cuestionarios con puntos y cuenta atrás, se une a un equipo y aparece en la escena como el personaje que armó. La escena muestra esos datos como tú decidas, y al terminar tienes los resultados en hojas de cálculo. Llegan también partículas, *magic move*, carreras de barras, seguimiento con inercia y sonidos anclados a animaciones. **El interlineado cambia**: lee «Al actualizar».

52.12.1. Público en vivo

- Los teléfonos se conectan a un relay que despliegas gratis en Cloudflare con `gaanim relay init` y configuras con `gaanim relay use <URL>`. Lee Público en vivo.
- `scene.poll(...)` abre una encuesta y `scene.quiz(...)` un cuestionario con respuesta correcta, tiempo y puntos, con selección múltiple (`multiple=True`) o una imagen sobre la pregunta. `load_questions(...)` lee las preguntas de un Markdown o un CSV.
- La API da datos, no diseños: `votes`, `percent`, `remaining`, `revealed`, la clasificación (`scene.leaderboard()`), los equipos (`scene.teams(...)`) y cada jugador (`scene.audience()`) son valores en vivo que mueven lo que quieras. Lee la referencia.
- `quiz.reveal()` revela la respuesta en los teléfonos cuando la presentación avanza; `quiz.revealed()` vale 0 hasta entonces y 1 después.

- `scene.stop(until=...)` avanza sola cuando el público cumple una condición.
- `scene.character(...)` y `scene.live_zone(audience, comportamiento)` dibujan a cada persona como su personaje; el comportamiento es una función de Python que Gaanim compila para que un `.gaanim` la ejecute sin Python. Lee Zonas vivas.
- `scene.rehearsal(...)` juega la escena con un público inventado en la vista previa y en las exportaciones.
- Cada partida se guarda en `resultados/` como `jugadores.csv`, `respuestas.csv` y `preguntas.csv`; `scene.roster("Código de alumno")` pide un dato más al unirse. La vista del presentador tiene **Save now** y **Open folder**, y `gaanim relay results` los descarga después. Lee Guardar los resultados.
- Ejemplos: `la_gran_trivia`, `quiz_orbit_demo`, `quiz_game_demo`, `audience_poll_demo` y `live_zones_demo`.

52.12.2. Motion design

- `scene.fx.particles(...)` y `scene.fx.confetti(...)`: partículas deterministas.
- `magic_move(antes, despues)` y `Transition.magic_move`: los objetos con el mismo nombre viajan a su nuevo estado.
- `scene.viz.bar_race(...)`: una carrera de barras desde fotogramas clave.
- `Anim.settle(...)` rebota al final de un movimiento y `Drawable.follow(..., delay=...)` sigue con retraso.
- `Anim.sound(...)` ancla un sonido a una animación y `scene.media.sfx(...)` lo coloca en un instante.
- `Anim.keyframes(...)`: varias paradas en un solo clip, con un easing por tramo y trayectoria suave opcional; también en `Parameter.animate`.
- `Anim.throw(...)` lanza un objeto que rebota hasta reposar y `Anim.inertia(...)` lo desliza hasta el punto de `snap` más cercano.
- `Composition.speed_ramp(...)` y `Composition.time_remap(...)`: cámara lenta y aceleraciones dentro de una composición.
- `Drawable.modifiers`: zigzag, esquinas redondeadas, pucker y bloat, torsión, wiggle y offset no destructivos, con parámetros animables.

52.12.3. Reproductor web

- Un enlace `?src=` empieza a reproducir antes de descargar el archivo entero: el reproductor lee el final del `.gaanim` y pide cada parte cuando hace falta, mientras el resto llega en segundo plano.
- El enlace puede apuntar a un momento (`#segmento=5`, `#pausa=12`, `#t=83.5`) y, con `#present`, ofrece el botón **Presentar**.
- El botón de enlace de la barra copia el enlace al archivo o al instante que se ve; el reproductor funciona dentro de un `<iframe>`. Lee Compartir sin Python.
- El reproductor web reproduce el audio del paquete y recibe votos en vivo al presentar un paquete con encuestas (lee En el reproductor web). Al presentar, el aviso del Presenter View, `F11` y el panel inferior ponen la presentación en pantalla completa.
- La barra muestra al pasar el cursor una miniatura del momento con el nombre y el número del segmento; un clic en el nombre del segmento actual abre la lista de segmentos con buscador. Tiene control de volumen cuando la escena tiene audio.
- **Actualiza el relay**: la versión 13 del protocolo permite presentar desde el navegador.

52.12.4. Ajustes

- `Drawable.hidden()` deja oculto un objeto creado a mitad de segmento hasta que una animación lo muestra.
- `dashed_line` acepta los mismos extremos que `line` y los sigue si se mueven.

- `Readout.move_to` y `Variable.move_to` aceptan `TextAnchor`; `spacing=` fija el espacio entre sus términos.
- Dentro de un layout, un readout reserva el ancho del valor más ancho que mostrará y alinea el número a la derecha, así que sus vecinos no se mueven mientras cuenta. `reserve=` añade un valor para conteos en vivo sin máximo conocido, como `votes`.
- `slides.brand(number_anchor=...)` pone el número de diapositiva aparte del pie.
- `gaanim check` dice el segmento, el instante y el texto de una caja que se sale de su contenedor.
- `gaanim --diff --height <PX>` captura a otra resolución.
- Las llaves (`geometry.brace` y `selection.animate.brace`) se dibujan como una llave tipográfica: brazos rectos, rizos del tamaño de su profundidad, una punta y un grosor que se afina hacia los extremos.

52.12.5. Al actualizar

- `line_spacing` es la distancia entre líneas base en múltiplos del tamaño de letra, como `line-height` en CSS. Con 1.2, las líneas quedaban a 0.9 veces el tamaño y ahora a 1.2: los textos de varias líneas ocupan más alto.
- Un readout colocado con `move_to` queda centrado justo en ese punto.
- En una caja que se ajusta a su contenido, un hijo que crece mide su contenido aunque tenga `basis`.
- `dashed_line(x1=…, y1=…)` con palabras clave deja de funcionar: pasa las coordenadas por posición.
- Una llave es una figura rellena sin trazo: coloréala con `fill`. Un `stroke` dibuja su contorno, y un tema la pinta como la familia `glyph`, no como `line`.
- `line`, `arc`, `traced_path` y `tracking_line` nacen sin relleno, como `polyline`: ya no hace falta `no_fill()`. Un arco o un trazo seguido que dejaba ver un relleno blanco entre sus extremos ahora solo dibuja su trazo; dale `fill()` si lo querías.
- Reemplaza todos los archivos de la carpeta de Gaanim: el soporte de Python cambió su interfaz.

52.12.6. Correcciones

- `Box.bounds()` mide el tamaño real de una caja dentro de un layout, y `surrounding_rect` se dibuja alrededor de cajas.
- Un texto que crece con `basis=0` se reparte en líneas en el espacio libre de su fila.
- Un readout dentro de un layout queda dentro de su celda, también cuando su número crece.
- Una caja dentro de otra dibuja su fondo aunque su único contenido sea un readout.
- `highlight` sobre una parte de un texto la agranda en su sitio; antes la desplazaba sobre sus vecinas.
- Una fila con `width="fill"` dentro de una columna que se ajusta a su contenido toma el ancho de la caja que la contiene, y una fila cuyo texto se reparte en líneas mide el alto de todas ellas.
- `Theme(fonts=...)` respeta la familia de `title` aunque también se defina `text`.
- Los fondos con shader y el postprocesado siguen moviéndose en una pausa, y las animaciones de entrada no se ven antes de tiempo al saltar entre segmentos.
- Las teclas de navegación entre pausas funcionan con el foco en un panel, y `P` fija la ventana encima.
- En el teléfono, el nombre del equipo se lee en blanco sobre su color; antes quedaba gris y casi no se veía.
- En teléfonos de menos de 360 px de ancho, la barra superior muestra el apodo y los puntos completos.
- Presenter View muestra en vivo lo que ve el público, con los votos y los jugadores de la sala; antes mostraba una miniatura con los del ensayo.

52.13. 0.7.2

Publicada el 29 de septiembre de 2026. Una caja se puede recorrer, buscar, animar en cascada y reordenar (`walk`, `find`, `cascade`, `stagger`, `swap`…), y un solo movimiento del cursor puede escalar, teñir, girar o desvanecer cientos de elementos con los nuevos falloffs. La reproducción ya no se vuelve más lenta a medida que avanza una presentación larga, y recargar una escena que mide cajas del layout es mucho más rápido. Las cajas que entran con `grow_*` ya no se ven antes de su turno y el overlay **Layout** del editor no mezcla segmentos. No hace falta cambiar tus escenas.

52.13.1. Cambios

- `box.walk()` devuelve las piezas de una caja a cualquier profundidad, en orden de dibujo, con el fondo de cada caja antes de su contenido. `box.walk(boxes=True)` incluye también las cajas anidadas.
- `box.stagger(make, each=…)` anima todas esas piezas de una vez como un `stagger`: `make` recibe cada pieza y devuelve su animación (o `None` para saltarla). Lee Layout.
- `box.find(type=…, where=…, text=…)` y `box.find_all(...)` buscan entre esas piezas; `text` compara con el contenido de un texto (`Text.content`).
- `box.each(make)` aplica un ajuste inmediato a cada pieza y devuelve la caja.
- `box.reveal()` revela una caja con la política que casi toda diapositiva pide: fondos y objetos con un fundido, textos que entran deslizándose y filetes (cajas sin hijos que dibujan algo) que crecen.
- `gaanim check` y `box.diagnostics()` avisan cuando el contenido de una caja se sale de ella o una caja se superpone con la anterior de su fila o columna. Un contenedor con recorte (`clip=True`) puede sacar contenido sin aviso.
- `box.cascade(each=…).fade_in()` (o cualquier otra animación) las anima todas en cascada, sin escribir la lambda de `stagger`.
- `box.move_child(hijo, a)`, `box.swap(a, b)` y `box.reverse()` reordenan los hijos, y con `duration=` se deslizan a su nuevo sitio. En un `stack` la posición no cambia qué hijo queda encima: usa `z_index`.
- `Falloff` calcula un valor por instancia a partir de su posición en el grupo (`Falloff.index`), su distancia a un objeto o un punto (`Falloff.distance`, `Falloff.linear`) o un ruido con semilla (`Falloff.noise`). Se reforma con `remap` e `invert` y se combina con `+`, `-`, `*`, `maximum` y `minimum`.
- `grupo.drive("scale", near.remap(1.0, 1.8))` conecta un falloff a un canal de cada miembro de un grupo: `scale`, `rotation`, `opacity`, `x`, `y` y, con `near.gradient(AZUL, ORO)`, `fill`. `grupo.look_at(cursor)` hace que cada miembro mire a un objeto que se mueve, y `clear_drive()` lo termina.
- Se evalúa en Rust por miembro y fotograma, sin llamar a Python: una rejilla de 20×12 con tres efectos cuesta alrededor de 0,1 ms por fotograma. Un seek cae en el mismo fotograma que la reproducción. Lee Falloffs.
- Énfasis nuevos: `animate.blink(3)` parpadea un objeto, `animate.broadcast()` lanza ondas concéntricas desde él, `animate.flash_around()` y `animate.flash_under()` pasan una ventana brillante por un marco o un subrayado (también sobre `texto["palabra"]`), `scene.fx.spotlight(objeto)` oscurece todo salvo un hueco alrededor del objetivo y `objeto.animated_boundary([AZUL, MORADO, CIAN])` dibuja un marco cuyo trazo cambia de color. Lee Animaciones.
- `box.children`, `box[i]` y la iteración de una caja están tipados como `Drawable` en lugar de `Any`.

52.13.2. Rendimiento

- Cada fotograma parte del inicio del segmento en que cae, en lugar de reconstruir la escena desde el principio: el coste de un fotograma ya no crece a lo largo de la presentación. En una presentación de 41 segmentos, la reproducción pasaba de 100 fps al principio a 55 fps al final y ahora se

mantiene en torno a 100 fps de principio a fin, y exportarla tarda un 30 % menos. El resultado es idéntico fotograma a fotograma; `GAANIM_SEGMENT_CHECKPOINTS=0` vuelve al comportamiento anterior para comparar. Lee CLI.
- Guardar el estado de la escena es unas cuatro veces más rápido. Lo hace al cargar y la primera vez que la reproducción entra en cada segmento, que en esa presentación cuesta un solo fotograma de unos 50 ms.
- `bounds()` sobre un objeto de una caja recién declarada mide solo su árbol de cajas en lugar de recompilar toda la escena: en la misma presentación, la parte de Python de una recarga bajó de 3,5 s a 1,4 s.
- Los fondos y bordes de las cajas ya no se redibujan en cada fotograma cuando no cambian.
- `GAANIM_FRAME_PROFILE=1` informa además de cuántos objetos están visibles.

52.13.3. Correcciones

- Una caja colocada por el layout, o un hijo suyo, ya no se ve entera antes de su turno cuando entra con `grow_from_center`, `grow_from_point`, `grow_from_edge` o `spin_in_from_nothing`.
- Un `geometry.connector` con extremos fijos que es hijo de una caja ya se coloca en su celda, como `geometry.arrow`, en lugar de dibujarse apilado con los demás.
- Una columna con hijos `.item(grow=1)` y altura automática ya cuenta el contenido de esos hijos: antes se medía casi sin altura y su contenido se desbordaba sobre la caja siguiente.
- Una caja, su fondo o sus hijos que entran con `grow_*` ya no aparecen antes de su turno cuando una caja se reorganiza después (`set`, `add`, `remove`, `swap`…).
- El overlay **Layout** del editor solo dibuja las cajas del segmento actual: los límites de los layouts de otros segmentos ya no quedan visibles.
- `just wheel` (y con él `just build-release-install`) borra las wheels de versiones anteriores antes de construir: ya no falla con «Expected one universal Gaanim wheel» tras subir de versión.
- `just build-release-install` ya no falla al borrar `gaanim-core` y `gaanim-play`, que dejaron de instalarse.

52.14. 0.7.1

Publicada el 29 de septiembre de 2026. Los hijos de una caja se animan con libertad: el layout decide dónde descansan, no cómo se mueven. Llegan `text_box` para calcar diseños al píxel, avisos de cajas vacías y un overlay **Layout** en el editor para inspeccionar cajas y zonas como en las herramientas de desarrollo de un navegador. Exportar desde el editor ya no bloquea la reproducción. No hace falta cambiar tus escenas.

52.14.1. Cambios

- Los hijos de una caja aceptan todas las animaciones de `.animate`: `fade_in_from`, `shift_by`, `move_to`, `scale_by`, `rotate_by`, `grow_from_edge`, `spin_in_from_nothing`… Parten del sitio que les da la caja y no empujan a sus hermanos, como un `transform` de CSS. Si la caja se reorganiza después, el hijo conserva su desplazamiento, su escala y su giro. Colocar un hijo de forma inmediata (`move_to` sin `.animate`) sigue lanzando `LayoutOwnershipError`. Lee Animar dentro de una caja.
- `text_box="cap"` o `"ink"` en un texto o en una caja cambia la caja con que el layout mide y coloca el texto: de la altura de las mayúsculas a la línea base, o ajustada a la tinta. Sirve para calcar diseños en los que las mayúsculas tocan el borde. Por defecto sigue siendo `"line"`.
- `scene.layout.check_layout()`, `box.diagnostics()` y `gaanim check` avisan de las cajas con fondo o borde que quedan con ancho o alto cero, como una barra vacía sin `width="fill"`, e indican su ruta (`column[2] > row[1]`).

- El editor tiene un overlay **Layout** (`0` y después `K`) que inspecciona las cajas como las herramientas de desarrollo de un navegador: contornos de cajas y zonas, y, bajo el cursor, padding, celdas, márgenes y gaps con sus medidas en px.
- Una exportación desde el editor sigue en segundo plano: el diálogo se cierra y puedes seguir reproduciendo. La barra de tiempo marca bajo cada segmento lo ya exportado, la barra de reproducción muestra el porcentaje y permite cancelar, y la terminal muestra la barra de progreso de `gaanim export`.

52.14.2. Correcciones

- El indicador de progreso de una exportación desde el editor ya no se queda en 0.
- `animate.move_to(x, y)` sobre un objeto dentro de otro usa coordenadas de la escena, no las de su contenedor.

52.15. 0.7.0

Publicada el 29 de septiembre de 2026. Llega un layout nuevo al estilo de CSS, construido sobre Taffy (el motor de flexbox y grid que usan otros proyectos de Rust): cajas con padding, gap, bordes, radio, sombra y recorte, estilos reutilizables, zonas y estructura animada, para componer escenas sin coordenadas (lee la guía de Layout). Además, Gaanim pasa a ser un solo ejecutable, que carga Python solo cuando abre un script, y Vello dibuja también el 3D con un estilo propio. Lee «Al actualizar»: el layout cambia de forma incompatible y desaparece la importación glTF.

52.15.1. Al actualizar

- `scene.layout.item(objeto, grow=1)` pasa a ser `objeto.item(grow=1)`, que devuelve el propio objeto.
- `scene.layout.card([...], background=..., padding=...)` pasa a ser `scene.layout.box(..., background=..., padding=...)`. Los *ports* de las tarjetas desaparecen: une conectores a los objetos.
- `row`, `column`, `grid` y `stack` reciben los hijos como argumentos (`row(a, b, c)`); una lista sigue funcionando porque se aplana. Una cadena se convierte en texto.
- `Layout` se llama `Box`; `configure(...)` es `set(...)` y `configure_item(...)` es `hijo.item(...)`. `add`, `remove` y `replace` aceptan `duration=` para animar el cambio.
- `layout_template` y `scene.layout.template` se sustituyen por `@component`: llama a la función directamente, `mi_plantilla(scene, ...)`.
- El texto dentro de una caja se mide por su caja de línea (ascendentes y descendentes de la fuente), así que textos del mismo estilo comparten altura y línea base. Las fórmulas `$…$` conservan su caja de tinta. Un texto puede desplazarse unos píxeles respecto a la versión anterior.
- Si una escena importa modelos glTF, quita las llamadas a `scene.media.gltf(...)` y a `animation(...)` de esos modelos.
- Reemplaza todos los archivos de la carpeta de Gaanim y borra `gaanim-core` y `gaanim-play`.

52.15.2. Cambios

- Unidades de diseño: `"24px"` son píxeles de un fotograma de 1080 de alto (`Scene(design_resolution=...)`), `"50%"` del padre, `"1fr"` en grids y tokens del tema como `"space_md"`.
- Propiedades de caja al estilo CSS: `padding`, `gap`, `margin` (con `"auto"`), `justify`, `align`, `align_self`, `grow`, `shrink`, `basis`, `wrap`, `border`, `radius="full"`, `shadow`, `clip` y tipografía heredada (`font_size`, `color`, `weight`…).
- `BoxStyle` y `scene.layout.classes(...)` para estilos reutilizables con `style=` y `class_=`.

- `Box.set(..., duration=...)` y `hijo.item(..., duration=...)` animan cualquier cambio de propiedades; `advance=False` encadena cambios en paralelo.
- Zonas: `Zones.rows`, `Zones.columns` y `Zones.grid` dividen el área segura, una zona o un objeto en regiones con nombre; `objeto.place(zona, ...)` y `objeto.animate.place(...)` colocan objetos que siguen siendo libres.
- Ejemplos nuevos: `layout_boxes`, `layout_flex`, `layout_grid`, `layout_reflow`, `layout_zones`, `ui_dashboard`, `ui_mobile_app` y `ui_lower_third`.
- Gaanim es un solo ejecutable, `gaanim`. Ya no existen `gaanim-core` ni `gaanim-play`. El motor está en una biblioteca, `gaanim_engine`, en lugar de copiarse en dos ejecutables, y el soporte de Python en otra, `gaanim_python_plugin`, que `gaanim` carga solo al ejecutar un script. Al actualizar, reemplaza todos los archivos de la carpeta y borra `gaanim-core` y `gaanim-play`. En Ubuntu cambia la instalación (ver Instalación).
- El inicio abre proyectos aunque instales Python con Gaanim ya abierto: el soporte de Python se carga al abrir el proyecto, no al arrancar.
- Vello dibuja el 3D: proyecta las mallas de triángulos y las líneas 3D con la cámara de la escena, las ordena de atrás hacia delante y calcula la luz en la CPU con un estilo propio de Gaanim: las primitivas con `Material3D` conservan su color, con luz del cielo y del suelo, una luz principal suave desde arriba a la izquierda y un brillo leve en los bordes; las caras planas tienen un solo color y las curvas, degradados suaves. Las superficies, los gráficos 3D y las líneas no reciben luz. Sin búfer de profundidad, la geometría que se cruza puede ordenarse mal, y el 3D queda debajo del 2D. El contenido 3D no proyecta sombras: `lighting_3d(shadows=...)` se acepta pero no tiene efecto (ver Cámara y 3D).
- Se elimina la importación glTF. `scene.media.gltf(...)` lanza `NotImplementedError`, `Drawable.animations()` devuelve una tupla vacía y `Drawable.animation(...)` lanza `NotImplementedError`. `scene.assets.preload` ya no acepta `.gltf` ni `.glb`.
- Los paquetes `.gaanim` graban el contenido 3D, y las vistas previas de Presenter View lo muestran. El desenfoque de movimiento, el postprocesado y los fondos (también los de shader y degradado) se aplican a las escenas 3D y con cámara en perspectiva.

52.15.3. Correcciones

- El reproductor web ya no se detiene al abrir un `.gaanim` con post-procesado o un fondo de shader: comprobar el shader esperaba a la GPU bloqueando el único hilo de la página. Ahora la comprobación se resuelve entre frames, y el efecto aparece un frame después.
- Un texto declarado con `opacity(0)` vuelve a verse con `animate.opacity(1)`: la opacidad ya no se copiaba también a cada glifo.
- `cancel()` y los demás efectos de selección encuentran una cadena entre comillas dentro de una ecuación, como `part("x", '"bloques vistos"')`.
- Un `z_index` puesto después de enlazar una propiedad reactiva, por ejemplo el nivel de `fill_level` con `computed`, ya no se ignora.
- `slides.brand` no dibuja la identidad en el primer segmento salvo con `show_on_cover=True`. El pie empieza en el borde izquierdo del área segura y se reduce si no cabe, y la regla ocupa el ancho del área segura.
- `checkmark` dibuja una marca rellena proporcional a su tamaño, en lugar de un disco blanco enorme, y un trazo explícito ya no muestra un rombo en la esquina interior.
- `create()` y `write()` vuelven a mostrar un objeto que un `fade_out()` anterior había dejado con opacidad 0.
- El editor vuelve a completar los parámetros de `scene.text(...)`, `scene.text.equation(...)`, `Text.become(...)` y `TextFlow(...)`: los tipos `TextRole`, `TextWrap`, `TextAlign`, `TextOverflow` y `TextDirection` vuelven a estar en los tipos del paquete.

- `scene.assets.load_project()` sin ruta busca el `gaanim.toml` más cercano subiendo desde la carpeta del módulo que la llama, así que funciona desde un paquete del proyecto (`capitulo4/estilo.py`) y no solo desde el script de entrada. Sin manifiesto, el error indica la carpeta de partida y cómo pasar la ruta (#266).
- Los métodos fluidos heredados de `Drawable` (`move_to`, `shift_by`, `fill`, `opacity`, `scale_by`, `next_to`…) devuelven el mismo objeto sobre el que se llaman, así que un `Readout`, una `Variable`, una `Dimension` o cualquier otro tipo compuesto conserva sus partes al encadenar: `scene.viz.readout(3.0).move_to(0, 0).number.fill("red").part()` sigue devolviendo un objeto nuevo (#267).
- `--sections` selecciona segmentos cuyo nombre tiene comas: pasa el nombre solo (`--sections "Tiempo, lugar y orientación"`) o escribe sus comas como `\,` dentro de una lista. El error de un nombre desconocido lista las opciones entre comillas y explica cómo escribir una coma (#270).

52.16. 0.6.2

Publicada el 28 de septiembre de 2026. El reproductor web abre el Presenter View, y los `.gaanim` se reproducen sin saltos de cámara ni tirones. No hace falta cambiar tus escenas.

52.16.1. Cambios

- El reproductor web abre el Presenter View: al presentar, abre en otra ventana el mismo Presenter View del escritorio, con la diapositiva actual y la siguiente, notas, cronómetro, vista general y dock. Las dos ventanas van a la par desde cualquiera de ellas: teclas, clics, el dock, la barra de reproducción y las pantallas en negro o en blanco. `P` lo vuelve a abrir. Ver Compartir sin Python.

52.16.2. Correcciones

- Al reproducir un `.gaanim`, la cámara ya no salta frame a frame a la posición del inicio: si la pantalla refrescaba más rápido que el paquete, cada refresco sin frame nuevo mostraba la cámara de t=0.
- Los `.gaanim` con mucha geometría por segundo, como líneas de corriente, se reproducen sin tirones: los frames se decodifican a medida que se muestran, no un segundo entero de golpe, así que la preview ya no baja de resolución en esos tramos.

52.17. 0.6.1

Publicada el 28 de septiembre de 2026. Llegan el reproductor web, la licencia MIT OR Apache-2.0, `scene.launch`, la rejilla de tempo y paquetes `.gaanim` más pequeños, con portada, icono propio y doble clic. Cambian el ancho de los trazos escalados, la opacidad tras `transform_to` y la colocación de un SVG cuyas partes mueves: lee «Al actualizar».

52.17.1. Al actualizar

- El ancho de `.stroke(color, ancho)` se mide siempre en unidades de escena, como promete su referencia: `scale_to`, `scale_to_3d`, `matrix_to`, las inclinaciones, la escala de un grupo y sus animaciones cambian la forma, no el pincel. Antes la escala también ensanchaba el trazo, y una escala distinta por eje lo deformaba: `rect(1, 1).stroke(c, 0.03).scale_to_3d(5.5, 2.8, 1)` dibujaba los lados verticales el doble de gruesos que los horizontales. Si una escena contaba con que la escala engrosara el trazo, escribe el ancho que quieres ver: `.stroke(c, 0.09)` para el trazo de antes con `scale_to(3)`. Una figura que crece desde cero muestra ya su trazo completo. Los trazos propios de un SVG importado siguen escalando con su `scale_to`, pero no con una animación de escala posterior. El ancho de un resplandor (`glow`) también queda en unidades de escena.

- `animate.transform_to(destino)` conserva la opacidad del objeto. Antes tomaba la del destino, así que un destino declarado con `.opacity(0)` dejaba invisible al objeto. Si usabas un grupo con `.opacity(0)` para esconder los destinos, ya no hace falta: se ocultan solos. Para cambiar la opacidad durante el morph, combínalo con `animate.opacity`. `replacement_transform_to` no cambia.
- Mover una parte de un SVG (`svg.part("g").shift_by(...)`) ya no mueve el SVG entero: las partes se colocan después de la raíz, así que `move_to`, la escala y el pivote de la raíz usan el dibujo tal como lo declara el archivo. Antes `move_to` centraba el SVG con la parte ya desplazada y todo el dibujo se movía para compensar; si lo corregías a mano, quita esa compensación.
- Los paquetes `.gaanim` usan el formato 2: sus datos van comprimidos con Zstandard. Una presentación larga ocupa menos de la mitad (la de 4½ minutos de Compartir sin Python baja de 42 a 18 MB) y un paquete pequeño, alrededor de un 10 % menos. El cambio rompe la compatibilidad en los dos sentidos: 0.6.1 ya no abre los paquetes grabados con 0.6.0 (vuelve a grabarlos desde su script con `gaanim export`), y 0.6.0 no abre los de 0.6.1, así que quien los reciba debe actualizar o usar el reproductor web.

52.17.2. Correcciones

- Acercar o alejar la vista con el modo interactivo al reproducir un `.gaanim`, en el escritorio o en el navegador, ya no deja copias de los fotogramas anteriores alrededor del marco.
- La ventana de un `.gaanim` se titula con el nombre del proyecto que lo grabó, no con el del script (`main`).
- Un tren de engranajes encadenado (`b.bind_rotation_from(a)`, `c.bind_rotation_from(b)`…) gira entero en la vista previa. Antes, desde la tercera rueda se quedaban quietas hasta hacer un seek o pausar, y con `Canvas.motion_blur` dejaban copias dobles: cada eslabón leía la rueda anterior con un fotograma de retraso.
- `traced_path` se reconstruye con su recorrido completo al saltar a otro instante, también al exportar un fragmento (`--from`), al capturar fotogramas sueltos y con `Canvas.motion_blur`, cuando la fuente se mueve con animaciones o sigue a un `Parameter` animado. Antes la estela salía vacía o desaparecía; `dissipating_time` fallaba igual.
- `Color(GOLD)` o `Color((r, g, b))` devuelven ese color, como cualquier parámetro que acepta uno. Los stubs tipan como `ColorLike` los parámetros de color que aceptan una cadena o una tupla (`glow`, `Overlay.flash`, `mechanics.gear`, `rolling_number`, `marker`…), así que un comprobador de tipos ya no marca como error código correcto.
- Una vista de cámara (`camera_view`) cuya pantalla o marco se transforma con `transform_to` en otra figura ajusta el aumento y el encuadre a la forma nueva. Antes el recorte seguía el contorno nuevo, pero el aumento y el encuadre se calculaban con el tamaño original, así que la vista salía estirada o miraba otra región.

52.17.3. Cambios

- Los archivos `.gaanim` se comportan como documentos del sistema: `gaanim register` (o el botón **Asociar archivos .gaanim** del inicio) hace que el doble clic los reproduzca, añade **Presentar** al menú contextual y muestra su portada en el Explorador de Windows y en Nautilus, Nemo, Caja y Thunar. `gaanim unregister` lo deshace; ninguno pide permisos de administrador. Los `.gaanim` tienen su propio icono: una hoja con un video animado vectorial y el logo de Gaanim, que ahora tiene las esquinas redondeadas. Ver Compartir sin Python.
- Cada `.gaanim` guarda una portada: el fotograma de la primera pausa o, sin pausas, el del primer segmento que más muestra. `scene.thumbnail()` elige otro, y `gaanim thumbnail archivo.gaanim portada.png` la extrae sin GPU. Un paquete sin portada se abre igual.
- La instalación en Ubuntu copia también `gaanim-play`, que reproduce los `.gaanim`.

- `scene.launch(...)`, o `scene.play(..., advance=False)`, empieza animaciones en el cursor sin moverlo: un giro que dura varios cortes sigue mientras programas lo demás. Animar el mismo canal antes de que termine lanza `ValueError`, y `gaanim check` avisa si la escena acaba antes que ella. `duration(0)` queda documentado como corte dentro de una composición.
- Rejilla de tempo: `scene.tempo(bpm, offset)`, `scene.beats(n)` y `scene.wait_until(beat=n)` o `wait_until(bar=n)` cortan a tiempo con la música, y avisan si un plano se pasó de su hueco. El editor dibuja una línea en cada compás de la barra de reproducción.
- `scene.media.audio(..., end=30.0, fade_out=1.5)` recorta y funde una música de fondo sin alargar el `play` que la activa.
- `echo(..., hold=True)` deja las copias congeladas cuando el objeto se detiene, como un papel cebolla, en lugar de que lo alcancen.
- `drawable.points([...])` fija los vértices de un polígono o una polilínea sin animarlos, igual que los demás setters.
- Con `role="code"`, `$` es literal: `scene.text("$ gaanim init", role="code")` ya no pide escribir `\$`.
- El ejemplo `examples/launch_tempo_demo.py` reúne `launch`, el tempo, el eco congelado, `points` y el `$` literal, con su referencia visual.
- Documentación: la guía de Efectos cubre postprocesos encadenados y animados, acabados listos, bloom, desenfoque de movimiento, ecos y fondos vivos; Movimiento añade repetidores, duplicadores y plexus. La referencia de animaciones lista qué animaciones son entradas y cuándo toca suelo `Easing.bounce`, y la de `hud()` explica cómo pasa por las transiciones.
- Reproductor web (experimental): abre un `.gaanim` en el navegador de la PC o del teléfono, sin instalar nada, con la misma barra de reproducción que el escritorio, en paololupo.github.io/gaanim/reproductor. Con `?src=` abre un archivo publicado en otro sitio. Todavía no reproduce el audio ni abre la ventana del Presenter View (ver Compartir sin Python). El sitio de la documentación lo enlaza desde la cabecera y la portada.
- Controles táctiles en el reproductor, al estilo de los reproductores de vídeo del móvil: un toque muestra u oculta los controles, con botones grandes para la pausa anterior, reproducir y la siguiente; dos toques en un lado, o deslizar, van a la pausa siguiente o anterior. También funcionan en el editor con una pantalla táctil.
- Gaanim tiene licencia: se distribuye bajo MIT OR Apache-2.0, a elección de quien lo usa. Los vídeos, imágenes, archivos `.gaanim` y scripts que creas con Gaanim son tuyos y no están sujetos a esa licencia. Las descargas incluyen `LICENSE-MIT`, `LICENSE-APACHE` y `THIRD-PARTY-NOTICES.txt` con los avisos de las bibliotecas y fuentes de terceros.

52.18. 0.6.0

Publicada el 27 de septiembre de 2026. Llega el formato `.gaanim`: una escena o presentación en un solo archivo que se reproduce, presenta y exporta en cualquier equipo. No hace falta cambiar tus escenas.

52.18.1. Cambios

- Paquetes de reproducción: `gaanim export mi-charla --output mi-charla.gaanim` graba la escena o presentación completa en un solo archivo que se reproduce, presenta y exporta a vídeo sin Python, sin el proyecto y sin sus recursos. Los callbacks, updaters y funciones reactivas quedan grabados como los fotogramas que produjeron, y un vídeo exportado desde el paquete es idéntico, píxel a píxel, al que exporta el script con la misma frecuencia y el mismo tamaño. Aún no admite 3D ni clips de vídeo (ver Compartir sin Python).

- `gaanim mi-charla.gaanim` y `gaanim --present mi-charla.gaanim` abren un paquete con el nuevo ejecutable `gaanim-play`, que se instala junto a `gaanim` y no enlaza Python. Presenter View muestra sus notas y miniaturas.
- `gaanim export mi-charla.gaanim --output mi-charla.mp4` convierte un paquete en vídeo con los formatos y opciones de siempre, y el botón Exportar del editor hace lo mismo con el paquete abierto. `--fps` elige la frecuencia al grabar un paquete.
- `gaanim check mi-charla.gaanim` vuelve a componer cada fotograma de un paquete y lo compara con el resumen grabado, sin Python.
- `gaanim --diff --example mi-charla.gaanim` captura y compara las pausas de un paquete, sin Python; coinciden con las de `--capture-stops` sobre el script.
- El inicio de Gaanim tiene la tarjeta Reproducir .gaanim (`Ctrl Shift O`), que abre el archivo en la misma ventana; también acepta arrastrarlo. Los archivos abiertos, desde el inicio o con `gaanim mi-charla.gaanim`, aparecen en Recientes junto a los proyectos. Sin Python instalado, `gaanim` abre igualmente el inicio para reproducir archivos `.gaanim`.
- El diálogo Exportar del editor ofrece el formato Gaanim para grabar un `.gaanim` del script abierto, con su barra de progreso; en la terminal, `gaanim export` muestra la suya.
- El `README.md` de los proyectos nuevos incluye cómo compartirlos como paquete.

52.18.2. Correcciones

- `drawable.bounds()` sobre un objeto recién creado (antes del siguiente `play` o `wait`, sin animaciones, grupos posteriores ni layouts que actúen sobre él) ya no compila la escena entera: compila solo su declaración y tarda alrededor de un milisegundo, con el mismo resultado. En una presentación de 41 segmentos con 54 mediciones, `gaanim check` pasa de 27,7 s a 1,7 s. En los demás casos sigue compilando la escena escrita hasta el cursor.
- Las presentaciones largas se reproducen, exportan y graban más rápido. Una presentación mantiene vivos los objetos de todos sus segmentos, y cada fotograma animado recorría todos ellos varias veces para restaurar unos pocos cientos: ahora el seek restaura en una sola pasada y la propagación de transformaciones, opacidad y cajas solo recalcula lo que cambió. Grabar el paquete de una presentación de 41 segmentos y 29 000 objetos pasa de 31 a 8½ minutos, con los mismos fotogramas.
- Grabar un paquete ya no repite la línea de tiempo en un segundo mundo cuando la escena no tiene estado que dependa de los instantes visitados (updaters, trazos acumulados, ecos, squash o animaciones personalizadas).
- En un vídeo, pulsar → (o el botón de escena siguiente) mientras sonaba la transición de una escena volvía al inicio de esa misma escena en lugar de pasar a la siguiente; ← tampoco reconocía la escena en curso durante la transición. La navegación entre escenas ahora compara instantes exactos.

52.19. 0.5.2

Publicada el 27 de septiembre de 2026. No hace falta cambiar tus escenas.

52.19.1. Cambios

- Nuevas guías de composición en los overlays del editor (`M`): márgenes seguros al 90 % y al 80 % y tercios del marco.
- Durante la inspección (`I`), la grilla y los ejes cubren toda la vista y no solo el marco de salida. En escenas 3D, la grilla se dibuja sobre el plano XZ alrededor del punto que mira la cámara, con el eje X en rojo y el Z en azul (ver Escena).

- Con la inspección activa, la barra de overlays muestra el zoom de la vista, en azul cuando ya no coincide con la cámara de la escena, junto a un botón para restablecerla (`R`). El icono de teclado lista todos los atajos.
- El editor recuerda qué overlays dejaste activados entre sesiones; el modo en sí sigue empezando oculto.

52.19.2. Correcciones

- En escenas 3D, el recuadro de selección y las coordenadas de los overlays se calculaban sobre el marco de salida en lugar de la ventana, así que quedaban desplazados respecto al objeto.

52.20. 0.5.1

Publicada el 27 de septiembre de 2026. No hace falta cambiar tus escenas.

52.20.1. Cambios

- La barra de overlays (`O`) y sus guías usan el mismo estilo que el resto del editor, con iconos, colores de la paleta y los atajos en cada botón. El punto bajo el cursor se ajusta a la grilla con Mayús, se copia con `Ctrl+C` como tupla lista para pegar y se oculta sobre los paneles. El objeto seleccionado se enmarca con su centro y su tamaño en unidades (ver Escena).
- En la inspección 2D (`I`), un clic sin arrastrar selecciona el objeto bajo el cursor, y la rueda acerca hacia el punto bajo el cursor en lugar del centro de la vista.
- El panel de error del script usa el estilo del editor: muestra arriba la excepción y el archivo y la línea donde se produjo, y debajo el traceback completo con el botón **Copiar**. El aviso de recarga también adopta el estilo del editor.
- La previsualización del editor ajusta su resolución para mantener la fluidez: si al reproducir no llega a 60 fps porque dibujar la escena cuesta demasiado, pasa al 75 % y luego al 50 % de los píxeles, y al pausar vuelve a la resolución completa, así que un fotograma detenido siempre se ve nítido. Si bajarla no acelera la reproducción (el límite está en la escena, no en el dibujo), deshace el cambio. El panel de `F12` muestra la resolución actual y `GAANIM_PREVIEW_RESOLUTION=full` la mantiene siempre completa (ver Línea de comandos). Las exportaciones no cambian.
- La previsualización ya no vuelve a rasterizar un fotograma idéntico al que muestra: en pausa, durante un `scene.wait()` o en una parada de la presentación, la GPU queda libre y el editor responde con más holgura. Los fotogramas con postprocesado o con un fondo de shader se siguen dibujando siempre, porque cambian fuera de la escena vectorial.
- Un objeto semitransparente pintado con un único color sólido (solo relleno o solo trazo; por ejemplo, las líneas de `connect` o las copias de un `repeat` con `opacity=(a, b)` cuando la figura solo tiene relleno) recibe la opacidad en su color en lugar de abrir una capa de opacidad. La imagen es la misma y dibujarlo cuesta mucho menos: con 89 círculos semitransparentes en movimiento y render por software, cada fotograma pasó de 152 a 41 ms. Se aplica a la previsualización y a las exportaciones. Los objetos con relleno y trazo a la vez siguen usando una capa, que evita que el trazo se mezcle con el relleno.
- `animate.custom` reutiliza el último resultado de tu función cuando se le pide el mismo progreso: como la función debe ser pura, los clips que ya terminaron (y los glifos de un texto que comparten la animación) no vuelven a llamar a Python en cada fotograma.
- Un número rodante en reposo ya no marca su contorno como cambiado en cada fotograma, así que el renderer no vuelve a compararlo.

- El editor muestra el primer fotograma algo antes: la previsualización solo prepara el antialiasing que usa.

52.20.2. Correcciones

- En la inspección 2D del editor (`I`), arrastrar desplazaba la escena mucho más que el cursor: con el marco por defecto de 16 × 9 unidades, un píxel movía más de una unidad. Ahora la escena sigue al cursor al mismo ritmo en cualquier marco, zoom o rotación de cámara, y `W`, `A`, `S`, `D` la desplazan a una velocidad constante en pantalla.
- El arrastre ya no cuenta el mismo movimiento dos veces (el cursor y el movimiento bruto del ratón) ni salta al volver a entrar en la ventana.
- En la inspección 2D, la rueda iba al revés que en 3D: hacia arriba alejaba. Ahora en ambos casos hacia arriba acerca.
- La selección con el ratón ya funciona con las figuras 2D de las escenas de Python, que antes no se podían seleccionar.
- En la inspección, el clic que empieza un arrastre ya no avanza a la siguiente parada, y las flechas solo navegan entre paradas y escenas en lugar de mover también la cámara. Arrastrar o usar la rueda sobre la línea de tiempo, los paneles o la barra de overlays ya no mueve la cámara.

52.21. 0.5.0

Publicada el 27 de septiembre de 2026. Cambian el pivote por defecto de las figuras declaradas con coordenadas de escena, la entrada y la salida de las vistas de cámara, los anchos de trazo fijados en un SVG y la separación de las etiquetas de cota: lee «Al actualizar».

52.21.1. Al actualizar

- Las figuras declaradas con coordenadas de escena (`line`, `polygon`, `polyline`, flechas, arcos, `curved_arrow_arc`, curvas, llaves y cotas estáticas) giran, escalan y se sesgan alrededor del centro de su caja cuando no tienen `with_pivot`. Antes su pivote era el origen de la escena, así que `arco.animate.rotate_by(...)` lo hacía orbitar alrededor de `(0, 0)` en vez de girar sobre sí mismo. Si una escena contaba con ese giro alrededor del origen, añade `.with_pivot(0, 0)`; si ya usabas `with_pivot` como remedio, puedes quitarlo cuando el pivote era el centro de la figura. Las figuras centradas en su origen, como `circle`, `rect` o `text`, no cambian.
- `CameraView.animate.pop_in()` es la salida simétrica de `pop_out()`: al llegar a la región oculta la pantalla y, en un `scene.camera.inset`, también el marco y los conectores; el siguiente `pop_out()` los vuelve a mostrar. Antes la pantalla seguía visible sobre la región (con una capa de vista, su contenido se veía encima de la escena) y el marco quedaba dibujado. Si los fundías tras `pop_in`, ya no hace falta; el fundido sigue funcionando sin parpadeos.
- Un `pop_out()` que hace de entrada deja la pantalla, y en un inset el marco y los conectores, ocultos hasta que empieza. Antes la pantalla esperaba visible sobre la región y el marco y los conectores de un inset se veían desde su declaración. Si querías ver el marco antes, anímalo aparte o usa `pop_out()` más tarde.
- El ancho fijado con `.stroke(color, ancho)` en un SVG importado, o en una de sus partes, se mide en unidades de escena aunque el SVG tenga `scale_to` o `scale_by`, igual que al animarlo con `animate.stroke(...)`. Antes se multiplicaba por esa escala, así que `.stroke(c, 1.6)` con `scale_to(0.58)` dibujaba 0.93 unidades mientras que el mismo ancho animado dibujaba 1.6. Escribe el ancho que quieres ver: `.stroke(c, 0.0093)` para el trazo fino de antes. Los trazos propios del archivo siguen escalando con el dibujo.
- `label_gap` de `mechanics.dimension_between` es la separación entre la línea y el borde más cercano de la anotación. Antes se medía hasta su centro, así que con los valores por defecto la etiqueta montaba sobre la línea. Si subías `label_gap` para despegarla, bájalo o quítalo.

- `scene.text.measure(...)` desaparece: `drawable.bounds()` mide cualquier objeto ya creado. Crea el texto y mide su caja (`scene.text("PGA", role="label").bounds().width`) en lugar de repetir su contenido y estilo. Desde 0.6.0, medir un objeto recién creado cuesta alrededor de un milisegundo; antes de esa versión cada llamada compilaba toda la escena escrita hasta ese punto.

52.21.2. Cambios

- `drawable.bounds()` devuelve la caja de cualquier objeto (texto, fórmulas, SVG, imágenes, grupos, figuras transformadas) en unidades de escena y en el cursor actual, con `x`, `y`, `left`, `right`, `bottom`, `top`, `width`, `height` y `center`. Sirve para tachar un valor o ajustar una caja a su contenido sin repetir el texto.
- `scene.stop("nombre", loop=anim)` añade un bucle ambiental: mientras la presentación descansa en la pausa, la animación se repite en lugar de congelar la imagen, así que un movimiento continuo sigue mientras hablas. El siguiente paso sale del bucle; al exportar se reproduce una vez. `scene.stops` informa de su duración en `loop_duration`.
- `scene.camera.inset(..., aspect=)` o `size=(ancho, alto)` da a un inset rectangular su propia proporción, para encuadrar entero un panel alto o una barra de herramientas en lugar de recortarlo a 16:9.
- `imagen.pixel(px, py)` convierte un píxel del archivo original (desde la esquina superior izquierda) en un punto de anclaje que sirve en `move_to`, `scene.camera.inset`, `pan_to` o conectores y sigue a la imagen.
- `drawable.matrix_to(((a, b), (c, d)))` y `animate.matrix_to(...)` aplican una transformación lineal 2D cualquiera alrededor del pivote, por ejemplo para llevar una cara a una proyección isométrica sin trocearla.
- Cámaras secundarias: `pantalla.camera_view(marco)` convierte una figura cerrada en una pantalla que muestra lo que ve una segunda cámara, como un *picture-in-picture* o una lupa, y devuelve un `CameraView`. El marco es la cámara: su centro marca adónde mira y su giro gira la vista, y sin `zoom=` su tamaño fija el aumento. La vista se recorta al contorno de la pantalla, entre su relleno y su trazo, y admite `fit`, `background`, `exclude` y `layers`; `no_camera_view()` la quita.
- `CameraView` tiene el vocabulario de `scene.camera`: `pan_to`, `zoom_to`, `rotate_to`, `follow` y `animate`. Con `zoom=` (un número, un `Parameter` o un `Computed`) el aumento es explícito, `animate.zoom_to` lo cambia a velocidad percibida constante y `view.zoom` lo expone para lecturas; sin marco, `center=` elige adónde mira. `animate.pop_out()` hace brotar la pantalla de la región que ve su cámara, sin saltos, y `animate.pop_in()` la devuelve.
- `scene.camera.inset(objetivo, zoom=, at=)` monta de una vez la pantalla, un marco que recuadra lo que muestra y dos conectores. Con `follow=True` el marco persigue a un objeto en movimiento y con `fixed=True` la pantalla se queda fija en la imagen aunque la cámara principal se mueva.
- `drawable.view_layer("nombre")` saca un objeto de la cámara principal: solo lo muestran las vistas que listan esa capa en `layers=`, por ejemplo una lupa de rayos X.

Ver `Drawable.camera_view` en Drawable, las vistas de cámara en Escena y las cámaras secundarias en Cámara y 3D.

- `drawable.blend("screen")` cambia cómo se funde un objeto con lo que hay debajo: `"screen"` y `"add"` para luces y destellos, `"multiply"` para resaltadores, y el resto de modos habituales (`"overlay"`, `"soft_light"`, `"difference"`…).

- `animate.dash_offset(valor)` anima el desplazamiento de los guiones de un trazo, y `Updater.dash_flow(speed=)` los hace fluir sin fin, como hormigas en marcha o el flujo de una tubería. Ambos respetan los seeks y la exportación.
- `drawable.tip(end="arrow", start="dot")` pone puntas de flecha o puntos en los extremos de cualquier trazo (polilíneas, curvas, arcos). Siguen al extremo actual del camino, así que `animate.grow_arrow()`, `create()` y `trim` las llevan consigo.
- `scene.viz.progress_ring(0.0)` crea un anillo de progreso con un porcentaje rodante en el centro; se anima con `ring.animate.set(0.75)`. `scene.viz.countdown(10)` crea un temporizador que `timer.count_down()` vacía en tiempo real.
- `PostProcess.shader(src, uniforms={"amount": parametro})` enlaza valores del shader a un `Parameter`, así que el efecto se anima como cualquier otro valor. `scene.canvas.post` (y `post=` de `Scene` y de los segmentos) acepta una lista de pasadas que se encadenan en orden.
- Presets de acabado sin escribir WGSL: `PostProcess.grain`, `vignette`, `chromatic_aberration`, `color_grade`, `lut` (archivos `.cube`), `halftone`, `dither`, `crt`, `pixelate` y `glitch`. Sus valores también aceptan un `Parameter`.
- `PostProcess.bloom(threshold, intensity, radius)` hace brillar lo que supera el umbral con una cadena de mips, igual en el visor y en la exportación.
- Fondos vivos: `Background.mesh_gradient`, `noise_gradient`, `aurora` y `dot_grid`. Los shaders de fondo propios pueden llamar a `gaanim_frame_size(resolution)` para trabajar en unidades de la escena.
- `drawable.echo(5, delay=0.04, decay=0.6)` deja copias que siguen al objeto en el tiempo; son exactas en cualquier búsqueda y también salen en SVG.
- `animate.points([...])` mueve cada vértice de un polígono o una polilínea en línea recta a su nueva posición, sin remuestrear el contorno: con `echo` reproduce el símbolo de Gaanim fotograma a fotograma.
- `drawable.squash_stretch(0.1)` estira un objeto en la dirección de su velocidad y lo aplasta en la perpendicular, conservando el área.
- `drawable.stroke_profile([...])` y `stroke_taper(start, end)` dan trazos de grosor variable que se afinan en los extremos, también en `create` y `show_passing_flash`.
- `scene.geometry.repeat(...)` y `scene.geometry.duplicate(shape, Distribution.grid/circle/along/random/phyllotaxis(...))` crean copias agrupadas de una figura; `grupo.animate.count(n)` las hace aparecer una a una, fundiendo la siguiente con la parte fraccionaria, y `scene.geometry.connect(...)` une puntos con líneas vivas que se desvanecen con la distancia (plexus).
- `scene.canvas.motion_blur(180, samples=8)` difumina el movimiento en las exportaciones y capturas promediando subcuadros; `drawable.motion_blur(False)` mantiene nítido un objeto.

Ver `Drawable.blend`, `Drawable.tip`, `Drawable.echo` y `Drawable.motion_blur` en Drawable, `Anim.dash_offset`, `Anim.points`, `Anim.count` y `Updater.dash_flow` en Animaciones, los anillos en Visualización y los fondos, el postprocesado y `Canvas.motion_blur` en Colores y temas.

52.21.3. Correcciones

- `.hud()` fija de verdad el objeto a la imagen también en 2D: antes se movía con `scene.camera` al desplazarla, acercarla o girarla. Las escenas con la cámara 2D quieta se ven igual.

- `.hud()` funciona en figuras simples como rectángulos, círculos o polígonos, que ahora se dibujan encima de la escena como el resto de superposiciones; antes la llamada no tenía efecto en ellas.
- `grow_from_center()` crece desde el centro de la caja aunque la figura se declare con coordenadas absolutas, como una línea entre dos puntos o un polígono de vértices fijos. Antes escalaba desde el pivote, que en esas figuras es el origen de la escena, y la figura viajaba desde `(0, 0)` hasta su sitio mientras crecía. Ya no hace falta `with_pivot` como remedio.
- `grow_from_edge` y `grow_from_point` mantienen fijo el punto indicado también cuando el objeto tiene un pivote propio con `with_pivot`.
- Tras `grow_from_edge` o `grow_from_point`, un `scale_by` o `scale_to` posterior parte del tamaño declarado en lugar de escala cero.
- `scene.mechanics.dimension_between` se ve desde su declaración, como cualquier otro objeto sin animación de entrada. Antes la línea y las extensiones solo aparecían con una entrada en `scene.play` y, si tenía etiqueta, esta se veía sola. Con `fade_in` o `create` sigue oculta hasta que la entrada empieza.
- Un `rotate_by` con pivote (`with_pivot` o `.pivot(x, y)`) sobre un objeto ya girado empieza desde su pose actual; antes saltaba al empezar el giro.

52.22. 0.4.2

Publicada el 26 de septiembre de 2026. No requiere cambios en tus escenas.

52.22.1. Cambios

- `drawable.skew_to(x, y)` y `drawable.animate.skew_to(x, y)` sesgan un objeto o un grupo entero alrededor de su pivote, sin trocearlo: `x` desplaza cada punto en horizontal `x` veces su altura sobre el pivote, e `y` en vertical `y` veces su distancia horizontal a él. Con `with_pivot` en la base, `skew_to(0.15, 0)` inclina una vivienda como el corte de un pórtico en un sismo. Ver `Drawable.skew_to` en Drawable.
- `gaanim --diff` ya no abre un visor nativo: al comparar escribe `report/index.html`, que imprime al terminar y se abre en el navegador sin conexión. El informe añade una lista filtrable con miniaturas, siete modos de comparación (diff, versión aprobada, actual, lado a lado, cortina, superposición y parpadeo), un recuadro sobre el área cambiada, zoom con píxeles nítidos, atajos de teclado y enlaces a un fotograma concreto. Ver Capturas y comparación visual. La opción `--no-gui` desaparece y ahora es un error: quítala de tus scripts y de la integración continua.
- El ejecutable ya no incluye la interfaz, los sprites, el texto ni los gizmos de Bevy, que Gaanim no usaba: el lienzo de Vello se compone en la ventana con un pase propio. Las escenas se ven igual; solo puede cambiar qué línea 3D queda encima cuando dos líneas transparentes coinciden exactamente.

52.22.2. Correcciones

- `Transition.zoom_through` muestra solo el segmento saliente mientras la cámara se acerca y solo el entrante mientras se aleja; antes los dos se veían durante toda la transición. El zoom se centra en `center`, que antes se ignoraba.
- `gaanim check` y `gaanim --diff` muestran en Windows las rutas sin el prefijo `\\?\`.
- `gaanim check` y la vista previa ya no pueden quedarse colgados buscando Python: el gestor de instalación de Python de Windows ya no intenta instalar una versión que falta, y cada comprobación se abandona a los 10 segundos.

52.23. 0.4.1

Publicada el 26 de septiembre de 2026. Además de correcciones, cambia el tema por defecto y la escala de los SVG: lee «Al actualizar» antes de actualizar un proyecto existente.

52.23.1. Al actualizar

- `Scene()` usa por defecto el tema `technical`: fondo gris casi negro `#121212`, texto y ejes claros y formas sin color propio rellenas con el acento ámbar `#F2A541`. Antes una escena empezaba sin tema y en blanco, y el texto y las formas sin color no se veían. `scene.canvas.theme` vale `"technical"`, y `scene.canvas.color(...)` y `validate_theme()` ya no lanzan `ValueError` en una escena nueva.
- El tema `technical` (y `Theme()` sin base) cambia su fondo azulado `#0B1018` por el gris neutro `#121212`, con texto `#E6E6E6`, texto atenuado `#A0A0A0`, panel `#1C1C1C`, cabecera `#2A2A2A` y regla `#707070`, y sus acentos azules por un ámbar `#F2A541` (formas y énfasis) y un gris neutro `#BDBDBD` para los datos de los gráficos: ningún color del tema tiene tinte azul.
- El tema `presentation` usa la misma base neutra: fondo `#121212` (antes el azul marino `#070B16`), texto `#F2F2F2`, texto atenuado `#A8A8A8` y paneles y reglas grises; conserva sus títulos y acentos dorados, y sus gráficos pasan del azul `#5B8FFF` al coral `#F4845F`.
- Las operaciones booleanas (`union`, `intersection`, `difference`, `xor`) y las máscaras de `fill_level` aproximan las curvas con una tolerancia de `0.0025` unidades en lugar de `0.25`, un valor que había quedado en escala de píxeles: los círculos vuelven a salir suaves en vez de poligonales.
- `scene.viz.matrix(...)` vuelve a aplicar los valores documentados en unidades lógicas: `row_gap` y `column_gap` de `0.24` (antes `24`) y un delimitador automático de `max(0.72, 0.6 × filas)` (antes `max(72, 60 × filas)`). Con los valores en píxeles, una matriz de 3 × 3 dejaba sus entradas fuera del marco y el paréntesis de una de 2 × 2 lo tapaba entero; las matrices con separaciones y `delimiter_size` explícitos no cambian.
- Lo explícito sigue ganando: `Scene(background=...)` conserva su fondo, `Scene(theme=...)` y `scene.canvas.set_theme(...)` sustituyen al tema predeterminado y los colores de cada objeto ganan a los del tema.
- Para recuperar el lienzo anterior, sin tema y blanco, usa `Scene(theme=None)` o `scene.canvas.set_theme(None)`; `gaanim check` avisa si el texto y las formas sin color no se verían sobre el fondo.
- En el stub, los nombres de tema tienen el tipo `ThemeName`, así que el editor autocompleta los temas incluidos y sus alias.
- `scene.media.svg(path)` importa el documento a 100 píxeles SVG por unidad lógica, la misma equivalencia que el resto del motor (un trazo de 3 px mide 0.03, como el trazo por defecto), en lugar de una unidad por píxel. Un SVG de 360 × 220 px ahora mide 3.6 × 2.2 unidades en vez de desbordar el marco de 16 × 9; los trazos, degradados, `clipPath`, texto y filtros del archivo escalan igual. Es un cambio incompatible: si tus escenas compensaban la escala anterior con `scale_to(f)` o `scale_by(f)`, multiplica ese factor por 100 (`scale_to(0.025)` pasa a `scale_to(2.5)`) o quítalo si el tamaño natural te sirve. En esta versión el ancho fijado con `.stroke(color, ancho)` escalaba con `scale_to`, así que también había que dividirlo entre 100; a partir de la siguiente se mide en unidades de escena (ver «Sin publicar»).
- El texto de un SVG con la familia genérica `sans-serif` usa la DejaVu Sans que trae Gaanim en cualquier sistema. Antes salía en Arial si estaba instalada y desaparecía en los equipos Linux que no la tienen.
- `gaanim check` imprime su informe con el nuevo formato de la terminal (`pass`, `warning`, `error`, `fail`) en lugar de las líneas `PASS:`, `WARN:` y `ERROR:`. Si un script leía ese texto, usa el código de salida, que no cambia: `0` sin errores, `1` con problemas y `2` si el proyecto no carga.

- `scene.geometry.line(p0, p1)` con dos puntos fijos es ahora una línea normal: acompaña al grupo que la contiene cuando lo desplazas. Las líneas con extremos que son objetos o referencias siguen recalculándose.

52.23.2. Cambios

- En la línea de tiempo del editor, la escena actual se distingue con el color de acento, un subrayado y un contorno. La escena bajo el cursor se ilumina, y queda contorneada cuando un clic saltaría a su inicio. Una línea vertical separa cada escena de la siguiente en los carriles de escenas y de reproducción.
- La terminal muestra solo lo que te sirve, con color y columnas alineadas: el logo de Gaanim al abrir la vista previa, exportar o pedir `--help`, y una línea por evento con su etiqueta (`watch`, `ready`, `reload`, `export`, `python`…), donde los detalles secundarios aparecen atenuados. Los mensajes internos de Bevy, wgpu, winit y egui (adaptador, shaders, ventana) solo se muestran si son errores; los de ALSA se resumen en un único aviso `audio`, y las trazas de Python empiezan en tu código y resaltan el error. `gaanim check` usa el mismo formato: `pass`, `warning`, `error` o `fail`. Consulta Solución de problemas para ver de nuevo todos los mensajes o quitar el color.
- `gaanim --help` y la ayuda de cada comando están reescritas: incluyen ejemplos, las teclas del modo presentación, las variables de entorno y el enlace a esta documentación.
- Las exportaciones respetan `WGPU_BACKEND` (`vulkan`, `dx12`, `metal`, `gl`) igual que la vista previa, para elegir el backend de la GPU cuando el que se elige por defecto falla.
- `move_along(path, start=0.5, end=0.2)` recorre el tramo al revés en lugar de lanzar `ValueError`, y `move_along` sobre una flecha sólida sigue su eje de la cola a la punta.
- `stroke(color, ancho, align=...)`, con `"inside"`, `"center"` u `"outside"`, coloca el contorno en drawables, glifos de texto y `SurroundingRect`. Con `"outside"` puedes dar un halo a un texto.
- `--sections` y `--from` aceptan el nombre de un `SectionStep`.

52.23.3. Correcciones

- Las exportaciones ya no salen en negro cuando la escena tiene muchas líneas o muchos objetos translúcidos, sobre todo a resoluciones altas. Si un fotograma no se puede dibujar, la exportación falla indicando cuál en vez de escribirlo en negro.
- `rotate_by` con pivote, `grow_from_center`, `grow_from_point`, `grow_from_edge`, `spin_in_from_nothing`, `fade_in_from` y `move_along` ya no muestran al principio la pose de un momento posterior de la línea de tiempo. Las rotaciones con pivote de más de media vuelta giran de forma continua.
- `show_passing_flash` mantiene el objeto oculto antes del destello, y `trim()` y los destellos funcionan también en líneas que se recalculan cada fotograma (líneas entre extremos, conectores, gráficas reactivas).
- `follow()` sigue correctamente a los puntos de `point_on_curve`.
- Un grupo declarado después de `wait` o `play` ya no oculta a sus miembros que eran visibles desde el principio.
- Los números de `readout` y `rolling_number` usan por defecto el color del texto del cuerpo; antes quedaban sin relleno si no había tema ni color.
- Los conflictos entre animaciones del mismo canal nombran el tipo de objeto y las dos animaciones implicadas.
- En Windows, `__file__`, `sys.argv` y `sys.path` usan rutas normales, sin el prefijo `\\?\`.

52.24. 0.4.0

La versión del diseño de movimiento: resortes, easings expresivos, repeticiones, escalonado espacial, animación de texto, transiciones y narración.

52.24.1. Movimiento

- `Easing.spring(bounce=...)` con un rebote exacto, masa y velocidad inicial, y los presets `GENTLE`, `QUICK`, `SNAPPY`, `BOUNCY` y `SMOOTH_SPRING`.
- Nuevos easings: `Easing.back`, `elastic`, `bounce`, `slow_mo`, `rough`, `squish`, `from_svg`, `steps(jump=...)` y `Easing.custom`.
- `Anim.repeat(count, yoyo=, delay=)`, `Anim.loop(...)` y `Composition.repeat(...)`.
- `move_along(..., orient=True)` orienta el objeto según la trayectoria y `Anim.path_arc(ángulo)` curva un `move_to` o `shift_by`.
- `stagger` acepta `origin=`, `grid=`, `total=` y `easing=` para escalonar por distancia; `distribute` reparte valores con el mismo orden.
- `grow_from_edge` y `grow_from_point`.
- `scene.random(seed)` y `scene.noise(...)` dan azar y ruido reproducibles; `Updater.wiggle` y `Updater.oscillate` añaden movimiento procedimental que se suma a las animaciones.
- `trim()` y `animate.trim(...)` recortan trazos; `animate.glow`, `animate.blur` y `animate.shadow` animan efectos.
- `label("nombre")` y `Composition.insert(item, at=...)` colocan animaciones respecto de etiquetas; `scene.marker("nombre")` marca instantes de la línea de tiempo que el editor muestra y que `gaanim export --from/--to` acepta.

52.24.2. Texto

- `Text.animator(...)` anima rangos de caracteres, palabras o líneas con desplazamiento, opacidad, escala, rotación, desenfoque, espaciado y color.
- `reveal(...)` y `conceal(...)` con máscaras, `blur_in(...)` y `tracking(...)`.
- `typewriter(...)` con cursor, `backspace(...)` y `retype(...)`; `scramble(...)` y `scramble_to(...)`.
- `selection.animate.marker(...)` subraya como un rotulador; las selecciones de texto también tienen `reveal`, `brace` y `annotate`.
- `write(by=, order=)` respeta por fin sus argumentos.
- Las marcas negativas de los ejes usan el signo menos tipográfico; `readout` y `variable` aceptan `decimal_separator`; `scene.text.measure` acepta `flow` y `line_spacing`; `lang` elige el idioma de la separación silábica.

52.24.3. Transiciones y cámara

- `Transition.wipe`, `clock_wipe`, `iris`, `blinds`, `push` y `Transition.morph`; todas las transiciones aceptan `easing=`. `Overlay.flash` y `Overlay.light_leak` se superponen al corte.
- `camera.animate.shake(trauma=...)` produce un temblor que decae de forma natural.
- `zoom_to` y `frame_to` interpolan el zoom de forma exponencial por defecto.
- Los conectores de `scene.geometry.connector` crecen desde la cola con `grow_arrow()`.

52.24.4. Narración

- `scene.voiceover(clave)` marca bloques cuyo ritmo marca una toma grabada, y `scene.live_take(clave)` sustituye las pausas por las que hiciste al presentar. Los textos pueden vivir en un guion Markdown.
- El editor graba el micrófono con teleprompter, medidor de nivel y normalización con FFmpeg.
- El audio de la vista previa sigue al cabezal al saltar.

52.24.5. Otros cambios

- `scene.geometry.points(...)` dibuja miles de puntos como un solo objeto.
- `gaanim export --from/--to` exporta un tramo; `gaanim --version` y `gaanim export --help`.
- `drive_from_samples(..., "xy")` mueve los dos ejes con pares `(x, y)`, y controlar `"x"` y luego `"y"` conserva ambos canales.
- Las secuencias PNG aceptan `%d` o `%0Nd` en el nombre.

- El Inicio del editor permite crear o abrir proyectos, lista los recientes y comprueba uv, Python 3.14 y FFmpeg.

52.24.6. Correcciones

- La exportación usa la misma cámara que la vista previa: el temblor, los vínculos, el seguimiento y el encuadre dinámico ya aparecen en los videos.
- `traced_path` se reconstruye correctamente al saltar a otro instante.
- Las cadenas de `Anim` no válidas lanzan `TypeError` o `ValueError` en vez de un error interno.
- `gaanim --diff --capture-stops` sin una versión aprobada de pausas captura y termina con éxito; varias correcciones en selecciones de texto con ligaduras y en varias líneas.

52.24.7. Al actualizar

- Si un zoom de cámara debe interpolarse de forma lineal, añade `interpolation="linear"` a `zoom_to` o `frame_to`.

52.25. 0.3.0

- **Postprocesado:** `PostProcess.shader(código_wgsl)` aplica un shader a toda la imagen 2D en el editor, al presentar, en las capturas y en la exportación. Se configura para toda la escena con `Scene(post=...)` o por segmento con `scene.segment(..., post=...)`.
- **Recarga al cambiar recursos:** guardar una imagen, un SVG, un Lottie, un shader o un documento Typst del proyecto recarga la escena (ver Proyectos y exportación).
- Los documentos de `scene.text.typst()` tienen el mismo tamaño que el texto del cuerpo.
- Los objetos ya no aparecen completos antes de su `write` o `create` tras rebobinar.

52.25.1. Al actualizar

- Si compensabas el tamaño de `scene.text.typst()` con `scale_to(2)` o `scale_to(3)`, quítalo.

52.26. 0.2.1 a 0.2.3

- **0.2.3:** Gaanim exige Python 3.14 (exactamente 3.14 en Linux, 3.14 o posterior en Windows) y encuentra en Linux los intérpretes instalados con uv. Si tu `.venv` usa otra versión, recréalo con `uv venv --python 3.14`.
- **0.2.2:** la recarga en caliente es incremental, a partir del primer segmento que cambió. Nueva barra de reproducción para líneas de tiempo densas, nuevos diálogos de exportación y nuevo estilo de Presenter View; `Space` solo pausa y ya no avanza de paso. Las líneas discontinuas se dibujan guion a guion con `create`.
- **0.2.1:** medir texto es mucho más rápido, lo que acelera la recarga de escenas con muchas insignias, chips o tarjetas.

52.27. 0.2.0

La versión que separa `Scene` en capacidades y pasa de píxeles a unidades lógicas.

- `Scene` orquesta el tiempo y la presentación; las fábricas se agrupan en `scene.geometry`, `scene.text`, `scene.layout`, `scene.media`, `scene.viz`, `scene.slides`, `scene.mechanics` y `scene.assets`.
- Las escenas se componen en un fotograma lógico de 16 × 9; la resolución se elige al previsualizar o exportar.
- La reactividad se expresa con `computed()` y entradas explícitas en lugar de expresiones simbólicas.

- `scene.cursor` y `scene.stops`; `gaanim --diff --capture-stops`.
- `scene.sections` con agenda y barra de progreso; `--sections` y `--from` para ensayar una parte de una presentación.
- Los ejes regeneran sus marcas y remuestrean sus curvas en cada `view_to`, y recortan los datos a la ventana; `data_to_scene` para anotaciones.
- `markup=False` en textos; `parts()` acepta un diccionario de nombres; `font_dir` y marcado en los temas; `side=` y tipografía en `dimension_between`; `grow_arrow` sin deformar la punta.
- Los valores por defecto que seguían en píxeles pasan a unidades de escena.
- Si falta FFmpeg, la exportación lo explica.
- El paquete incluye esta documentación para los asistentes de código.

52.27.1. Migrar desde 0.1

Gaanim 0.2 convierte `Scene` en el orquestador del tiempo y la presentación. Las fábricas conservan sus firmas y comportamiento, pero se acceden mediante una capacidad ligada a la misma escena. No existen alias para los métodos planos retirados.

52.27.1.1. Unidades lógicas

Las escenas ya no se componen en píxeles. `Scene()` usa un fotograma lógico centrado de `16×9`; la resolución pertenece al editor o al comando de exportación.

```
# Antes: composición acoplada a 1920×1080
scene = Scene(1920, 1080, margin=60)
circle = scene.geometry.circle(120).move_to(360, -120)

# Ahora: las mismas proporciones en unidades lógicas (escala 1920 / 16 = 120)
scene = Scene(frame=(16, 9), margin=0.5)
circle = scene.geometry.circle(1).move_to(3, -1)
```

`Scene(1280, 720)` produce un `TypeError` de migración. Usa `Scene(frame=(16, 9))` y elige los píxeles con `--width` y `--height`. Una salida con aspecto distinto requiere `--fit contain` o `--fit cover`.

52.27.1.2. Equivalencias

0.1	0.2
`scene.circle(r)`	`scene.geometry.circle(radius)`
`scene.equation(...)`	`scene.text.equation(...)`
`scene.row(children)`	`scene.layout.row(children)`
`scene.image(path)`	`scene.media.image(path)`
`scene.parameter(value)`	`scene.viz.parameter(value)`
`scene.badge(text)`	`scene.slides.badge(text)`
`scene.force_at(...)`	`scene.mechanics.force_at(...)`
`scene.assets_dir(path)`	`scene.assets.assets_dir(path)`
`dot.at(x, y)`	`dot.move_to(x, y)`
`dot.move_to(x, y)` (animado)	`dot.animate.move_to(x, y)`
`tracker.animate_to(v)`	`tracker.animate.set(v)`
`space.animate_view(x, y)`	`space.animate.view_to(x, y)`
`matrix.scale_by(k)` (álgebra)	`matrix.scalar_multiply(k)`
`AnimationGroup(a, b)`	`parallel(a, b)`
`Succession(a, b)`	`sequence(a, b)`
`LaggedStart(a, b, lag=t)`	`stagger(a, b, each=t)`

`scene.play(items, lag=t)`	`scene.play(stagger(*items, each=t))`
`.smooth()`	`.easing(Easing.SMOOTH)`
`.linear()`	`.easing(Easing.LINEAR)`
`.spring()`	`.easing(Easing.spring(stiffness=90, damping=12))`
`.ease("spring")`	`.easing(Easing.spring(stiffness=300, damping=20))`
`.steps(n)`	`.easing(Easing.steps(n))`
`scene.play(items, rate="linear")`	`scene.play(items, easing=Easing.LINEAR)`
`obj.animate.write(0.8)`	`obj.animate.write().duration(seconds=0.8)`
`obj.at_anchor(x, y, anchor)`	`obj.move_to(x, y, anchor=anchor)`
`obj.color(value)`	`obj.fill(value)`

Los métodos inmediatos (como `fill` o `move_to` sin `.animate`) son cortes reversibles en el cursor actual y no consumen tiempo. `animate` nunca se invoca: es una propiedad de solo lectura. El `Anim` resultante es una descripción pura y `Scene.play` valida el lote completo antes de incorporarlo a la línea de tiempo.

`scene.text("Hola")` no cambia. Desde 0.2, `scene.text` es una capacidad invocable `Typography` que también expone `equation`, `typst`, `measure` y `code`.

`Easing` y `EasingCurve` se importan desde `gaanim`. Son objetos tipados e inmutables, por lo que el autocompletado enumera presets y familias y los nombres desconocidos fallan en vez de degradarse a otra curva.

52.27.1.3. Qué permanece en Scene

`play`, `wait`, `stop`, `fade_out_all`, `segment`, `link`, `reuse`, `persist`, `release`, `render` y `snapshots` continúan directamente en `Scene`. `canvas` expone ahora `frame_width`, `frame_height`, `aspect_ratio` y los límites del área segura; `camera` opera sobre esas mismas unidades lógicas.

52.27.1.4. Expresiones simbólicas

`gaanim_expr`, `_Expr` y `gaanim.math` se eliminaron sin adaptadores. Usa estas equivalencias:

- `gm.sin(parameter)` pasa a `computed(f, inputs=[parameter])`, con `f = lambda value: math.sin(value)`.
- `lambda x: amplitude * gm.sin(x)` pasa a una función que recibe la amplitud como argumento, `lambda x, amplitude: ...`, con `inputs=[amplitude]`.
- Un `readout` derivado declara sus entradas: `scene.viz.readout(area, inputs=[radius])`, con `area = lambda r: math.pi * r**2`.
- Extremos, cámara y valores escalares aceptan `float`, `Parameter`, `Variable` o `Computed`; pasar un parámetro directamente conserva su identidad.
- `plot(function, derivative=2)` pasa a `plot(function, derivative=d2)`, donde `d2` es la segunda derivada escrita como función. La derivada debe tener la misma firma y las mismas entradas; no hay derivación numérica implícita.
- El tiempo se declara con `inputs=[scene.viz.time]`.

La guía de Reactividad explica el modelo actual.

52.27.1.5. Ejemplo completo

```python
from gaanim import BLUE, Scene

scene = Scene(frame=(16, 9))
circle = scene.geometry.circle(0.96).fill(BLUE)
title = scene.text("Capacidades", role="title")
```

```
page = scene.layout.column([title, circle], within="safe", gap=0.24)
scene.play([page.animate.fade_in()])
scene.render()
```

CAPÍTULO

53. Glosario

Los términos que usa esta documentación y el nombre que tienen en la API

Esta documentación explica en español y conserva en inglés los nombres de la API. Aquí tienes cada término con el identificador que le corresponde en el código.

Ancla (`Anchor`, `anchor_point`) uno de los nueve puntos de referencia de la caja de un objeto (`Anchor.CENTER`, `Anchor.TOP_LEFT`…). Sirve para colocar objetos (`move_to(x, y, anchor=...)`) y, con `objeto.anchor_point(...)`, como extremo que acompaña al objeto cuando se mueve.

Animación (`Anim`) la descripción de un cambio en el tiempo, como `circle.animate.move_to(2, 0)` con `.duration(1.0)`. No hace nada hasta que la pasas a `scene.play([...])`.

`.animate` propiedad de solo lectura de cada handle que ofrece los mismos métodos que los cambios inmediatos, pero devuelve animaciones.

Área segura (`within="safe"`) el fotograma menos el margen de la escena (`Scene(margin=...)`). Layout la usa para no pegar el contenido a los bordes.

Binding (vínculo) relación por la que una propiedad copia la de otro objeto, como `bind_x_from`, `bind_rotation_from` o `follow`. Ver Reactividad.

Capacidad cada grupo de fábricas de una escena: `scene.geometry`, `scene.text`, `scene.layout`, `scene.media`, `scene.viz`, `scene.slides`, `scene.mechanics`, `scene.assets`, `scene.canvas` y `scene.camera`.

Captura (`scene.snapshots`) imagen exacta de un instante de la escena, que `gaanim --diff` compara con una versión aprobada. Ver Capturas y comparación visual.

Composición (`Composition`) grupo de animaciones ordenadas con `parallel`, `sequence` o `stagger`, que se reproduce como una unidad.

Corte cambio inmediato, sin duración, que un método aplicado fuera de `.animate` registra en el cursor actual (por ejemplo `circle.fill(RED)` después de un `play`). Es reversible: al volver atrás en la línea de tiempo, el objeto recupera su estado anterior.

Cursor (`scene.cursor`) el instante de la línea de tiempo en el que se añade lo siguiente que escribes. Avanza con cada `scene.play` y `scene.wait`.

Drawable cualquier objeto visible de la escena (una figura, un texto, una imagen, un gráfico, un layout). En español decimos también **objeto**.

Easing (`Easing`) la curva que reparte el progreso de una animación en el tiempo: `Easing.LINEAR`, `Easing.SMOOTH`, `Easing.spring(...)`… Se aplica con `.easing(...)`.

Escena (`Scene`) el documento que describes en Python. Contiene los objetos, la línea de tiempo, el tema y la cámara, y termina con `scene.render()`.

Exportar (`gaanim export`) renderizar la escena a un archivo de video, imagen animada o secuencia PNG. Ver Proyectos y exportación.

Fábrica método que crea un drawable y devuelve su handle, como `scene.geometry.circle(1)` o `scene.text("Hola")`.

Fragmento parte de un texto seleccionada por su contenido, como `text["energía"]`, para colorearla o animarla por separado. Si el texto tiene una parte con ese nombre, se usa la parte.

Handle el valor de Python que devuelve una fábrica y con el que manipulas el objeto: `circle = scene.geometry.circle(1)`. Sus métodos son encadenables: `circle.fill(BLUE).move_to(2, 0)`.

Layout sistema que coloca objetos en filas, columnas, rejillas y capas sin coordenadas manuales (`scene.layout`). Ver Layout.

Línea de tiempo la secuencia de todo lo que ocurre en la escena, con su instante exacto. Gaanim puede saltar a cualquier punto de ella y mostrar lo mismo que al reproducir.

Manifiesto (`gaanim.toml`) archivo que define un proyecto: nombre, tipo, script de entrada y carpetas de recursos y exportación.

Marcador (`scene.marker`) nombre de un instante de la línea de tiempo. El editor lo muestra en la barra de reproducción y `gaanim export --from/--to` lo acepta en lugar de segundos.

Parada (`scene.stop`) punto donde el modo presentación espera a que avances. La vista previa, las capturas y la exportación no se detienen en ella. En la vista del presentador cada parada es un **paso**.

Parámetro (`Parameter`, `scene.viz.parameter`) valor numérico invisible y animable del que pueden depender otros objetos.

Parte (`part`) fragmento de un texto o una ecuación con nombre propio, como `part("masa", "m")`, que puedes seleccionar con `text["masa"]` y que las transformaciones emparejan por nombre.

Presenter View (vista del presentador) ventana con la diapositiva actual, la siguiente, las notas y el tiempo, que se abre con `gaanim --present`. Ver Presentaciones.

Proyecto carpeta con un `gaanim.toml`, un script de entrada, recursos y exportaciones. Se crea con `gaanim init video` o `gaanim init slides`.

Punto de referencia (`PointRef`) punto lógico, sin dibujo, que otros objetos pueden seguir o usar como extremo; lo crean `polar_point`, `point_ref`, `offset_point` y `point_between`.

Reactivo se dice del objeto o valor que se recalcula a partir de otros en cada fotograma, como una `tracking_line` o un `computed`.

Recarga en caliente la vista previa vuelve a ejecutar el script al guardar y conserva lo que no cambió.

Rol de texto (`role=`) estilo con nombre que el tema define para un texto: `"title"`, `"subtitle"`, `"body"`, `"caption"`…

Segmento (`scene.segment`) tramo con nombre de la línea de tiempo. En una presentación, cada segmento es una diapositiva; entre segmentos puede haber una transición.

Stagger (`stagger`) composición que arranca varias animaciones con un desfase entre ellas, por orden o por distancia a un origen.

Tema (`theme=`, `Theme`) conjunto de colores, tipografías y estilos de texto que la escena aplica por defecto, como `"presentation"` o `"technical"`.

Transición (`Transition`) el paso visual de un segmento al siguiente, como `Transition.cross_fade(0.4)` o `Transition.wipe(...)`.

Unidades lógicas la medida de toda la escena. El fotograma mide 16 × 9 unidades con el origen en el centro; los píxeles solo se eligen al previsualizar o exportar.

Updater (`Updater`, `add_updater`) comportamiento continuo añadido a un objeto, como `Updater.rotate(1.5)` o `Updater.wiggle(...)`. Ver Reactividad.

Valor calculado (`computed`) número derivado de parámetros, variables o del tiempo mediante una función de Python pura con entradas explícitas.

Versión aprobada (baseline) el conjunto de capturas aceptadas contra el que `gaanim --diff` compara. Se actualiza con `--bless`.

Vista previa la ventana que abre `gaanim archivo.py` o `gaanim proyecto`, con reproducción, saltos y recarga en caliente.

CAPÍTULO

54. Solución de problemas

Qué hacer cuando una escena no se abre, no se ve como esperas o no se exporta

Los problemas más habituales, agrupados por el momento en que aparecen. Para cada uno se indica la causa y qué hacer. Antes de nada, ejecuta `gaanim check <script-o-proyecto>`: ejecuta la escena sin ventana y muestra el error de Python con su línea.

54.1. Instalación y Python

Los errores al arrancar (`Python 3.14 was not found`, una versión de Python incorrecta, `python3.dll` no encontrado, `authoring environment not ready`) están explicados en la sección de problemas de Instalación.

- **`python main.py` falla:** es lo esperado. El paquete de Python solo contiene los tipos para tu editor; las escenas se ejecutan siempre con `gaanim main.py` o `gaanim .`.
- **Mi editor no autocompleta la API:** configura como intérprete el `.venv` del proyecto, donde `gaanim init` instala el paquete de autoría.

54.2. Abrir la escena

- **`gaanim check: could not load project`:** revisa que `entry` en `gaanim.toml` sea una ruta relativa a un archivo que exista dentro del proyecto.
- **`TypeError` al crear `Scene(1920, 1080)` o `Scene(1280, 720)`:** las escenas ya no se definen en píxeles. Usa `Scene(frame=(16, 9))` y elige la resolución al exportar (ver Novedades).
- **`AttributeError` con `scene.circle`, `scene.equation` y similares:** las fábricas viven en capacidades: `scene.geometry.circle`, `scene.text.equation`… La tabla de equivalencias está en Novedades.
- **No se encuentra una imagen, un SVG o un modelo:** las rutas relativas se buscan en el directorio de trabajo, en la carpeta `assets/` del script y en la carpeta del script, salvo que cargues el proyecto con `scene.assets.load_project()` (busca el `gaanim.toml` más cercano, desde la carpeta del script hacia arriba) o fijes una carpeta absoluta con `scene.assets.assets_dir(str(Path(__file__).parent / "assets"))`.
- **`script did not submit a scene`:** el script debe terminar llamando a `scene.render()`.

54.3. La escena no se ve como esperas

- **Un objeto no aparece:** las líneas reactivas, los trazos (`traced_path`), los objetos con `follow` y los puntos sobre curvas empiezan ocultos; añade su `fade_in` o `create` a un `scene.play`.
- **Una animación no hace nada:** `.animate` solo describe el cambio. Pásalo a `scene.play([...])`.
- **Dos animaciones del mismo objeto chocan:** no puedes animar el mismo canal (por ejemplo, la posición) dos veces a la vez. El error nombra el objeto y las dos animaciones; combínalas en una o ponlas en secuencia.

- **Los objetos son enormes o diminutos:** las medidas están en unidades lógicas, no en píxeles. El fotograma mide 16 × 9; un círculo de radio `1` ocupa una octava parte del ancho. Un SVG se importa a 100 px por unidad (uno de 300 px mide 3 unidades); ajústalo con `scale_to`.
- **Todo sale en blanco:** una escena creada con `Scene(theme=None)` (o tras `scene.canvas.set_theme(None)`) no tiene tema: el fondo es blanco y el texto, las formas y los ejes sin color propio también son blancos. Quita `theme=None` para volver al tema `technical` predeterminado, elige otro tema o pon un fondo que contraste; `gaanim check` lo avisa.
- **Mis objetos tienen otro color del que esperaba:** las escenas usan por defecto el tema `technical`, que rellena las formas sin color con el acento y pinta texto y ejes en gris claro. Un `.fill(...)`, `.stroke(...)` o `color=` explícito siempre gana al tema.
- **Un texto o una tarjeta se salen de su sitio dentro de un layout:** no uses `move_to()` en objetos dentro de un layout; expresa la intención con `align`, `offset` o una restricción (ver Layout).
- **El movimiento cambia al saltar a otro instante:** alguna función de `computed`, `readout` o `add_updater_fn` guarda estado o lee el reloj. Consulta las reglas de Reactividad.
- **Una lectura numérica muestra `invalid`:** su función lanzó una excepción o devolvió un valor no finito en ese instante.

54.4. Recarga y vista previa

- **Guardé y no cambió nada:** comprueba que el archivo está dentro del proyecto y no en una carpeta ignorada (`exports`, `snapshots`, `target`, carpetas ocultas o de entornos). Los paquetes instalados fuera del proyecto no se recargan hasta reiniciar Gaanim.
- **La recarga tarda más de lo normal:** arranca con `GAANIM_RELOAD_PROFILE=1` para ver en la terminal el tiempo de cada segmento del script, las mediciones con `bounds()` que compilaron la escena y en qué línea, y qué segmentos tardan más en compilarse. Cuando medir compiló la escena a un coste notable, Gaanim lo avisa siempre en la terminal y en `gaanim check`. Los segmentos con funciones de Python (updaters, funciones reactivas, easings propios) se recompilan siempre. Para comparar con una recarga completa, arranca con `GAANIM_INCREMENTAL=0`.
- **Necesito los mensajes del motor:** Gaanim oculta los avisos de Bevy, wgpu, winit y el audio. Para verlos, arranca con `RUST_LOG=info` (o `RUST_LOG=wgpu=warn` para un solo módulo). `NO_COLOR=1` quita los colores; el logo solo aparece cuando la salida es una terminal.
- **La vista previa o la exportación fallan al iniciar la GPU:** prueba otro backend con `WGPU_BACKEND`, por ejemplo `WGPU_BACKEND=vulkan` o `WGPU_BACKEND=dx12` en Windows y `WGPU_BACKEND=gl` en Linux. Sin esa variable, la vista previa usa Vulkan (Metal en macOS) cuando hay una GPU que lo admite y, si no, prueba todos los backends.
- **Dos superficies 3D que se cruzan se ven mal ordenadas:** el 3D no tiene búfer de profundidad y ordena los triángulos por profundidad. Separa las superficies o subdivídelas (ver Cámara y 3D).

54.5. Exportar

- **La exportación dice que `ffmpeg` no está en el `PATH`:** instala FFmpeg y añádelo al `PATH`, o exporta una secuencia PNG (`--output frames/frame.png`), que no lo necesita.
- **Error por la proporción de la salida:** si `--width` y `--height` no tienen la proporción de la escena, añade `--fit contain` (con bandas) o `--fit cover` (recortando).
- **El video se detiene en las pausas o no lo hace:** la exportación ignora siempre `scene.stop(...)` y produce un video continuo; solo el modo presentación espera en ellas.

- **`--from` o `--to` fallan:** los valores deben ser segundos no negativos o nombres de `scene.marker(...)`, y `--to` debe ser mayor que `--from`.
- **Un fotograma no se pudo dibujar:** en escenas muy densas, Gaanim reintenta con más memoria y, si no basta, falla indicando el fotograma. Prueba una resolución menor o reduce la cantidad de objetos translúcidos.
- **Transparencia:** `--transparent` solo funciona con WebM, MOV, WebP y PNG, y la escena necesita un fondo transparente. Para un editor de vídeo, usa MOV (ProRes 4444): el alfa de WebM se lee mal fuera del navegador.

54.6. Capturas y comparación

- **`--capture-stops` dice que no hay nada que comparar:** la versión aprobada no tiene `stops.json`. Aprueba una captura de pausas con `--capture-stops --bless`.
- **Todas las capturas cambian por diferencias mínimas:** ajusta `--pixel-threshold` y `--max-changed-ratio` (ver Capturas y comparación visual).

Si tu problema no aparece aquí, abre un aviso en el repositorio de Gaanim con el mensaje de error completo y el script mínimo que lo reproduce.