

# Informática Gráfica.

## Sesión 11: Animación.

Carlos Ureña, Sept 2025.  
Dept. Lenguajes y Sistemas Informáticos.  
Universidad de Granada.

### Índice

---

La animación por ordenador. Ejemplos sencillos. ....	3
Técnicas de animación .....	49
Interpolación y curvas para animación .....	62
Animaciones en Godot .....	97

1. La animación por ordenador. Ejemplos sencillos..

### Resumen de la sección

---

Sección 1.

#### La animación por ordenador. Ejemplos sencillos.

1. La animación clásica
2. Animación por ordenador
3. Estructura de los sistemas de animación
4. Implementación de sistemas de animación interactiva.
5. Ejemplos de animaciones sencillas.

En esta sección se realiza una introducción a la animación y la animación por ordenador

- Una breve introducción a la animación clásica: dispositivos mecánicos y cine de animación tradicional.
- Conceptos básicos de la animación por ordenador. Tipos de animación: *off-line* e *interactiva*

## La Animación clásica

La palabra **Animación** (en sentido clásico) se refiere al proceso de darle apariencia de movimiento a objetos estáticos.

- Desde tiempos muy antiguos se ha intentado desarrollar dispositivos o mecanismos para lograr esto.
- La idea básica es presentar una secuencia de imágenes impresas (en papel o celuloide) a un ritmo que produzca *ilusión* de movimiento continuo.
- Los primeros dispositivos (diseñados a principios del siglo XIX) usan un efecto *estroboscópico*: el *Zootropo* (*Zoetrope*), y múltiples variantes:
- Con la aparición del cine, se crea la animación como una industria y un arte: se dibuja cada imagen y se fotografía sobre una película de celuloide.

### Subsección 1.1.

#### La animación clásica

1. La animación por ordenador. Ejemplos sencillos..  
1.1. La animación clásica.

### Dispositivos mecánicos para animación

El efecto estroboscópico consiste en iluminar la imagen del sujeto en una posición fija del disco a intervalos regulares de tiempo, de forma que se incrementa la sensación de movimiento.

(no se percibe el movimiento del disco, sino la secuencia de imágenes)



GIF animado de: *Simon Ritter von Stampfer*, Public domain, via Wikimedia Commons:  
[commons.wikimedia.org/wiki/File:Prof.Stampfer%27s\\_Stroboscopische\\_Scheibe\\_No.X.gif](https://commons.wikimedia.org/wiki/File:Prof.Stampfer%27s_Stroboscopische_Scheibe_No.X.gif)

Sesión 11: Animación

Created 2025-12-09

Page 6 / 112.

### Cine de animación clásico

A lo largo del siglo XX se desarrollan técnicas de animación clásica (no por ordenador) y *stop motion*

- Se comienza dibujando cada cuadro a mano de forma independiente, se hace una fotografía de cada cuadro (se crea un fotograma de la película).
- En *stop motion* se crea un escenario real estático que se cambia incrementalmente para cada fotografía.
- Despues se desarrolla el *keyframing / inbetweening*: unos artistas dibujan unos cuadros clave y otros interpolan los cuadros intermedios.
- Al principio cada cuadro era una imagen era única.
- Despues se forma cada cuadro superponiendo varias *capas* de dibujo, cada una de ellas dibujada en un soporte transparente.

## Cine de animación clásico: keyframes e in-betweens

Ejemplo de tres keyframes y los in-betweens:

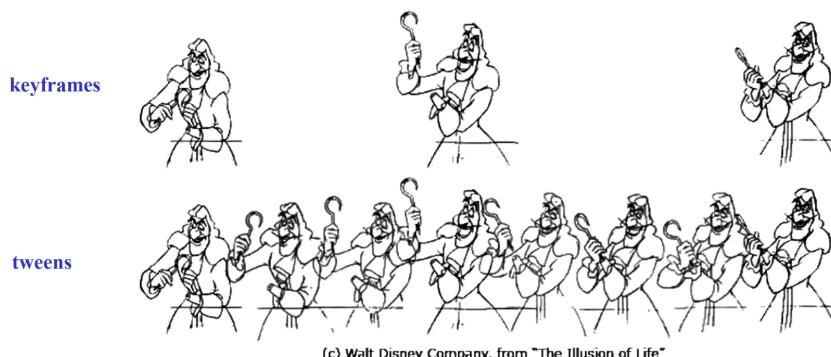


Imagen del libro de Frank Thomas y Ollie Johnston: **The Illusion of Life** (1981):  
<https://books.disney.com/book/the-illusion-of-life/>

Sesión 11: Animación

Created 2025-12-09

Page 9 / 112.

1. La animación por ordenador. Ejemplos sencillos..  
1.1. La animación clásica.

## Cine de animación clásico: cámaras multiplano

Este tipo de cámaras tienen varios grados de libertad que permitían mover las capas de dibujo de forma independiente:

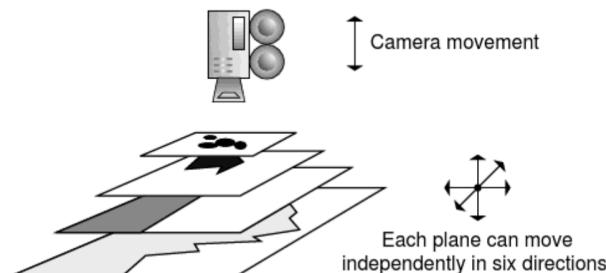


Imagen del libro: Rick Parent **Computer Animation Algorithms and Techniques** (3rd Edition)  
Ed. Elsevier. 2012. ISBN: 9780124158429.

## Cine de animación clásico: cámaras multiplano

Los cuadros de una animación pueden estar compuestos de varias capas, para eso se diseñaron las **cámaras multiplano**:

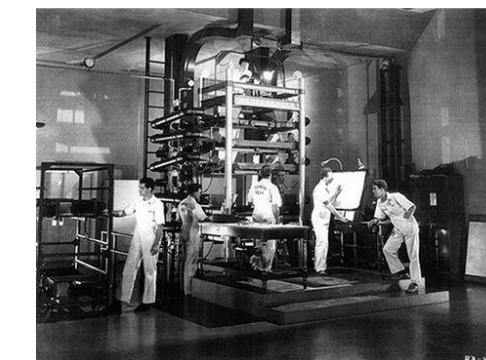


Imagen de The Walt Disney Family Museum Blog:  
[waltdisney.org/blog/machine-imagination-walt-disneys-pinocchio-and-multiplane-camera](http://waltdisney.org/blog/machine-imagination-walt-disneys-pinocchio-and-multiplane-camera)

Sesión 11: Animación

Created 2025-12-09

Page 10 / 112.

---

### Subsección 1.2. Animación por ordenador

---

## Animación por ordenador

A partir de los años 70, se comienza a dar soporte para la animación usando ordenadores, de forma progresivamente mayor hasta la actualidad, donde los ordenadores son una herramienta esencial para las animaciones. El ordenador se usa para:

- Modelado de personajes, escenarios y elementos 2D o 3D.
- Iluminación y texturas realistas.
- Definición de la dinámica y comportamiento de los objetos de la escena.
- Render de los fotogramas.
- Mezcla de imagen real fotografiada o filmada con elementos generados por ordenador.
- Composición de vídeo y efectos post-render.

### Eventos. Tipos.

En el contexto de la animación, un **evento** es un cambio de estado relevante, de varios tipos:

**Eventos de entrada:** como los que ya hemos estudiado, son eventos producidos directa o indirectamente por el usuario usando algún dispositivo, de uno de estos dos tipos:

- **Físico:** teclado, joystick, ratón, etc...
- **Virtual:** un botón u otros elementos de un interfaz de usuario. También los hemos llamado **señales**

**Eventos de la escena:** eventos que resultan de la interacción de los elementos de la escena entre ellos, p.ej.:

- La trayectoria de un objeto alcanza una pared.
- Un coche supera la línea de meta.
- Un NPC (*personaje no jugable*) ha entrado en una habitación.
- El personaje del jugador ha perdido toda su energía.

## Animación por ordenador: terminología

Usaremos los siguientes términos:

- **Cuadro o fotograma** (o **frame**): cada una de las imágenes estáticas que componen una animación.
- **Fotogramas por segundo** (o **frame rate, FPS**): número de fotogramas por unidad de tiempo real (usualmente segundos) a la que se reproduce la animación, puede ser constante o variar. A veces es un objetivo que no siempre se consigue en todos los *frames*.
- **Período (T)**: tiempo (usualmente en milisegundos) entre fotogramas consecutivos (es 1000/FPS).
- **Período objetivo**: período que se quiere lograr en una animación.
- **Tiempo**: usamos este término para referirnos, durante la reproducción de una animación, al **tiempo real** transcurrido desde su inicio o desde un instante de referencia en la misma.

### Animación por ordenador no interactiva (*off line*)

En la animación *off line* usada en cinematografía no hay interacción con el usuario durante el proceso de render de las imágenes.

- Se planean con antelación las animaciones de cada elemento de cada secuencia de animación.
- Se decide de antemano un *framerate* fijo (normalmente entre 24 y 60 FPS).
- Cada fotograma tiene asociado un instante de tiempo fijo y conocido.
- Cada fotograma producido es almacenado en el sistema de archivos.
- El tiempo que se tarda en sintetizar cada cuadro no está relacionado con el *frame rate* (puede ser muy superior al período).

## Animación por ordenador interactiva (*on line*)

En la animación *interactiva* (típicamente usada en videojuegos y simuladores), el usuario interactúa con el sistema durante la reproducción de la animación, influyendo en la misma:

- El usuario produce *eventos de entrada* (teclado, ratón, joystick, etc..)
- Se define como cada elemento reacciona a los posibles eventos y como evoluciona con el tiempo.
- Cada fotograma producido es mostrado en un monitor.
- Se elige un *frame rate* mínimo (un período máximo) como objetivo. A veces puede no lograrse.
- Por tanto: el tiempo que se tarda en sintetizar cada cuadro está, en principio, acotado por el período objetivo (en un sistema monohebra).

---

### Subsección 1.3. Estructura de los sistemas de animación

---

## El bucle de animación

La producción de una animación *off-line* o de una animación interactiva supone la ejecución de un **bucle de animación**

- En animaciones interactivas, se ejecuta hasta que se decide parar la animación o acabar el programa.
- En animaciones *off line* se ejecuta hasta que se han visualizado todos los *frames* que se planeaban producir.
- En cada iteración:
  - ▶ Se actualiza el estado del modelo de escena (ya que cambia con el tiempo).
  - ▶ Se produce una imagen.
- La actualización del estado puede suponer cálculos complejos: iluminación avanzada, física, IA de personajes, post-procesamiento de imagen, etc...)

## Modelo de escena en animación

El modelo de escena debe incluir (igual que en IG en general):

- **Modelos geométricos:** mallas de triángulos, árbol de escena, transformaciones, etc...
- **Modelos de aspecto:** texturas, materiales, luces, etc...

Además, en animación se debe añadir a dicho modelo:

- **Modelos dinámicos:** cómo evolucionan los modelos geométricos o de aspecto de los objetos de la escena con el tiempo, p.ej. qué trayectoria seguirá un objeto.
- **Modelos de comportamiento:** para animación interactiva, hay que definir cómo reaccionará cada elemento a los posibles eventos que pueden ocurrir, p.ej.: qué hará el personaje jugador cuando el usuario pulsa una tecla determinada, o cuando atraviesa una puerta.

## Estado de los objetos

En animación interactiva cada objeto u elemento visible tiene un estado que puede depender en todo o en parte del tiempo. El estado puede tener distintas componentes:

- **Estáticas:** independiente del tiempo (constantes) (p.ej: la geometría en coordenadas maestras y la textura de una bala de cañón).
- **Dinámicas y continuas:** variables continuas cuyo valor es una función no constante del tiempo (p.ej: la posición, velocidad, aceleración y orientación de la bala, las cuales son funciones del tiempo transcurrido desde que se disparó).
- **Dinámicas y discretas:** variables que toman un valor en un conjunto discreto de valores (p.ej: la bala puede estar en estado de reposo, de movimiento o explotando). Los cambios del valor se llaman **transiciones** y se suelen modelar con **máquinas de estados**

## Cambio y evolución del estado en los objetos

Cada objeto dinámico de una escena animada debe de incorporar métodos para:

- **Actualizar el estado:** calcular el nuevo estado del objeto para un frame que se va a sintetizar, a partir del estado actual y del instante de tiempo  $t$  de dicho frame.
- **Procesar evento** ejecuta una posible transición tras un evento, es decir: calcula el nuevo estado del objeto a partir de un evento de entrada que ha ocurrido en un instante de tiempo. Se conocen:
  - ▶ El tipo y atributos del evento (p.ej: la posición del ratón, etc...)
  - ▶ A veces, el instante de tiempo real en el que se produjo el evento.

## El bucle de animación interactiva

El esquema en pseudo-código del bucle de animación interactiva es:

```
1: while se esté ejecutando la animación do
2:   t ← instante de tiempo real actual
3:   for each evento  $E$  pendiente de procesar do
4:     for each objeto  $O$  afectado por  $E$  do
5:       Hacer que  $O$  procese el evento  $E$ .
6:     end
7:   end
8:   for each objeto dinámico  $O$  de la escena do
9:     Actualizar el estado de  $O$  al instante  $t$ .
10:  end
11:  Visualizar la escena sobre el framebuffer actual.
12:  Presentar el framebuffer actual en pantalla.
13:  Esperar bloqueado hasta  $t + T$            ▷ (si sobra tiempo)
14: end
```

## Sistemas de animación y tiempo real

Los sistemas de animación tienen algunas características de *sistemas de tiempo real*:

- Un *sistema de tiempo real (STR)* es un software que tiene requerimientos no funcionales estrictos en cuanto a los tiempos de respuestas de sus componentes
- Por ejemplo: el sistema que detecta una colisión y despliega un airbag de un coche debe hacer eso en un tiempo limitado, del orden de milisegundos, es un **STR estricto**
- En animación interactiva, el *frame rate* es un objetivo que se intenta alcanzar, pero no siempre se consigue. Por eso se puede considerar un sistema de tiempo real **no estricto**.

## Herramientas de tiempo real

Se requieren herramientas software (librerías o soporte del lenguaje) de tiempo real para

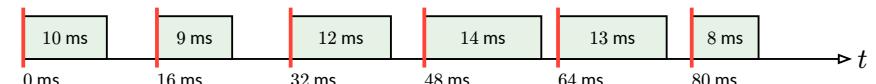
- Ser capaz de medir el tiempo transcurrido con precisión, para
  - ▶ conocer el instante de tiempo real de cada fotograma que se va a producir, y para
  - ▶ conocer el tiempo sobrante o de retraso hasta el siguiente fotograma.
- Ser capaz de esperar hasta el fin del período entre dos frames, se puede hacer dos formas alternativas:
  - ▶ hacer una espera bloqueada explícita (en aplicaciones de escritorio), o bien
  - ▶ requerir que se ejecute una función pasado un determinado período de tiempo (en aplicaciones Web).

Todos estos aspectos suelen ser transparentes al programador en **engines de videojuegos** como Godot, Unity, Unreal Engine, etc...

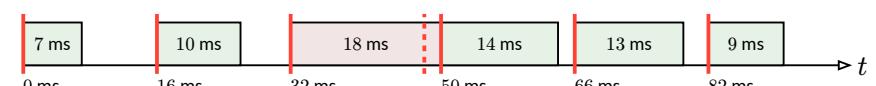
## Diagrama de una animación

Suponiendo un período  $T = 16$  milisegundos (ms), cada caja representa la actualización de estado y el render de un frame (tarda un tiempo posiblemente distinto en cada frame)

Todos los frames se pueden calcular dentro del período:

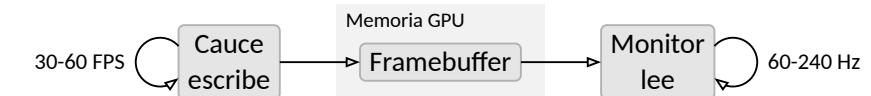


Un frame (en rojo) tarda más del período, el siguiente se retrasa:



## La tasa de refresco del monitor

En un esquema sencillo, el cauce escribe en un *framebuffer* (memoria de imagen), el cual a su vez está conectado al ordenador:



- El cauce escribe periódicamente en el *framebuffer* (30-60 FPS).
- El monitor está continuamente leyendo la imagen del *framebuffer* y presentándola en pantalla
- La **tasa de refresco del monitor (monitor refresh rate)** es el número de veces por segundo que eso ocurre (entre 60 y 240 Hz)
- No tiene sentido usar para una animación interactiva un *frame rate* superior a la tasa de refresco del monitor (algunos frames producidos no se verán en pantalla).

## Parpadeo debido a framebuffer único

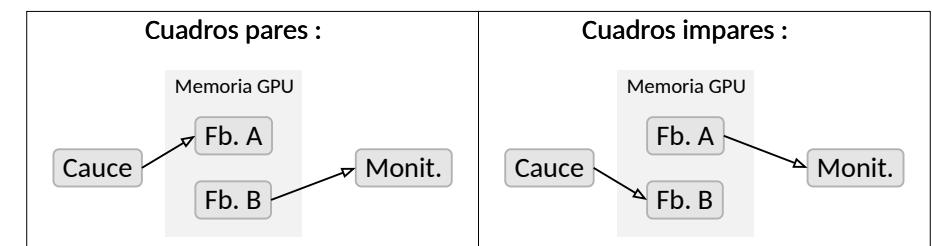
Si se usa un único framebuffer (como en el ejemplo de arriba)

- En una animación, el cauce no está sincronizado con el refresco del monitor (el *framerate* de la animación es distinto del *refresh rate* del monitor).
- Por tanto, ocurrirá que el refresco del monitor se realiza cuando el cauce está a mitad de visualizar una escena en el *framebuffer*.
- La imagen que se presenta en el monitor será incompleta (solo se visualizan algunos triángulos de la escena).
- Al efecto visible se le llama en general **flickering (parpadeo)**

## Framebuffer doble para animación interactiva

Para evitar problemas se usa un *doble buffer*: consiste en usar al menos dos *framebuffers* en lugar de uno solo:

- Uno de ellos se muestra en el monitor mientras el otro se actualiza.
- Cada vez que se termina de renderizar un frame, los roles de ambos buffers se intercambian (se hace *swap* de los buffers).
- Los frames mostrados en el monitor están siempre completos.



---

### Subsección 1.5.

#### Ejemplos de animaciones sencillas.

---

## Animación de las agujas de un reloj

Supongamos que queremos visualizar una animación de un reloj con tres agujas: una para las horas, otra para los minutos y otra para los segundos.

- Tenemos una malla de polígonos que es un modelo de una aguja en posición vertical (paralelo al eje Y, hacia arriba), con el origen en el punto del eje del reloj.
- Para visualizar las tres agujas usamos matrices de rotación entorno el eje Z, cada una es una rotación distinta.
- Cada una de esas matrices de rotación depende del tiempo  $t$ , que es el tiempo en segundos transcurrido desde el comienzo del un día determinado.
- Lógicamente, cada una de esas matrices usa un ángulo que depende de  $t$ , pero con una constante de proporcionalidad distinta.

## La animación de tres agujas:

Llamamos  $\theta_h$ ,  $\theta_m$  y  $\theta_s$  a los ángulos de las tres agujas en un instante  $t$  (es un tiempo conocido en segundos)

- Las matrices de rotación serán  $R_z(\theta_h)$ ,  $R_z(\theta_m)$  y  $R_z(\theta_s)$ , donde, en general, para un ángulo  $\gamma$  cualquiera (en radianes) la expresión  $R_z(\gamma)$  representa la matriz de rotación entorno al eje Z de un ángulo  $\gamma$  (en el sentido de las agujas del reloj si  $\gamma > 0$  y lo vemos desde Z+). Por tanto:

$$R_z(\gamma) = \begin{pmatrix} c & -s & 0 & 0 \\ s & c & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad \text{donde: } \begin{cases} c = \cos(\gamma) \\ s = \sin(\gamma) \end{cases}$$

## Cálculo de los ángulos

Los tres ángulos  $\theta_h$ ,  $\theta_m$  y  $\theta_s$  dependen linealmente de  $t$  (en segundos):

- El segundero da una vuelta completa ( $2\pi$  radianes) cada minuto (60 segundos):

$$\theta_s = \frac{2\pi}{60} \cdot t$$

- El minutero da una vuelta cada hora ( $= 60^2$  segundos):

$$\theta_m = \frac{2\pi}{60^2} \cdot t$$

- Las horas dan una vuelta cada 12 horas ( $= 12 \cdot 60^2$  segundos):

$$\theta_h = \frac{2\pi}{12 \cdot 60^2} \cdot t$$

## Modelo del reloj

Por tanto, modelamos el reloj usando un grafo de escena en el cual el nodo raíz es un objeto compuesto que contiene:

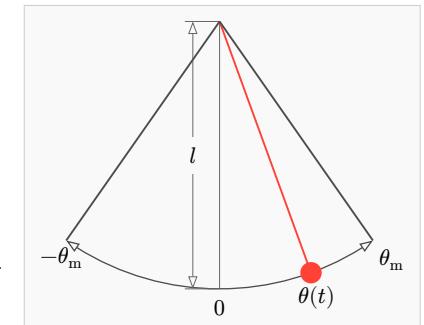
- un nodo para la esfera (que no cambia con el tiempo), con centro en el origen.
- un nodo para cada aguja, cada uno de ellos contiene
  - la matriz de transformación de la aguja.
  - un nodo hijo (el modelo de la aguja en coordenadas maestras).

La función para actualizar el estado simplemente asigna las matrices de rotación según el valor de  $t$  recibido, usando las fórmulas que hemos visto antes.

## Modelo del péndulo

Supongamos el modelo de un péndulo: es un disco con cierta masa colgando de un punto fijo por una cuerda de longitud  $l$ . El ángulo  $\theta$  entre la cuerda y la vertical varía con el tiempo  $t$ , es una función  $\theta(t)$  del tiempo  $t$  (en segundos).

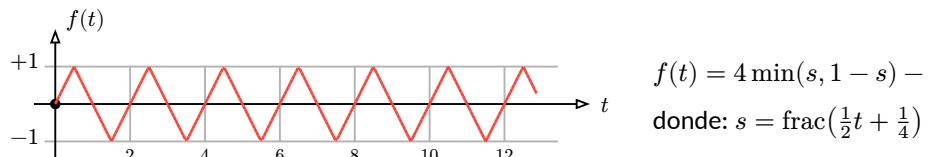
- El péndulo oscila periódicamente desde un ángulo  $-\theta_m$  hasta el ángulo  $\theta_m$
- Esa oscilación tiene un período  $T > 0$  expresado en segundos, es decir: se tardan  $T$  segundos en volver a pasar por el mismo sitio con la misma dirección.



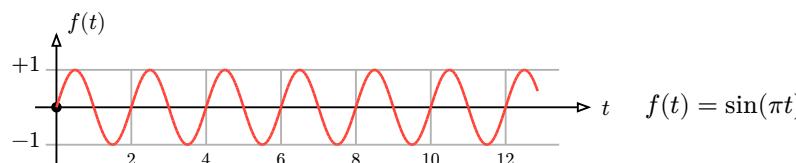
## Función oscilante auxiliar

Necesitamos una función  $f(t)$  que oscile entre  $-1$  y  $+1$  con un período de 2 unidades de tiempo.

Una posibilidad es una función **lineal a trozos** (cambios bruscos)

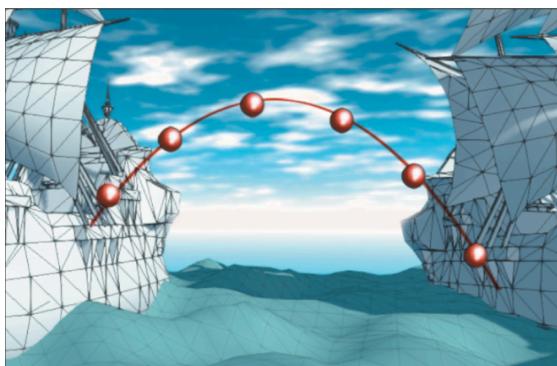


Para lograr un movimiento más suave, usamos una **sinusoidal**



## Ejemplo de la bala de cañón

Supongamos que queremos simular la trayectoria de una bala de cañón esférica que es lanzada desde un punto (se eleva hasta que vuelve a la altura inicial)



## Evolución del ángulo $\theta$ con el tiempo.

Así que ya podemos usar la función  $f$  para modelar como evoluciona el ángulo  $\theta$  con el tiempo:

$$\theta(t) = \theta_m \cdot f(2t/T)$$

Donde:

- La amplitud de la oscilación es  $2 \cdot \theta_{\max}$  en lugar de 2
- El período es  $T$  en lugar de 2.

El modelo del péndulo será un modelo jerárquico que incluye una matriz de rotación entorno al eje Z,  $R_z(\theta)$ , con ángulo  $\theta$  (respecto de la vertical).

## Supuestos

Suponemos las siguientes condiciones:

1. La bola sale del cañón en una posición inicial y con una velocidad inicial conocidas.
2. La bola está sujeta a la gravedad.
3. La bola es esférica y de un color plano (la orientación o rotación es irrelevante)
4. La bola no tiene efectos de fricción con el aire
5. Nos interesa simular la trayectoria hasta que alcanza de nuevo la altura inicial.

En estas condiciones, la trayectoria de la bala **solo depende de la velocidad y dirección iniciales**.

## Trayectoria con aceleración constante

Las condiciones que determinan la trayectoria son:

- La bala tiene (en el instante inicial  $t = 0$ ) su centro en una posición  $\mathbf{p}(0) = (p_x, p_y, p_z)$  y una velocidad inicial  $\mathbf{v}_0 = (v_x, v_y, v_z)$ , ambos son dos vectores en 3D, conocidos, con unidades de metros (m) y metros por segundo (m/s), respectivamente. Se cumple  $v_y > 0$  (se lanza hacia arriba).
- La aceleración (debida a la gravedad) es un vector  $\mathbf{a} = (0, -g, 0)$  constante en el tiempo, donde  $g = 9.8$  (en m/s<sup>2</sup>).

En estas condiciones, la posición (en metros) de la bala en un instante cualquiera  $t$  es  $\mathbf{p}(t)$ , definido como:

$$\mathbf{p}(t) = \mathbf{p}(0) + \mathbf{v}_0 t + \frac{1}{2} \mathbf{a} t^2$$

## Implementación de la trayectoria

La función  $\mathbf{p}(t)$  es un ejemplo de una **curva paramétrica** en el espacio 3D (una secuencia continua de puntos que depende de un valor real, en este caso el tiempo).

Para visualizar la animación se usa:

- Un modelo en coordenadas maestras de la bala,
- Una matriz de traslación  $T_{\mathbf{p}(t)}$ , que traslada el centro del modelo a la posición  $\mathbf{p}(t)$  cada vez que se pide una actualización del modelo para el instante  $t$ .

La duración de la animación en segundos es

$$t = \frac{2v_y}{g}$$

(es el tiempo que tarda la bala en volver a la altura inicial  $p_y$ ).

## Ejemplo de un perseguidor

Las anteriores animaciones se basan en modificar un ángulo o una posición usando una expresión para una matriz que depende del tiempo según una función simple conocida.

- No se requieren variables de estado ( adicionales a la posición, el ángulo y algunas constantes conocidas).
- En muchas animaciones más complejas, se requieren variables de estado (discretas o continuas) para modelar el comportamiento.

Un ejemplo sería un modelo de un **perseguidor**

- Es un objeto que está en una posición pero se desplaza hacia una posición objetivo.
- Puede sufrir cambios de estado en respuesta a eventos externos: pausar, reanudar, fijar un nuevo objetivo.

## Condiciones del comportamiento

El comportamiento del perseguidor es este:

- Cuando el perseguidor está pausado, no realiza ningún movimiento.
- Cuando el perseguidor no está pausado, se desplaza en línea recta hacia el objetivo. Lo hará a velocidad constante (1m/s), hasta que su posición se encuentre sobre el objetivo (o se fije otro objetivo).
- Cuando el perseguidor no está pausado pero su posición ya coincide con la del objetivo, no se mueve.
- Los eventos externos producen cambios de estado, cada uno de esos eventos tiene asociado un instante en el tiempo. Se implementan como funciones que cambian el estado del perseguidor, las cuales reciben un valor de tiempo (nunca inferior al de la llamada previa).

## Variables de estado del perseguidor

Las diversas variables de estado que se necesitan para implementar su comportamiento son:

**Estado:** una variable discreta que solo puede tomar dos valores: *persiguiendo*, y *pausado*. Inicialmente es *pausado*.

**Posición:** (*p*) posición actual en el espacio del objeto (un vector). Inicialmente está en el origen.

**Objetivo:** (*o*) posición del objetivo en el espacio (un vector). Inicialmente es el origen.

**Último instante:** (*u*) instante en el que se actualizó por última vez la posición del objeto, por tanto es el instante al que corresponde la posición. Inicialmente es 0.

## Eventos que producen cambios de las variables

Los eventos u operaciones que producen cambios de alguna de las variables de estado tienen todas asociado un instante *t*, son:

**Actualizar estado:** se llama (al menos) antes de visualizar un nuevo *frame*, para llevar la posición a la correspondiente al instante *t*

**Fijar objetivo:** supone cambiar la variable *q* por un nuevo valor y pasar al estado *persiguiendo* (si no estaba ya en ese estado).

**Pausar:** si el estado no es *pausado*, cambia el estado a *pausado*, a partir de ese instante el perseguidor no cambia su posición (pero mantiene el objetivo que tenía)

**Reanudar:** si el estado no es *persiguiendo*, cambia el estado a *persiguiendo*, a partir de ese instante su posición puede cambiar con el tiempo.

## Actualización de estado

La función de actualización recibe un instante *t* y actualiza la posición *p*:

```
1: function ACTUALIZARESTADO(t)
2:   if estado == persiguiendo then
3:     d ← o - p                         ▷ d es el vector hasta el objetivo
4:     if \|d\| > 0 then
5:       r ← min(t - u, \|d\|)             ▷ r es el tiempo en mov.
6:       p ← p + rd/\|d\|                ▷ actualizar posición
7:     end
8:   end
9:   u ← t
10: end
```

Nota: el valor *r* es el tiempo en movimiento (desde la última actualización previa) hasta llegar al objetivo o hasta esta actualización (lo que ocurra antes). Se asume velocidad unidad.

## Implementación de eventos

La implementación de las operaciones es:

```
1: function FIJAROBJETIVO(t, q)
2:   ACTUALIZARESTADO(t)
3:   o ← q
4: end
```

```
1: function PAUSAR(t)
2:   ACTUALIZARESTADO(t)
3:   estado ← pausado
4: end
```

```
1: function REANUDAR(t)
2:   ACTUALIZARESTADO(t)
3:   estado ← persiguiendo
4: end
```

## Resumen de la sección

En esta sección se abordan diversas técnicas de animación muy usadas en la producción de infografías y desarrollo de videojuegos:

### **Keyframes e Inbetweens:** (cuadros clave y cuadros intermedios)

Esta técnica agiliza el diseño e implementación de las animaciones, ya que reduce la complejidad de estos procesos (reduce el número de cuadros que se diseñan manualmente)

### **Rigging y Skinning:** (esqueleto y piel)

También facilita el diseño de objetos complejos (como personas o animales), ya que evita el tener que adaptar manualmente el modelo de la piel o ropa a las distintas poses de un modelo articulado.

### Sección 2.

## Técnicas de animación

1. Keyframes e Inbetweens
2. Rigging y Skinning

Sesión 11: Animación

Created 2025-12-09

Page 50 / 112.

2. Técnicas de animación.

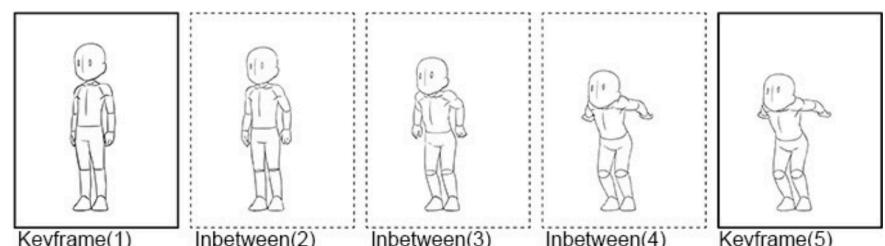
2.1. Keyframes e Inbetweens.

### **Keyframes e Inbetweens**

La técnica de *keyframing* permite una mayor agilidad en la producción de animaciones, ya que separa la tarea de definir unos **cuadros clave (keyframes)** de la tarea de producir los frames **intermedios** entre ellos (**inbetweens**)

#### Subsección 2.1.

### Keyframes e Inbetweens



## Cuadros clave para animación de grafos de escena

Para animaciones por ordenador de grafos de escena parametrizados, se puede usar la metodología de los **cuadros clave**:

- Cada cuadro clave tiene asociada un valor de tiempo ( $t_j$ )
- El diseñador o animador especifica el vector de valores de los parámetros en cada cuadros clave. Por tanto, conocemos el valor  $v_{i,j}$  del parámetro  $i$  en el cuadro clave número  $j$
- Para los cuadros intermedios (entre cuadros clave) se hace **interpolación**: cada parámetro se interpola entre el valor de ese parámetro en el cuadro clave anterior y el posterior.

## Edición de las curvas de interpolación

Aquí vemos una captura del *editor de curvas* de 3Ds Max (Autodesk). El usuario está editando 6 curvas correspondientes a 6 parámetros:

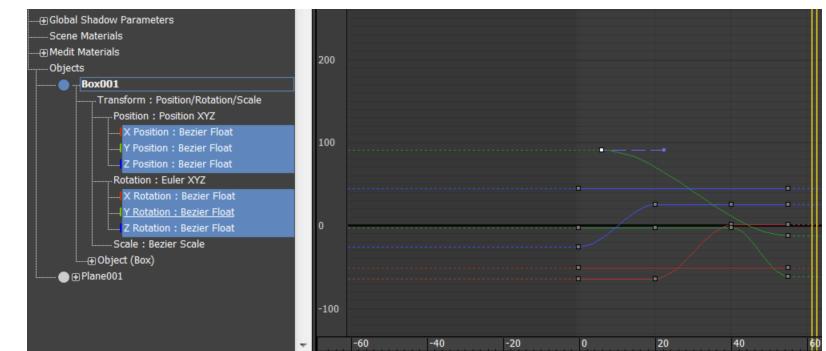


Figura obtenida de la página [Curve Editor Introduction](#) en el sitio Web [Autodesk 3Ds Max Learning Center](#) (Abril 2024).

## Edición de las curvas de interpolación

También es posible: (a) controlar las derivadas, y (b) que los instantes clave  $t_i$  de cada parámetro sean distintos:

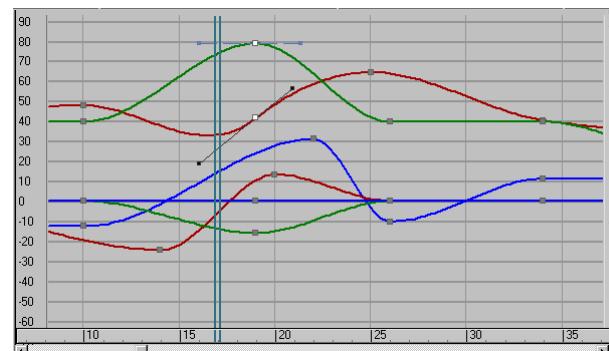


Figura obtenida de la página [Track View Workspace](#) en el sitio Web [Autodesk 3Ds Max Learning Center](#) (Abril 2024).

---

Subsección 2.2.  
**Rigging y Skinning**

---

## Modelado de personajes

Si queremos modelar mallas de triángulos que representen personajes o animales:

- Se puede usar un grafo de escena jerárquico parametrizado, hecho de partes rígidas: cabeza, torso, muslo, pierna, pie, antebrazo, brazo, manos, los dedos, falanges, etc...
- Cada elemento es un nodo del grafo que se puede rotar respecto a la articulación que lo une al padre. El vector de parámetros son los ángulos de esas articulaciones (permite definir poses e interpolar entre ellas).
- El problema es modelar de forma realista la piel o la ropa, que se extienden de forma continua entre distintas partes y se *adaptan* a las mismas: es poco realista modelarlo como una malla rígida.

## Rigging y Skinning: Rigging

La solución consiste en separar el problema en dos partes, llamadas *Rigging* y *Skinning*

**Rigging:** se define un grafo de escena parametrizado (*esqueleto*)

- Está hecho de segmentos de recta rígidos (*huesos*), con extremos en las articulaciones
- El modelo tiene asociados  $n$  parámetros (típicamente ángulos de rotación, u otros)
- Cada hueso tiene una matriz de transformación asociada.
- Este esqueleto se usa para definir las poses del modelo.

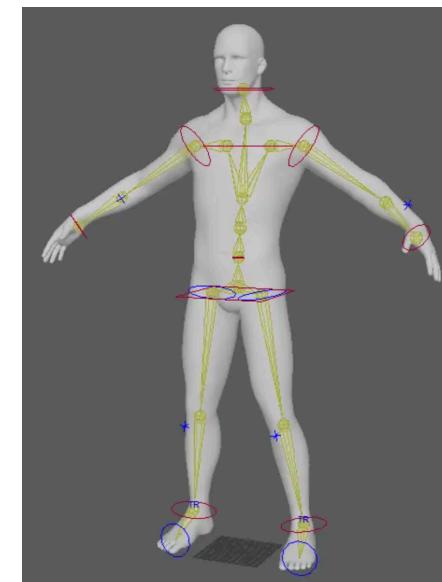
## Rigging y Skinning: Skinning

Una vez está definido el *rig* (esqueleto), se procede al *skinning*

**Skinning:** se define una malla indexada de triángulos (*skin*, piel) que se adapta al modelo en una pose *neutra*.

- Cada vértice de la malla se asocia a uno o varios de los huesos (usando un vector de pesos), según su distancia a los mismos.
- Para una pose arbitraria (distinta de la neutra), la matriz de transformación de cada vértice de la piel es una suma ponderada de las matrices de transformación de los huesos asociados a dicho vértice.

## Ejemplo de un esqueleto (*Rig*) y su piel (*Skin*)

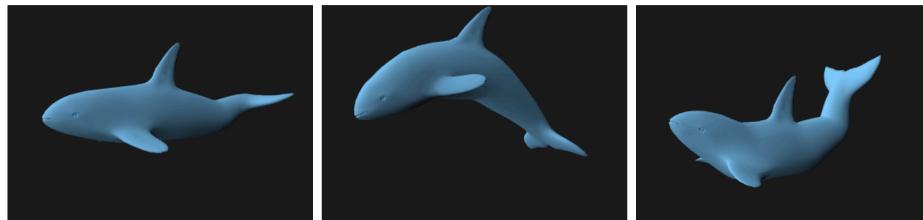


Se observa el modelo jerárquico (segmentos) del esqueleto, así como la malla de triángulos que representa la piel del modelo. En la página se observa una animación interactiva del modelo.

Figura obtenida de la página  
[Rigging and Skinning](https://gamedevinsider.com/) en el sitio Web  
<https://gamedevinsider.com/> (Abril 2024).

## Implementación en un vertex shader

Aquí podemos ver una implementación sencilla del *rigging* basado en vectores de pesos de cada vértice, en un *vertex shader* con WebGL, para un modelo simple:



Página con descripción de la técnica y la animación en **WebGL Fundamentals**:

<https://webglfundamentals.org/webgl/lessons/webgl-skinnning.html>

Enlace directo a la animación:

<https://webglfundamentals.org/webgl/webgl-skinnning-3d-gltf-skinned.html>

---

### Sección 3.

#### Interpolación y curvas para animación

---

1. Interpolación de valores reales.
2. Curvas paramétricas.
3. Curvas e interpolación en 2D/3D.

## Resumen de la sección

En esta sección se exponen las metodologías para implementar funciones reales de variable real que interpolan entre dos o más valores reales, dados como datos de entrada, pudiéndose controlar la forma de la función fijando sus derivadas

Esta técnica es útil para el diseño de animaciones usando funciones o curvas paramétricas, que permiten, entre otras cosas, realizar interpolación de variables continuas (especialmente para *in betweening*) y diseñar trayectorias de movimiento, todo ello de forma flexible y eficiente.

---

### Subsección 3.1.

#### Interpolación de valores reales.

---

## El problema de la interpolación

Queremos reproducir una animación entre dos instantes de tiempo, de forma que a lo largo del intervalo entre ambos, una variable  $v$  tome un valor dado al inicio de la animación y otro distinto al final. La variable puede ser de varios tipos:

**Valor real:** por ejemplo: una posición en una línea, un ángulo, etc...

**Vector de reales:** por ejemplo: una posición en 2D o en 3D, o una velocidad, etc...

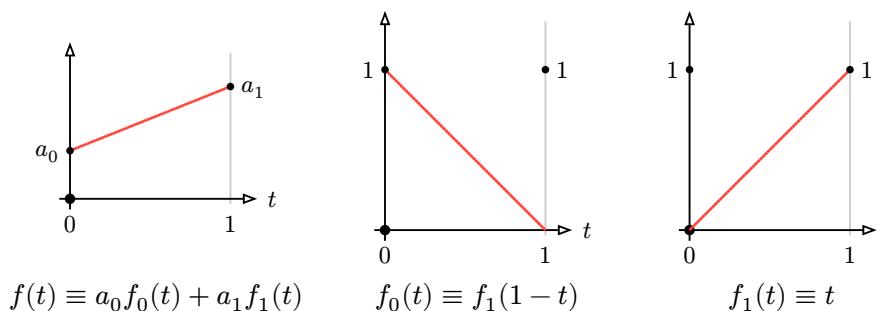
**Matriz de reales:** por ejemplo: una transformación geométrica, una matriz de cámara, etc...

Por simplicidad, inicialmente consideramos que:

el intervalo de tiempo va de 0 a 1 y que la variable es un valor real

## Función lineal.

Una posibilidad muy sencilla es usar una función  $f$  **lineal**:



El problema es que la pendiente es constante, pero nos gustaría usar una curva cuya pendiente vaya tiendiendo a cero en los extremos (para una entrada y salida suaves). Por tanto queremos que  $f'(0) = f'(1) = 1$ .

## Interpolación en el intervalo unidad

Por tanto, tenemos el siguiente problema de interpolación:

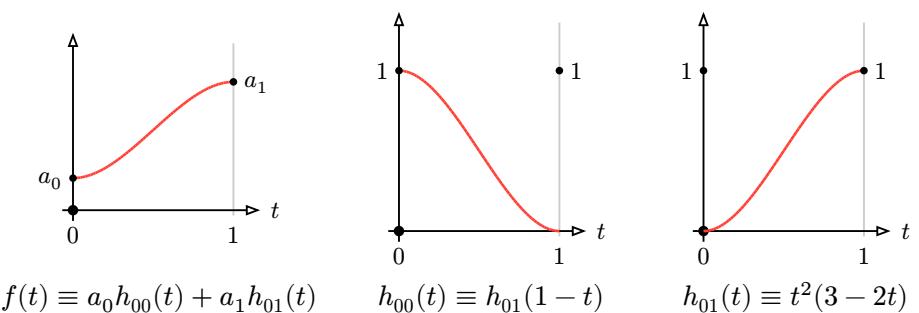
- El instante inicial es  $t = 0$  y el final es  $t = 1$ .
- El valor inicial de la variable  $v$  es  $a_0$  y el final es  $a_1$ , ambos son datos conocidos de entrada (pueden ser positivos o negativos).
- Queremos encontrar una función  $f$  tal que:
  - ▶  $f$  tiene un parámetro real  $t$ , siempre en el intervalo  $[0, 1]$
  - ▶  $f$  produce valores reales (positivos o negativos).
  - ▶ Se cumple  $f(0) = a_0$  y  $f(1) = a_1$

Cualquier función que usemos para interpolar entre  $a_0$  y  $a_1$  se puede poner como la suma ponderada de dos funciones  $f_0$  y  $f_1$ :

$$f(t) \equiv a_0 f_0(t) + a_1 f_1(t) \quad \text{donde: } \begin{cases} f_1(0) = 0 \\ f_1(1) = 1 \\ f_0(t) = f_1(1-t) \end{cases}$$

## Curvas cuadráticas

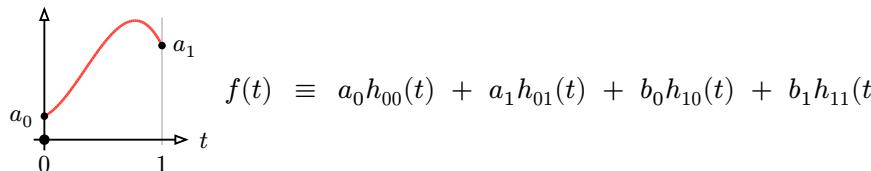
Para lograr una curva con pendiente nula en los extremos se puede usar el polinomio de grado 2 que cumple las 4 condiciones que le hemos impuesto:



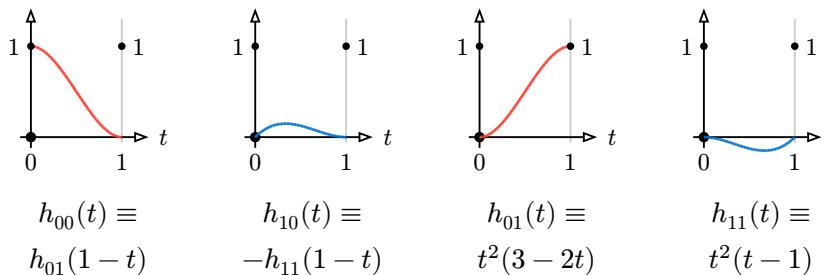
Sin embargo, en muchas aplicaciones queremos controlar los valores de las derivadas de  $f$  en 0 y 1 (hacerlos iguales a cualquier valor, distinto de 0).

## Control de derivadas con splines cúbicos de Hermite

Ahora  $f$  es un polinomio de grado 3 conocido como el *Spline cúbico de Hermite*:



Aquí  $b_0$  y  $b_1$  son las derivadas de  $f$  en 0 y en 1. Las funciones base son:

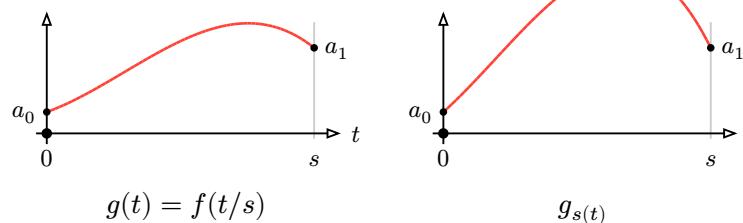


## Splines de Hermite en intervalos arbitrarios

Para fijar las derivadas  $b_0$  y  $b_1$  en un intervalo de longitud  $s > 0$ , usaríamos una versión modificada ( $g_s$ ) del Spline cúbico de Hermite:

$$g_s(t) \equiv a_0 h_{00}(t/s) + a_1 h_{01}(t/s) + sb_0 h_{10}(t/s) + sb_1 h_{11}(t/s)$$

donde se multiplican los coeficientes de las derivadas por  $s$ . Aquí vemos el resultado:



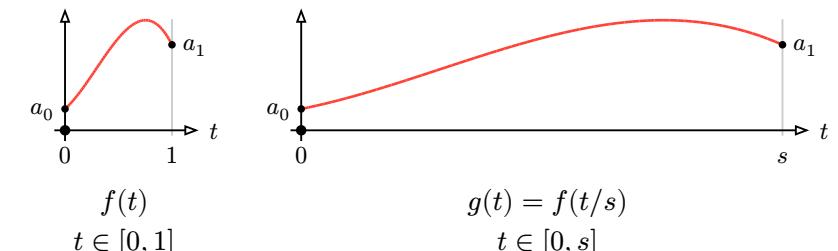
## Interpolación en intervalos de longitud arbitraria

Supongamos ahora que queremos interpolar en un intervalo  $[0, s]$  con una longitud  $s > 0$  no necesariamente igual 1.

- Si tenemos una función  $f$  que interpola en  $[0, 1]$ , podemos construir otra función  $g$  que interpola en  $[0, s]$ , sin más que hacer

$$g(t) \equiv f(t/s) \quad \text{donde: } t \in [0, s]$$

- Las derivadas de la función  $g$ , sin embargo, se dividen por  $s$



## Splines de Hermite en múltiples intervalos

En muchas aplicaciones, queremos que una función  $f$  tome una serie de valores arbitrarios  $a_0, a_1, \dots, a_{n-1}$  en una secuencia ordenada de puntos de tiempo conocidos  $t_0 < t_1 < \dots < t_{n-1}$ , y con derivadas  $b_0, b_1, \dots, b_{n-1}$

- Buscamos una función  $f$  tal que  $\forall i \in \{0, 1, \dots, n-1\}$ :
  1.  $f(t_i) = a_i$  y  $f'(t_i) = b_i$
  2.  $f$  interpola suavemente entre  $a_i$  y  $a_{i+1}$  en el intervalo  $[t_i, t_{i+1}]$
- Para lograr esto: **se usan  $n - 1$  splines cúbicos de Hermite**, cada uno de ellos desplazado al intervalo  $[t_i, t_{i+1}]$  y con los parámetros adecuados

## El spline de Hermite en el intervalo $[t_i, t_{i+1}]$

Para valores de  $t$  en el intervalo  $[t_i, t_{i+1}]$  usaremos, por tanto, el Spline  $g_i$ , definido así:

$$g_i(t) \equiv a_i h_{00}(u_i) + a_{i+1} h_{01}(u_i) + s_i b_i h_{10}(u_i) + s_i b_{i+1} h_{11}(u_i),$$

donde:

- $u_i$  = variable (dependiente de  $t$ ) que va desde 0 hasta 1 cuando  $t$  va desde  $t_i$  hasta  $t_{i+1}$ , es decir:

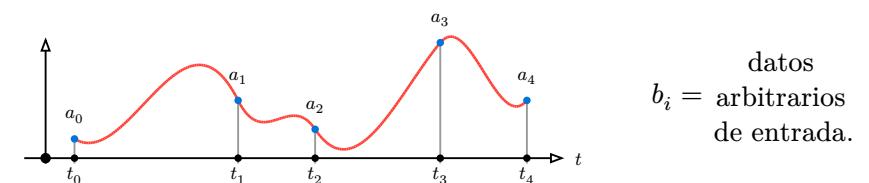
$$u_i \equiv (t - t_i)/s_i$$

- $s_i$  = longitud del intervalo ( $> 0$ ):

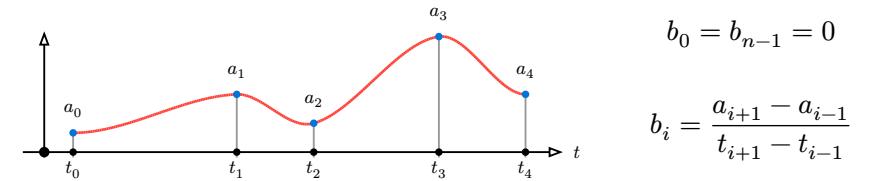
$$s_i \equiv t_{i+1} - t_i$$

## Gráfica del spline de Hermite en varios intervalos

Aquí vemos un ejemplo de una curva que interpola entre 5 valores



Se pueden asignar las derivadas de forma que la curva sea más suave:



### Subsección 3.2.

#### Curvas paramétricas.

## Curvas paramétricas

Una **curva paramétrica** es una función  $f$  que asigna a cada valor de un parámetro  $t$  real (en el intervalo  $[a, b]$ ) un punto  $f(t)$  en el espacio.

- El *espacio* puede ser el plano (2D) o el espacio tridimensional (3D).

Para expresar una trayectoria de un objeto en el espacio, usaremos *curvas paramétricas*:

- El valor de  $t$  es el *tiempo transcurrido* (en segundos, por ejemplo)
- Los puntos  $f(t)$  son las posiciones del objeto en el espacio a lo largo del tiempo.
- El intervalo  $[a, b]$  es el intervalo de tiempo en el que se está moviendo el objeto, típicamente  $a = 0$ , y  $b$  es la duración.
- Las curvas que vamos a usar son típicamente *continuas* (no hay saltos bruscos de un punto a otro distinto).

## Componentes de una curva

Puesto que  $\mathbf{f}(t)$  es un punto en el espacio 2D o 3D, sus coordenadas (en algún marco de referencia  $\mathcal{R}$ ) siempre serán funciones reales de variables real:

- En el espacio 2D, hay dos de estas funciones  $f_x$  y  $f_y$ :

$$\mathbf{f}(t) = (f_x(t), f_y(t))$$

- En el espacio 3D, hay tres funciones ( $f_x$ ,  $f_y$  y  $f_z$ )

$$\mathbf{f}(t) = (f_x(t), f_y(t), f_z(t))$$

## Ejemplo de curva paramétrica: línea recta

Un ejemplo sencillo de curva paramétrica en 2D es una curva que representa un simple trayectoria rectilínea entre dos puntos del plano  $\mathbf{p}_0 = (x_0, y_0)$  y  $\mathbf{p}_1 = (x_1, y_1)$ , en el intervalo  $[0, 1]$ .

- La función  $\mathbf{f}$  que define la curva será:

$$\mathbf{f}(t) = (1 - t)\mathbf{p}_0 + t\mathbf{p}_1$$

- Se puede descomponer en dos funciones  $f_x$  y  $f_y$  cada una de ellas es, por supuesto, una función de  $t$

$$f_x(t) = (1 - t)x_0 + tx_1$$

$$f_y(t) = (1 - t)y_0 + ty_1$$

- La velocidad es constante e igual a la distancia entre los dos puntos (ya que se recorre en una unidad de tiempo).

## Ejemplo de curva paramétrica: movimiento circular

Otro ejemplo podría ser un movimiento circular en el plano, con período  $T > 0$  segundos, con centro en el origen y radio  $r$ , constante

- La función  $\mathbf{f}$  que define la curva (suponiendo  $t$  en segundos) será:

$$\mathbf{f}(t) = r \cdot (\cos(2\pi t/T), \sin(2\pi t/T))$$

- Al igual que en el ejemplo anterior, obviamente se puede descomponer en dos funciones  $f_x$  y  $f_y$

$$f_x(t) = r \cdot \cos(2\pi t/T)$$

$$f_y(t) = r \cdot \sin(2\pi t/T)$$

- Teóricamente está definida para cualquier  $t > 0$  (no tiene porque parar nunca).

## Ejemplo de curva paramétrica: espiral

Un ejemplo algo más complejo es usar la curva anterior, pero hacer depender el radio  $r$  también del tiempo, es decir, se usa una expresión  $r(t)$  en lugar de la constante  $r$ :

- Una trayectoria en espiral que se aleja indefinidamente del origen, hacemos  $r(t) = 1.5t$

$$\mathbf{f}(t) = 1.5t \cdot (\cos(8\pi t), \sin(8\pi t))$$

- Una trayectoria circular que se aleja y acerca al origen, periódicamente:

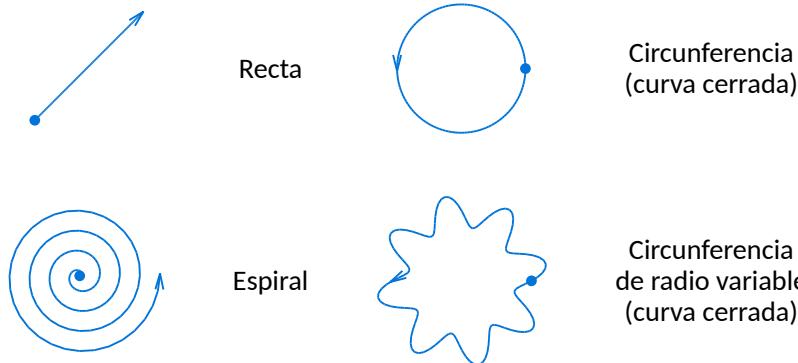
$$\mathbf{f}(t) = r(t) \cdot (\cos(2\pi t), \sin(2\pi t))$$

donde el radio oscila entre 1 y 1.6 (con 8 ciclos por vuelta)

$$r(t) = 1 + 0.3 \cdot (1 + \sin(8 \cdot 2\pi t))$$

## Gráficas de las curvas

Aquí vemos, a modo de ejemplo, las gráficas de las cuatro curvas que hemos descrito (se ve un disco en  $t = 0$ )



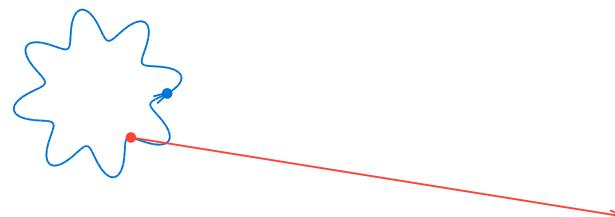
Una curva cerrada es una en la que  $f(0) = f(1)$

## Ejemplos del vector tangente

En una circunferencia:



En la circunferencia de radio variable:



## Vector de velocidad (tangente)

En un punto  $t$  a lo largo de una curva  $f$  (con componentes  $f_x, f_y$  y  $f_z$ ) se puede definir el vector tangente a la curva en ese punto  $v(t)$  como la derivada de  $f$  respecto de  $t$ .

$$v(t) \equiv \frac{df(t)}{dt} = \left( \frac{df_x(t)}{dt}, \frac{df_y(t)}{dt}, \frac{df_z(t)}{dt} \right)$$

- La dirección de  $v(t)$  es tangente a la curva, es la dirección en la que se mueve en  $t$  un punto que recorra la curva.
- La longitud de  $v(t)$  (se nota como  $\|v(t)\|$ ) es la velocidad a la que se mueve el punto en el instante  $t$ .

## Cálculo del vector tangente

En muchas aplicaciones queremos usar el vector tangente a una curva en un punto,

- Típicamente querremos alinear un objeto con la dirección de la curva en un punto dado (por ejemplo, si el objeto es un coche, debe estar alineado con la curva que está recorriendo).
- A veces se conoce la expresión analítica exacta de la tangente (p.ej. en una circunferencia, la tangente es perpendicular al radio)
- Si no se conoce la derivada analíticamente, se puede aproximar numéricamente con una diferencia finita:

$$v(t) \approx \frac{f(t + \Delta) - f(t - \Delta)}{2\Delta} \quad \text{donde } \Delta \text{ es pequeño}$$

## Curvas 2D y 3D para trayectorias

Subsección 3.3.

### Curvas e interpolación en 2D/3D.

3. Interpolación y curvas para animación.  
3.3. Curvas e interpolación en 2D/3D..

## Splines cúbicos de Hermite

El uso splines cúbicos de Hermite permite definir una curva suave que pasa por una serie de puntos dados y que tiene una dirección y velocidad dadas en cada uno de esos puntos.

- Se definen por tramos, cada tramo es un polinomio cúbico de Hermite.
- Cada tramo se define por dos puntos de control y dos vectores de control (uno en cada extremo).
- La curva es continua en posición y dirección.

Para ello, se considera las componentes X, Y (y Z, si procede) de forma independiente, y se usan las funciones reales de variable real que ya hemos visto.

En muchas aplicaciones será necesario diseñar trayectorias 2D y 3D que sean suaves y que:

- Pasan (o se acercan) a determinados puntos dados como entrada.
- Tienen una dirección y velocidad determinada en esos puntos.

Se pueden usar diversas técnicas de interpolación:

- Splines cúbicos de Hermite (los hemos visto para funciones, ahora los generalizamos a curvas).
- Curvas de Bezziers.
- Curvas B-Spline.

Sesión 11: Animación

Created 2025-12-09

Page 86 / 112.

3. Interpolación y curvas para animación.  
3.3. Curvas e interpolación en 2D/3D..

## Splines cúbicos de Hermite en 2D

En el caso 2D, se tienen como datos de entrada:

- Una secuencia de instantes de tiempo  $t_0 < t_1 \dots t_{n-1}$  en  $[0, 1]$
- Una serie de puntos  $\mathbf{p}_0, \mathbf{p}_1, \dots, \mathbf{p}_{n-1}$  en el plano, por los cuales se quiere que pase la curva ( $\mathbf{p}_i = (x_i, y_i)$ )
- Una serie de vectores  $\mathbf{v}_0, \mathbf{v}_1, \dots, \mathbf{v}_{n-1}$  que indican la dirección y velocidad de la curva en cada uno de los puntos ( $\mathbf{v}_i = (x'_i, y'_i)$ )

El problema es encontrar la curva  $\mathbf{f} = (f_x, f_y)$  que cumple las condiciones.

- Para eso se resuelve el problema de interpolación para cada una de las funciones  $f_x$  y  $f_y$ .
- Se pueden usar los Splines cúbicos de Hermite, que ya hemos visto.

## Splines cúbicos de Hermite en 2D

Por tanto, imponemos estas condiciones:

- La función  $f_x$  cumple:  $f(t_i) = x_i$  y  $f'(t_i) = x'_i$
- La función  $f_y$  cumple:  $f(t_i) = y_i$  y  $f'(t_i) = y'_i$

Así que usamos los Splines de Hermite en cada tramo para definirlas

$$f_x(t) \equiv x_i h_{00}(u_i) + x_{i+1} h_{10}(u_i) + s_i x'_i h_{01}(u_i) + s_i x'_{i+1} h_{11}(u_i)$$

$$f_y(t) \equiv y_i h_{00}(u_i) + y_{i+1} h_{10}(u_i) + s_i y'_i h_{01}(u_i) + s_i y'_{i+1} h_{11}(u_i)$$

donde  $i$  es el índice del tramo donde está  $t$ . Vectorialmente, se escribe:

$$\mathbf{f}(t) \equiv \mathbf{p}_i h_{00}(u_i) + \mathbf{p}_{i+1} h_{01}(u_i) + s_i \mathbf{v}_i h_{10}(u_i) + s_i \mathbf{v}_{i+1} h_{11}(u_i),$$

donde  $u_i := (t - t_i)/s_i$  y  $s_i := t_{i+1} - t_i$

## Problema: curva Hermite para una trayectoria

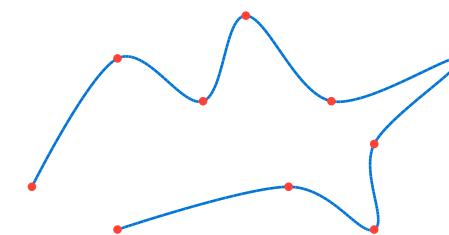
### Problema 11.1:

Implementar un proyecto en Godot en el cual el nodo raíz tiene un script que define dos arrays con: una serie de  $n$  puntos  $\mathbf{p}_0, \mathbf{p}_1, \dots, \mathbf{p}_{n-1}$  del plano  $y = 0$ , y: una serie de instantes de tiempo  $t_0, t_1, \dots, t_{n-1}$  (en segundos) con  $t_0 = 0$

- Sitúa en cada uno de esos puntos un disco pequeño visible, a modo de marcador.
- Incluye una función para calcular la posición y velocidad de la curva de Hermite que pasa por los puntos en los instantes dados, a partir de un  $t$  en  $[0, t_{n-1}]$ . En cada punto  $\mathbf{p}_i$  el vector de velocidad  $\mathbf{v}_i$  se calcula a partir de los puntos anterior y siguiente (como en el ejemplo anterior).
- En el método `_process(delta)` del script, calcula la posición y velocidad de la curva en el tiempo transcurrido desde el inicio, y mueve un objeto (un coche, por ejemplo) a esa posición, alineado con la dirección de la curva (usando el vector de velocidad como vector de dirección).

## Curva Hermite por varios puntos

Una curva (en azul) que pasa por varios puntos (en rojo).



$$\mathbf{v}_0 = \mathbf{v}_{n-1} = (0, 0)$$

$$\mathbf{v}_i = \frac{\mathbf{p}_{i+1} - \mathbf{p}_{i-1}}{t_{i+1} - t_{i-1}}$$

- En el primer y último puntos la velocidad es nula (se comienza y acaba de forma suave).
- En el resto de puntos, la velocidad es el vector que va desde el punto anterior al siguiente (dividida por el tiempo entre uno y otro).
- Este tipo de curvas son continuas, con derivada (tangente) continua, pero la curvatura es discontinua.

## Curvas de Beziers (cuadráticas)

La **Curva de Beziers** permite diseñar una trayectoria desde un punto  $\mathbf{p}_0$  hasta otro punto  $\mathbf{p}_2$ , pero de manera que la forma de la curva está influenciada por un tercer punto  $\mathbf{p}_1$ . Para ello:

- Se define la curva  $\mathbf{q}_0(t)$ , que interpola linealmente desde  $\mathbf{p}_0$  hasta  $\mathbf{p}_1$  (es un segmento de recta), e igualmente, se define  $\mathbf{q}_1(t)$  como la recta desde  $\mathbf{p}_1$  hasta  $\mathbf{p}_2$ :

$$\mathbf{q}_0(t) = (1-t)\mathbf{p}_0 + t\mathbf{p}_1 \quad \mathbf{q}_1(t) = (1-t)\mathbf{p}_1 + t\mathbf{p}_2(t)$$

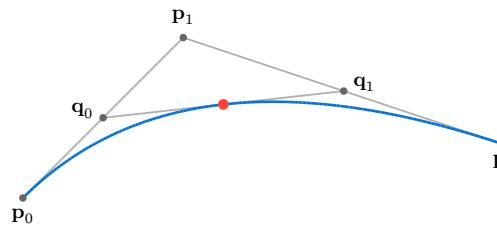
- La **Curva de Beziers (cuadrática)** ( $\mathbf{B}$ ) es la curva  $\mathbf{f}$  resultado de interpolar entre  $\mathbf{q}_0(t)$  y  $\mathbf{q}_1(t)$ , es decir:

$$\mathbf{B}(t) \equiv (1-t)\mathbf{q}_0(t) + t\mathbf{q}_1(t)$$

## Propiedades de la curva de Beziers cuadrática

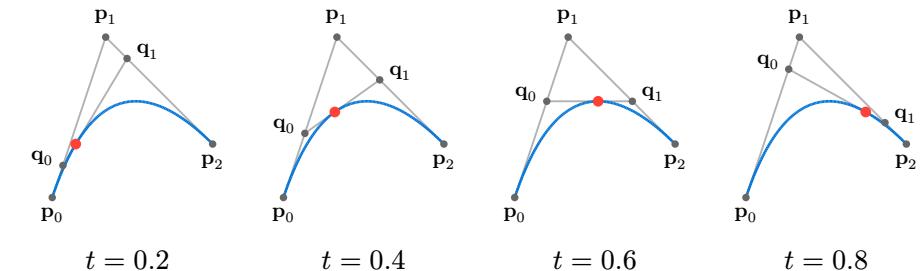
Esta curva cumple las siguientes propiedades:

- La curva sale de  $p_0$  (en  $t = 0$ ) y llega a  $p_2$  (en  $t = 1$ )
- No pasa necesariamente por  $p_1$ , pero este punto determina la dirección y velocidad al salir de  $p_0$  y al llegar a  $p_2$ .
- La curva (en X y en Y) se puede expresar como un polinomio de grado 2 en  $t$



## Diagrama de una curva de Beziers cuadrática

Aquí vemos los tres puntos, las dos rectas entre  $p_0$ ,  $p_1$  y  $p_2$ , así como la recta entre  $q_0$  y  $q_1$ , y el punto en la curva (en rojo):



## Curva de Beziers cúbica, y generalización

Las **curvas de Beziers cúbicas** usan 4 puntos ( $p_0, p_1, p_2, p_3$ ) en vez de 3

- Con esos puntos se pueden definir dos curvas de Beziers cuadráticas: la curva  $B_0$  que usa  $p_0, p_1, p_2$ , y la curva  $B_1$  que usa  $p_1, p_2, p_3$ .
- A partir de  $B_0$  y  $B_1$  se define la **Curva de Beziers cúbica** (polinomios de grado 3), interpolando:

$$B(t) \equiv (1-t)B_0(t) + tB_1(t)$$

- El esquema se puede generalizar recursivamente a  $n$  puntos usando polinomios de grado  $n - 1$
- Las curvas pasan por el primer y último puntos.
- Los puntos suelen denominarse **puntos de control** de las curvas.

## Curvas de B-spline

Las **Curvas B-spline** son parecidas a las curvas de Beziers, pero permiten mayor control sobre la forma de la curva.

- Los puntos distintos del primero y el último se suelen llamar **puntos de control**.
- Se permite controlar mejor las velocidades de la curva, usando un vector de valores reales, llamado **vector de nodos**.
- Al igual que las curvas de Beziers, se pueden definir recursivamente, para cualquier grado de los polinomios.
- Son ampliamente usadas en diseño asistido por ordenador y animación.

## Introducción

Sección 4.

### Animaciones en Godot

- 1. Animaciones con *AnimationPlayer*.
- 2. Animaciones mediante scripts

Godot ofrece diversas opciones para incorporar animaciones en un proyecto:

- Animaciones creadas con el editor usando nodos de tipo *AnimationPlayer*.
- Animaciones programadas mediante scripts, usando el método *\_process*
- Animaciones relacionadas con la simulación física, usando el método *\_physics\_process*

Sesión 11: Animación

Created 2025-12-09

Page 98 / 112.

4. Animaciones en Godot.

4.1. Animaciones con *AnimationPlayer*.

### Los nodos de tipo *AnimationPlayer*. Pistas.

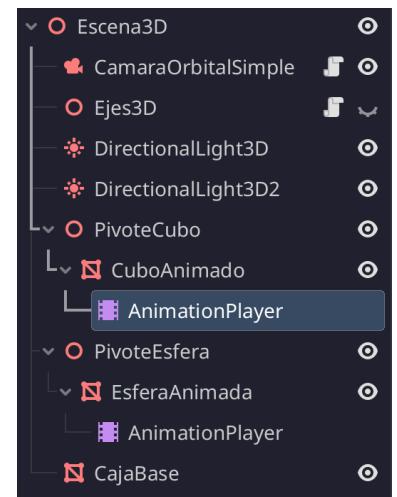
Subsección 4.1.

#### Animaciones con *AnimationPlayer*.

La clase *AnimationPlayer* (derivada de *Node*) permite crear animaciones para aplicaciones 2d o 3D, mediante un editor gráfico.

Cada nodo *AnimationPlayer* puede coneter una o varias **pistas** (*tracks*). Una pista es una especificación de como varía con el tiempo una **propiedad de algún otro nodo del árbol de escena**.

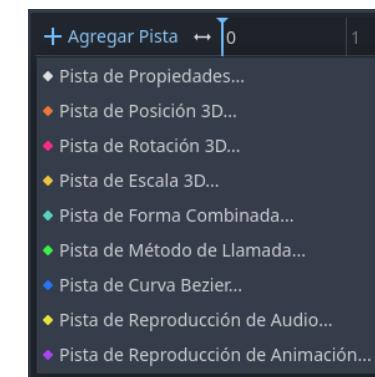
Un árbol de escena puede contener uno o varios nodos de este tipo.



## Tipos de pistas.

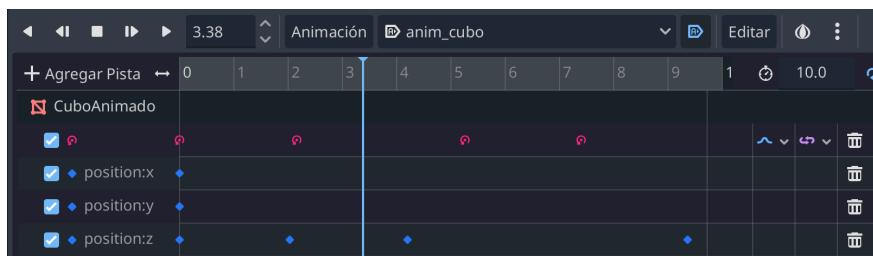
Una pistas pueden asociarse a distintos tipo de propiedades:

- Propiedades asociadas a la transformación del objeto: posición, rotación, y escala.
- Cualquier propiedad numérica, que varía con el tiempo según una curva de Bezziers.
- Cualquier propiedad no numérica del objeto, por ejemplo su textura, el material, o la malla usada (en un *MeshInstance*).



## Animación del cubo

Aquí vemos el panel correspondiente al nodo *AnimationPlayer* del cubo. La animación dura 10 segundos, se repite, y se activa al iniciar la aplicación.



Tiene 4 pistas (tracks):

- Una controla la rotación. Produce rotaciones entorno al eje Y.
- Otras tres controlan la posición en los ejes X, Y y Z. Cada keyframe tiene asociada una posición distinta en el eje Z. Son de tipo B-spline.

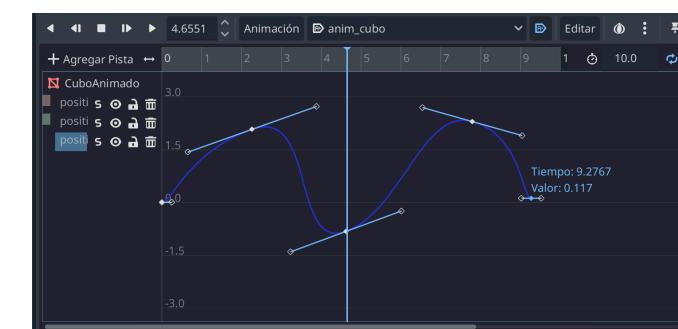
## Keyframes (cuadros clave)

En cada pista se debe añadir al menos un **keyframe** (lo usual es añadir varios)

- Un keyframe asocia un valor concreto a la propiedad en un instante de tiempo concreto.
- Para propiedades numéricas, se puede configurar como se interpola el valor entre dos keyframes consecutivos:
  - ▶ linealmente,
  - ▶ usando una curva suave
  - ▶ mediante curvas de Bezziers (B-spline). Estas se pueden editar interactivamente.
- Para propiedades no numéricas, el valor establecido en un keyframe se mantiene hasta el siguiente keyframe o hasta el final de la animación.

## Curvas de la posición Z del cubo

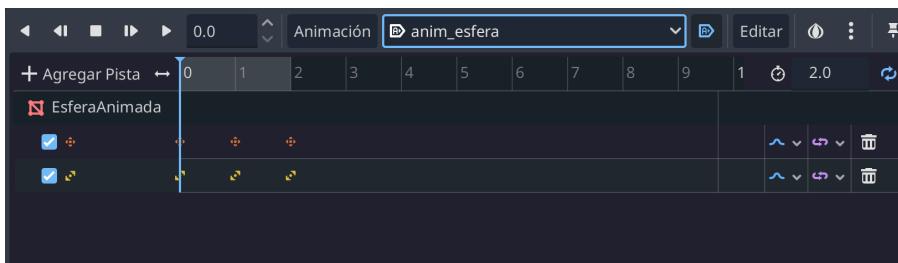
La posición del cubo se controla mediante tres curvas de tipo B-spline (una por cada eje). Aquí vemos la curva correspondiente a la posición en Z (que es la que varía):



Cada punto blanco es un **keyframe**. Se pueden arrastrar para cambiar su tiempo y su valor. Los segmentos que hay sobre ellos permiten modificar la derivada (pendiente) en cada uno de ellos.

## Animación de la esfera

Aquí vemos el panel correspondiente al nodo ***AnimationPlayer*** de la esfera. La animación dura dos segundos, se repite, y se activa al iniciar la aplicación.

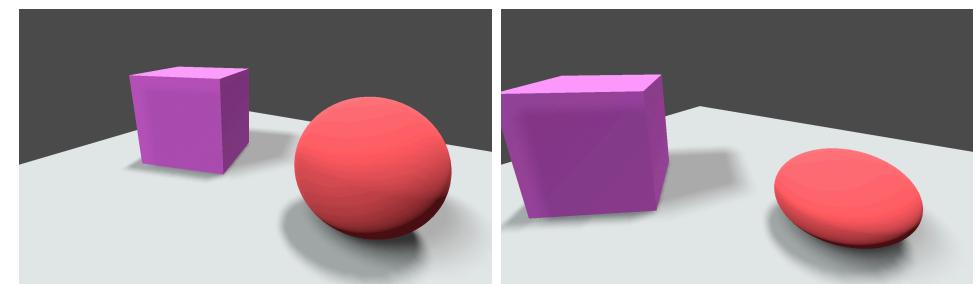


Tiene dos pistas (*tracks*):

- Una para controlar la posición. Cada *keyframe* tiene asociada una posición con una altura en Y distinta.
- Otra para controlar la escala. Ahora se asocia un escalado en Y diferente en cada *keyframe*.

## Una y otra imagen

Aquí vemos dos capturas de la aplicación en dos estados de la animación:



- El cuadrado de la izquierda se desplaza en horizontal y rota.
- La esfera de la derecha se desplaza en vertical y se escala.

## El método *process*

Otra forma de crear animaciones es mediante *scripts*. Se hace escribiendo código en el método ***\_process*** (de la clase ***Node***).

- Se invoca para cada nodo presente en un árbol de escena, antes de cada frame.
- Tiene un parámetro ***delta***, de tipo ***float***, que indica el tiempo (en segundos) transcurrido desde el último frame, se trunca a un límite máximo si hay un retardo grande (en una prueba ese límite es de 133 ms).
- Normalmente representa el tiempo real, pero si se trunca al valor máximo, ya no es así.
- En cada frame, el orden de invocación es:
  - ▶ En orden creciente de la **prioridad**, es decir, el valor de la propiedad entera ***process\_priority*** (vale 0 por defecto en todos los nodos).
  - ▶ Entre nodos de igual prioridad, **se usa un recorrido en pre-orden del árbol** (de arriba a abajo en el panel del árbol de escena del editor).

## Animaciones programadas

El código asociado al método `_process` puede, a modo de ejemplo:

- Modificar cualquier propiedad de un nodo, por ejemplo:
  - ▶ la posición, rotación o escala de un nodo **Spatial** (3D) o **Node2D** (2D)
  - ▶ la visibilidad de un nodo
  - ▶ el color o cualquier otro atributo del material de un nodo
- Mover cámaras, cambiar sus parámetros.
- Mover fuentes de luz, o insertar o destruir fuentes de luz.
- Crear o destruir nodos.
- Insertar nuevos nodos en un árbol.
- Cambiar el lugar de un nodo.
- Poner en marcha o parar audios o vídeos.
- Cualquier otra acción programable mediante código GDScript.

## Problemas: animaciones de ejemplo

### Problema 11.3:

Desarrolla un proyecto Godot para el ejemplo de animación de un **reloj con tres agujas** que se indica en la subsección 1.5 de este PDF.

### Problema 11.4:

Desarrolla un proyecto Godot para el ejemplo de animación de un **péndulo** que se indica en la subsección 1.5 de este PDF.

### Problema 11.5:

Desarrolla un proyecto Godot para el ejemplo de animación de una **bala de cañón** que se indica en la subsección 1.5 de este PDF.

## Problema: posición oscilante

### Problema 11.2:

Crea un proyecto Godot con una animación de una esfera cuya posición en X oscile periódicamente, con estas condiciones:

- El centro de la esfera tiene coordenada Z igual a 0, su coordenada Y es igual al radio, y su coordenada X varía entre  $-s$  y  $+s$ , donde  $s > 0$  es una constante declarada en el script.
- El período (tiempo en volver al mismo punto viajando en la misma dirección) es una constante  $T > 0$  declarada en el script (con unidades de segundos).
- La esfera se mueve siempre a velocidad constante en magnitud (es siempre  $s/T$ ), y el signo depende de la dirección.
- Tu animación debe producir esa velocidad constante, incluso teniendo en cuenta que los sucesivos valores de **delta** pueden cambiar entre frames. Especialmente, la magnitud de la velocidad debe ser constante aunque entre dos frames haya ocurrido un cambio de dirección en un extremo.

## Fin de transparencias.