

Animación

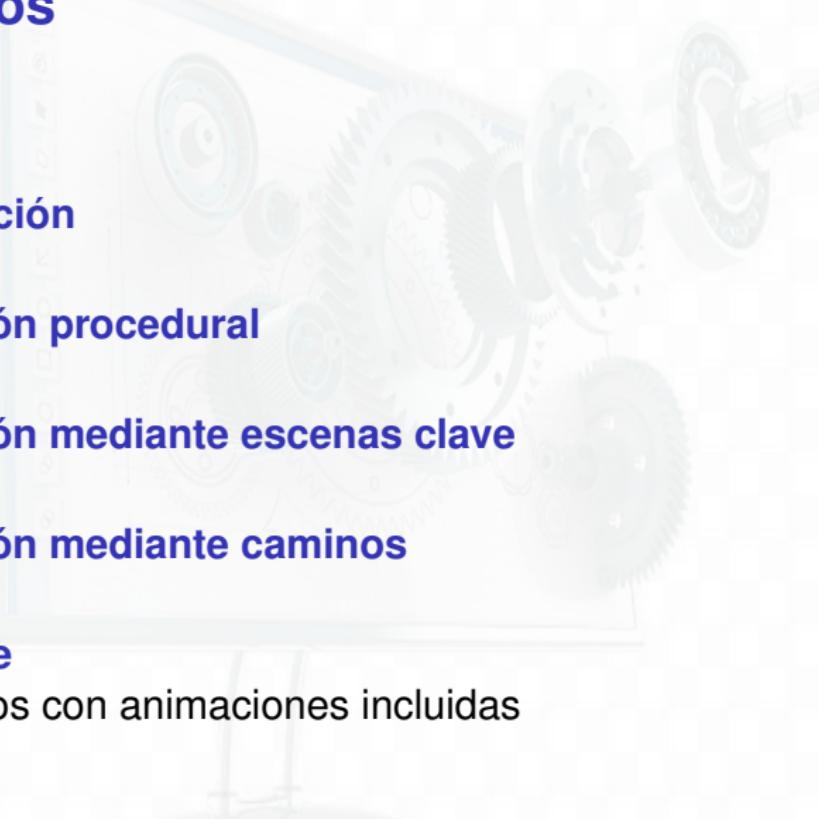
Francisco Velasco Anguita

Dpto. Lenguajes y Sistemas Informáticos
Universidad de Granada

Sistemas Gráficos

Grado en Ingeniería Informática
Curso 2021-2022

Contenidos

- 
- 1 Introducción**
 - 2 Animación procedural**
 - 3 Animación mediante escenas clave**
 - 4 Animación mediante caminos**
 - 5 Apéndice**
 - Modelos con animaciones incluidas

Objetivos

- Conocer los tipos de animación existentes
- Programar animaciones controlando la velocidad
- Programar animaciones mediante escenas clave
- Programar animaciones mediante caminos

Animación

Introducción

- **Animación:** Creación de la ilusión de que las cosas cambian.
- Se basa en el fenómeno de la persistencia de la visión.
- Percepción de movimiento: 24 imágenes por segundo.



Clasificación

● Animación convencional → A mano

- ▶ Orientada principalmente a animación 2D con apariencia plana
- ▶ Ventajas: Mayor flexibilidad y expresividad en los personajes
- ▶ Desventajas: Creación de todos los dibujos a mano



● Animación asistida por ordenador

- ▶ Se usa el ordenador en algunas fases del proceso: creación de dibujos, coloreado, ...
- ▶ Ventajas: Permite automatizar ciertos procesos reiterativos

● Animación por ordenador

- ▶ Orientada principalmente a animación 3D con entornos complejos
- ▶ Ventajas: Automatismo y manejo de grandes cantidades de información
- ▶ Desventajas: Falta de expresividad



Animación por ordenador

El problema de la falta de expresividad

- **Motion capture**

- ▶ Se busca dotar de expresividad humana a los personajes ...



Animación por ordenador

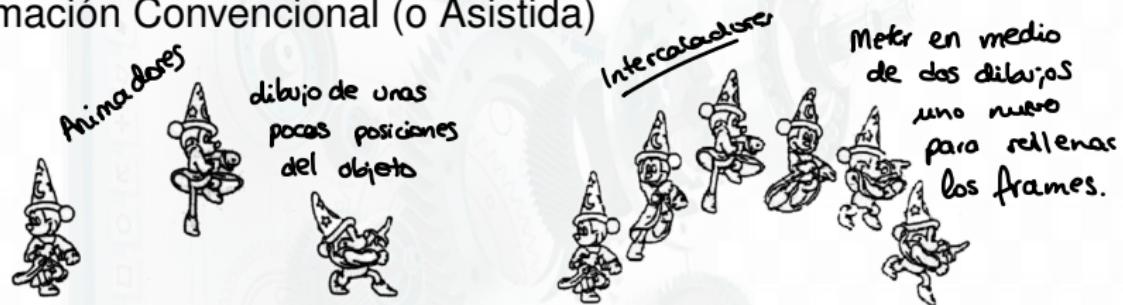
Motion capture

- ... también a nivel expresión facial

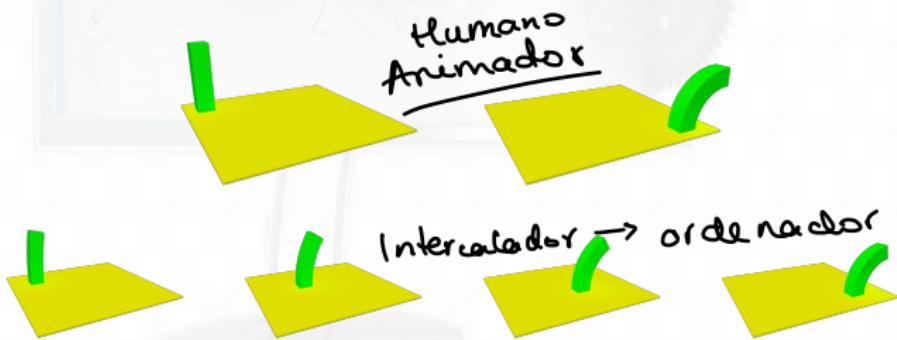


Animación Convencional vs. por Ordenador

- Animación Convencional (o Asistida)



- Animación por Ordenador



Etapas en una animación por ordenador

- Un corto de animación por ordenador requiere varias etapas
 - ▶ Guion, Storyboard, grabación de los diálogos, etc.
(los detalles en la asignatura de 4º)

• **Guion (Script)**

Uno de los aspectos más importantes de la animación.

¿Qué se quiere contar?

• **Esquema de la historia (Storyboard)**

Resumen gráfico (en viñetas) de la historia



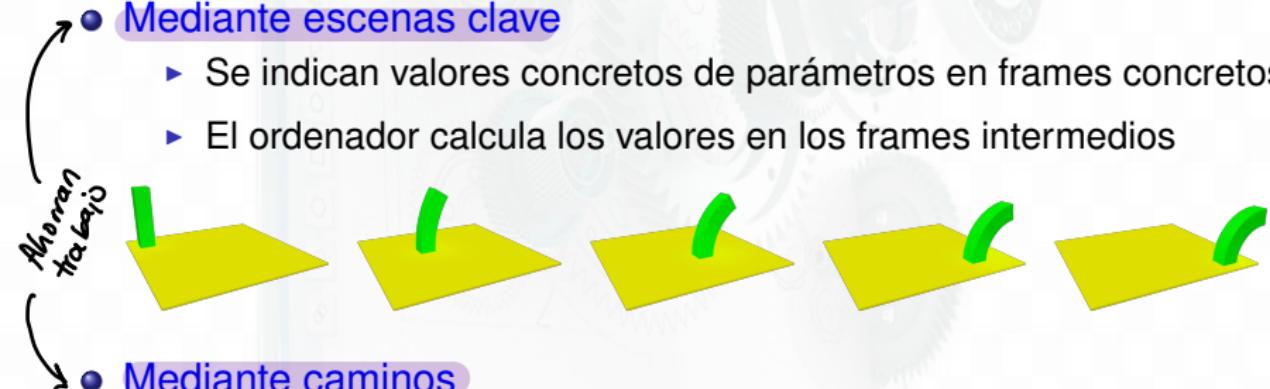
Modos de implementar la animación

- Animación procedural

- ▶ Modificando valores de parámetros (métodos `update();`)

- Mediante escenas clave

- ▶ Se indican valores concretos de parámetros en frames concretos
- ▶ El ordenador calcula los valores en los frames intermedios



- Mediante caminos

- ▶ La posición de un objeto viene determinada por una línea



Animación procedural

Three.js

- Cada objeto animable dispone de un método que:
 - ▶ Lee el tiempo
 - ▶ Modifica los parámetros que correspondan
- Dicho método es llamado para cada frame
- Permite una animación muy personalizada

Algoritmo: Método update de la clase MyScene

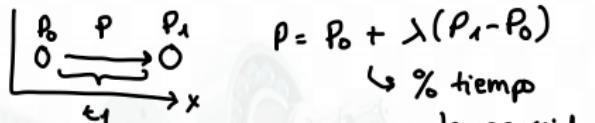
```
this.objetosAAnimar.forEach((unObjeto) => {  
    unObjeto.update();  
});
```



array de objetos activos en
cuanto a animación

Animación procedural

Método update

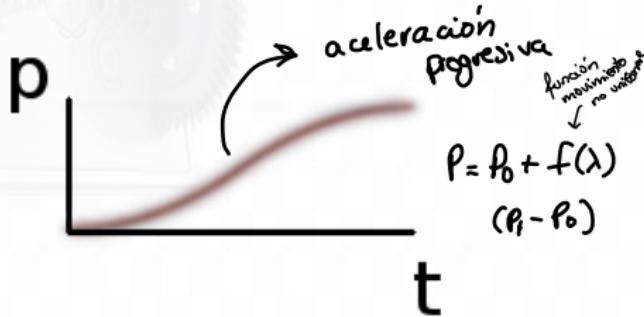


$$p = p_0 + \lambda(p_1 - p_0)$$

↪ % tiempo
transcurrido
 $\in [0, 1]$

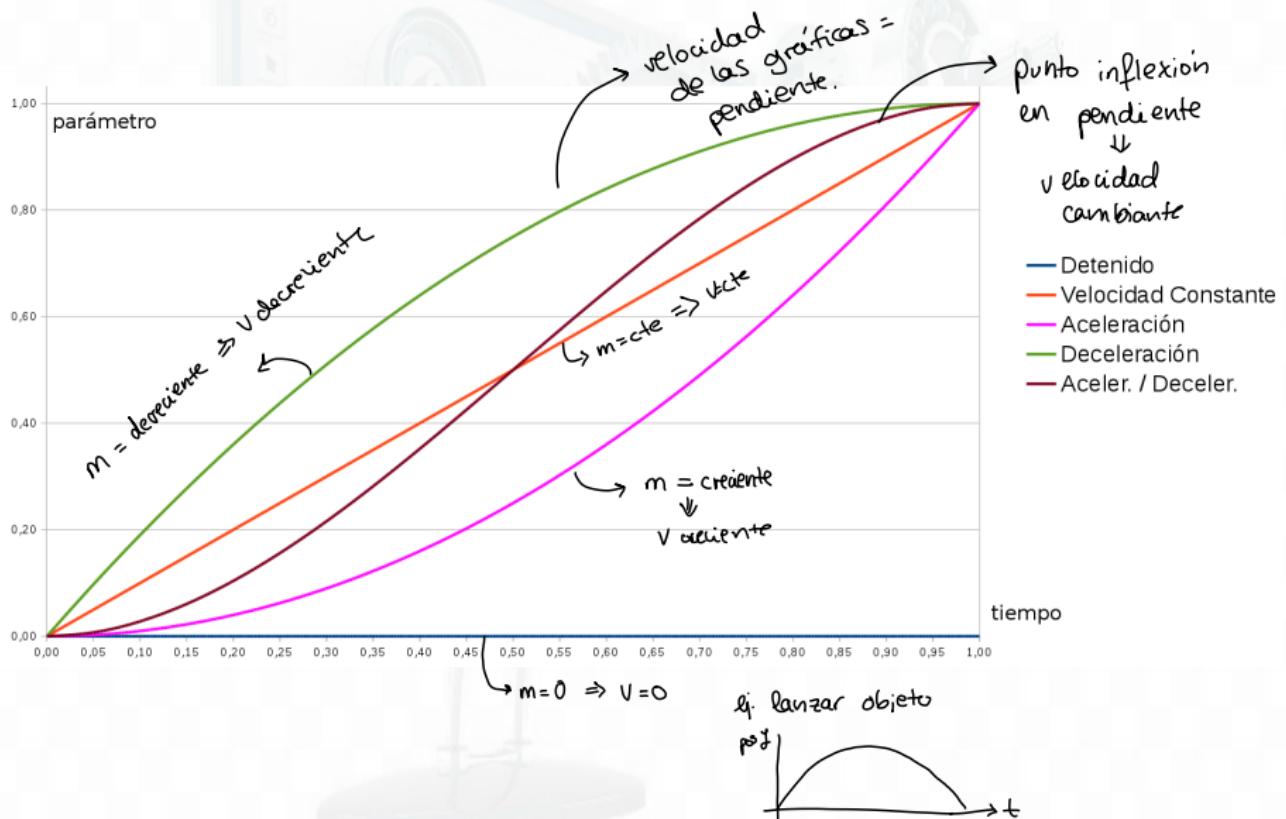
↑
Para mov. a V
constante

- Tiene toda la responsabilidad de la animación
- Debe conocer qué movimientos puede hacer la figura
- Saber en qué estado se encuentra cada movimiento
- Saber y controlar en qué momento ocurre cada cosa y a qué velocidad



Curvas de función: Controlan la velocidad de cambio de los parámetros

Curvas de función



Curvas de función

Expresiones aritméticas

Animaciones

rotaciones
intensidad luz
posición
otro.

- Premisas
 - ▶ El parámetro p varía entre v_0 y v_1
 - ▶ El tiempo t varía entre t_0 y t_1
- El valor actual de p se calcula como $p = v_0 + f(\lambda) \cdot (v_1 - v_0)$
 - ▶ Donde $\lambda = \frac{t-t_0}{t_1-t_0} \in [0, 1]$
 - ▶ $f(0) = 0$
 - ▶ $f(1) = 1$
- $f(\lambda)$ determina la forma de la función (ejemplos)
 - ▶ Velocidad Constante: $f(\lambda) = \lambda$
 - ▶ Aceleración: $f(\lambda) = \lambda^2$
 - ▶ Deceleración: $f(\lambda) = -\lambda^2 + 2 \cdot \lambda$
 - ▶ Aceleración al comienzo, deceleración al final:
$$f(\lambda) = -2 \cdot \lambda^3 + 3 \cdot \lambda^2$$

Velocidad independiente del ordenador

- En muchas ocasiones un objeto se mueve modificando su posición una determinada cantidad en cada frame
 - ▶ Por ejemplo, `objeto.position.x += 1;` espacio += 1 espacio;
- Sin embargo, si un ordenador 'A' es capaz de renderizar el doble de frames/segundo que otro ordenador 'B', dicho objeto se moverá el doble de rápido en 'A' que en 'B'
- ¿Cómo conseguir que los objetos se muevan a la velocidad deseada en todos los ordenadores?
 - ▶ Recurrimos a la ecuación de la cinemática $e = v \cdot t$
 - ★ ¿Cuánto debemos incrementar la posición del objeto en cada frame?
 - ★ El resultado de multiplicar la velocidad deseada por el tiempo que transcurrió desde el último frame.

$$\Delta e = v \cdot \Delta t$$

\hookrightarrow entre frames

Velocidad independiente del ordenador

Ejemplo

Ejemplo: Velocidad independiente del ordenador

```
// Al crear el objeto, creamos y referenciamos un reloj
this.reloj = new THREE.Clock();

// Se tiene en un atributo la velocidad
// (expresada como unidades / segundo)
this.velocidad = 10;           ↳ espacio, giro, intensidad, etc.

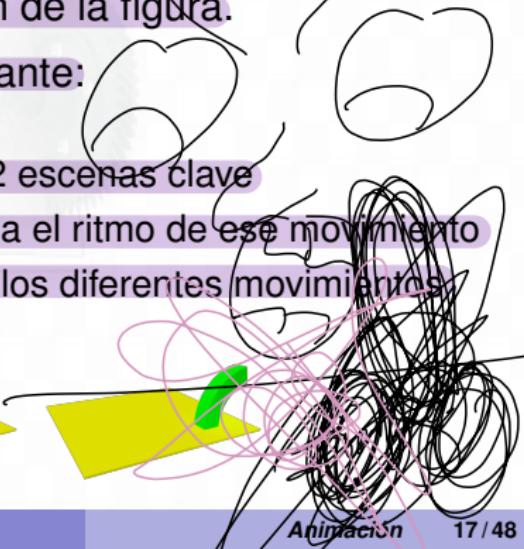
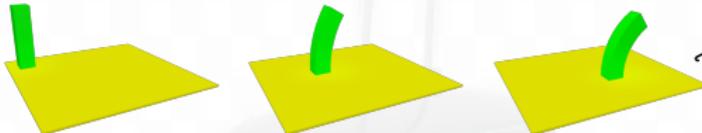
// En el método update(), que se ejecuta en cada frame
// y actualiza el objeto
var segundosTranscurridos = this.reloj.getDelta(); // segundos desde la última llamada
objeto.position.x += this.velocidad * segundosTranscurridos;
```

ejemplo => modelo reloj √ segundos = $\frac{2\pi}{60}$ → rad/s

una vuelta
cada minuto

Animación mediante escenas clave

- La animación se analiza y descompone en movimientos sencillos
- En cada movimiento se eligen momentos importantes
 - ▶ El principio, el final, tal vez algún momento intermedio
- Esos momentos importantes son las **Escenas clave**
- Para cada escena clave se definen los valores concretos de los parámetros que determinan la posición de la figura.
- La animación queda configurada mediante:
 - ▶ La definición de cada escena clave
 - ▶ El tiempo que transcurre entre cada 2 escenas clave
 - ▶ La definición de la gráfica que controla el ritmo de ese movimiento
 - ▶ La definición de cómo se encadenan los diferentes movimientos



Diseño escenas

Clave

$$p.y = h$$

$$s.x = 1$$

$$s.y = 1$$

$$s.z = 1$$



acelera

$$t_1$$

frena

$$t_2$$

acelera

$$t_3 = t_1$$

frenar

$$t_4 = t_0$$



anotación velocidades

Mov. sencillo



análisis
animación

$$\begin{cases} p.y \\ s.x \\ s.y \\ s.z \end{cases}$$

$$\sqrt{p.y} = R$$

$$s.x = 1$$

$$s.y = 1$$

$$s.z = 1$$

$$p.y = R/2$$

$$s.x = 2$$

$$s.y = 0.5$$

$$s.z = 2$$

Animación mediante escenas clave Three.js

- Se usa la biblioteca **Tween.js** → **biblioteca para interpolar**
 - ▶ <https://github.com/tweenjs/tween.js/>
 - ▶ `import * as TWEEN from '../libs/tween.esm.js'`
- Cada animación Tween es un **movimiento entre un origen y un destino**
- Se definen **2 variables locales con los parámetros a usar en el movimiento**
 - ▶ Cada variable local contiene los valores para los parámetros
 - ▶ Una variable con los valores para el origen y otra para el destino
- La animación se completa indicando:
 - ▶ El **tiempo, en ms, que transcurre entre el origen y el destino**
 - ▶ Cómo se modifican los parámetros de las figuras según los **parámetros de las variables locales usadas en la animación**

Animación con Tween

Three.js

Ejemplo: Uso de Tween.js

```

import * as TWEEN from '../libs/tween.esm.js'

// La figura que se quiere animar
this.figura = new THREE.Mesh ( . . . );

// Variables locales con los parámetros y valores a usar
var origen = { x: 0, y: 300 };
var destino = { x: 400, y: 50 };

// Definición de la animación: Variables origen, destino y tiempo
var movimiento = new TWEEN.Tween(origen).to(destino, 2000); // 2 seg

// Qué hacer con esos parámetros
movimiento.onUpdate (() =>{
    this.figura.position.x = origen.x;
    this.figura.position.y = origen.y;
});

// La animación comienza cuando se le indique
movimiento.start();

// Hay que actualizar los movimientos Tween en la función de render
TWEEN.update();

```

↑
origen varía por interpolación

Biblioteca Tween.js

Control de la velocidad (1)

- Se realiza con el método `easing(param)`
- Donde `param` puede ser
 - ▶ Velocidad constante
`TWEEN.Easing.Linear.None`
 - ▶ Aceleración al empezar y/o deceleración al acabar
`TWEEN.Easing.Quadratic.InOut`
 - ★ Cambiando `InOut` por `In` o `Out`, hace que sea solo aceleración o deceleración
 - ★ Cambiando `Quadratic` por `Cubic`, `Quartic`, `Quintic`, `Exponential` se consigue una mayor aceleración/deceleración

Biblioteca Tween.js

Control de la velocidad (y 2)

- ▶ Con retroceso
TWEEN.Easing.Back.InOut
- ▶ Elástico
TWEEN.Easing.Elastic.InOut
- ▶ Rebote
TWEEN.Easing.Bounce.InOut
 - ★ En los tres, cambiando InOut por In o Out, hace que el efecto se produzca solo al principio o al final

Biblioteca Tween.js

Ajuste de otros aspectos de la animación

- Número de repeticiones
 - ▶ Método `repeat (n)`
Se puede indicar `Infinity` para repeticiones infinitas
- Movimiento de vaivén
 - ▶ Método `yoyo (true)`
- Acciones a realizar antes y después de la animación
 - ▶ Método `onStart () => { ... }`
 - ▶ Método `onComplete () => { ... }`
- Encadenamiento de animaciones
 - ▶ Método `chain (otraAnimacion)`
- Pausado y continuación de una animación
 - ▶ Métodos `pause ()` y `resume ()`
- Detención de una animación
 - ▶ Método `stop ()` (se volvería a iniciar con `start ()`)

Biblioteca Tween.js

Three.js

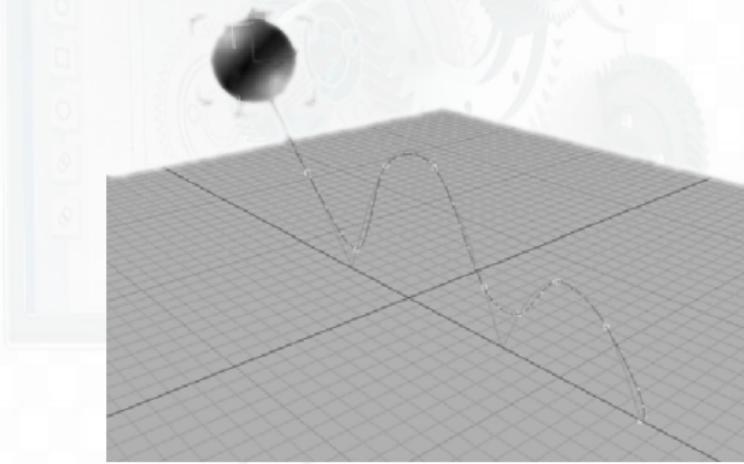
- Cada método devuelve el propio objeto
 - ▶ Se pueden encadenar los mensajes, definiendo la animación de una manera muy compacta

Ejemplo: Definición de una animación

```
var origen = { p : 0 };
var destino = { p : 100 };
var movimiento = new TWEEN.Tween( origen )
    .to( destino, 1000 )
    .easing( TWEEN.Easing.Linear.None )
    .onUpdate( () => { this.figura.position.x = origen.p } )
    .onComplete( () => { origen.p = 0; } )
    .repeat( Infinity )
    .yoyo( true )
    .start();
```

Animación mediante caminos

- Se define una trayectoria
- El objeto a animar sigue dicha trayectoria



Animación mediante caminos

Procedimiento

- Se define el camino mediante un Spline



- La posición y orientación del objeto a animar se toman del spline



La cámara sigue el camino marcado por el spline

Animación mediante caminos

Three.js

Definición de Splines

- Se definen indicando sus puntos de paso

Ejemplo: Definición de Splines

```
var spline = new THREE.CatmullRomCurve3 ([  
    new THREE.Vector3 (0, 0, 0), new THREE.Vector3 (0, 1, 0), ... ]);  
    // Además del array de puntos  
    // se puede añadir un segundo parámetro (true) para obtener un spline cerrado
```

- Si se desea dibujar la línea

Ejemplo: Dibujado de un Spline

```
// Se crea una geometría  
var geometryLine = new THREE.BufferGeometry ();  
// Se toman los vértices del spline, en este caso 100 muestras  
geometryLine.setFromPoints (spline.getPoints(100));  
// Se crea una línea visible con un material  
var material = new THREE.LineBasicMaterial ({color: 0xff0000, linewidth: 2});  
var visibleSpline = new THREE.Line (geometryLine, material);
```

Animación mediante caminos

Uso del Spline en la animación

Three.js

- Se puede obtener una posición y una dirección tangente

Ejemplo: Uso de Splines para modificar parámetros

```
// Se necesita un parámetro entre 0 y 1
//     Representa la posición en el spline
//     0 es el principio
//     1 es el final
var time = Date.now();
var looptime = 20000; // 20 segundos
var t = (time % looptime) / looptime;

// Se coloca y orienta el objeto a animar
var posicion = spline.getPointAt(t); → extrae punto y posiciona objeto en dicho punto
object.position.copy(posicion);
var tangente = spline.getTangentAt(t); → extrae la tangente
posicion.add(tangente); // Se mira a un punto en esa dirección
object.lookAt(posicion);
// Lo que se alinea con la tangente es la Z positiva del objeto
```

Si queremos que sea la punta del cono lo que se orienta



$$\text{cono.rotation } x = \pi/2$$

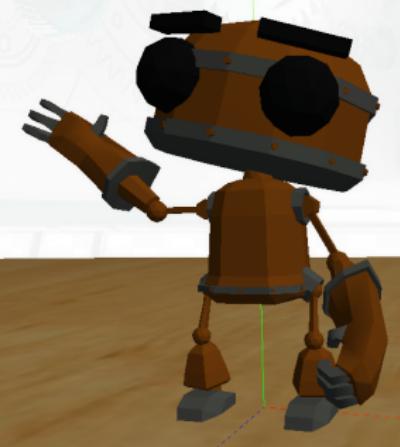
Animación combinando técnicas

- El método onUpdate de TWEEN es como un método update personalizado pero
 - ▶ Es llamado solo cuando esa animación está activa
- Se puede usar Tween para interpolar un parámetro entre 0 y 1 con una determinada curva de velocidades
 - ▶ Usar dicho parámetro para obtener la posición y orientación de una trayectoria (spline)
 - ▶ Y posicionar un objeto en dicho camino
- Se puede usar TWEEN para interpolar un parámetro entre 0 y 1
 - ▶ Nos indica que porcentaje ha transcurrido de un movimiento mayor
 - ▶ E iniciar y detener animaciones en función de eso

Apéndice: Modelos con animaciones incluidas

Archivos glTF

- glTF (GL Transmission Format) es una especificación libre para transmisión y carga de escenas 3D
 - ▶ <https://github.com/KhronosGroup/glTF>
- Entre sus características se encuentra que puede almacenar animaciones por escenas clave, no solo los modelos 3D
 - ▶ Aunque puede incluir solo el modelo, sin animaciones



Clases Three para gestionar animaciones glTF

• AnimationClip

- ▶ Representa una animación concreta de un modelo (ej. saludar)
 - ★ Una animación implica modificar varios parámetros (ej. la rotación de un brazo, la posición del cuerpo, etc.)
- ▶ Almacena varios objetos KeyframeTrack
 - ★ Cada objeto KeyframeTrack contiene un array de tiempos y otro de valores con los que se modificará un parámetro

• AnimationAction

- ▶ Es la clase que gestiona la animación
 - ★ Cuándo empieza, a qué velocidad transcurre, cuándo acaba, etc.
- ▶ Contiene un objeto AnimationClip y las referencias a los parámetros que hay que modificar con los valores de los KeyframeTrack del AnimationClip

• AnimationMixer

- ▶ El controlador general de las animaciones del modelo
- ▶ Contiene el método update que hace que todas las animaciones bajo su control se actualicen

Estructura Three.js de un modelo animado glTF



```
▼ AnimationMixer ⓘ  
  ► _root: Group {uuid: "498E9CC9-BA95-4513-96A6...  
  ▼ _actions: Array(14)  
    ► 0: AnimationAction {_mixer: AnimationMixer}  
    ▼ 1: AnimationAction  
      ► _mixer: AnimationMixer {_root: Group, _a...  
      ▼ _clip: AnimationClip  
        name: "Death"  
      ▼ tracks: Array(20)  
        ► 0: NumberKeyframeTrack {name: "Head_Z...  
        ► 1: NumberKeyframeTrack {name: "Head_Z...  
        ► 2: NumberKeyframeTrack {name: "Head_Z...  
        ► 3: VectorKeyframeTrack {name: "FootL...  
        ► 4: QuaternionKeyframeTrack {name: "Fo...  
        ▼ 5: VectorKeyframeTrack  
          name: "Body.position"  
        ► times: Float32Array(24) [0, 0.04166...  
        ► values: Float32Array(72) [0.0000524...
```

La clase AnimationMixer

- Se tiene un [AnimationMixer](#) por cada modelo con animaciones
 - ▶ Cada vez que se invoca su método [update](#) se actualizan las animaciones activas del modelo
 - ▶ ¿Qué ocurre si hay 2 animaciones activas y ambas 'mueven' el mismo elemento (ej. un brazo) de manera distinta?
 - ★ Cada animación tiene un peso configurable, entre 0 y 1
 - ★ El modelo *se moverá* según la mezcla de sus animaciones activas, la mezcla la realiza el mixer teniendo en cuenta los pesos
 - ▶ La mezcla de animaciones permite, por ejemplo, que un personaje pueda caminar y saludar a la vez



La clase AnimationAction

- Hay un **AnimationAction** por cada animación del modelo
- Es la clase con la que se controla cada animación individual
- Se construye a través del mixer con su método **clipAction**, pasándole una animación extraída del archivo glTF

Archivos glTF: AnimationMixer y AnimationAction

```
// Del archivo glTF se extrae:  
// El modelo jerárquico, que se referencia con la variable modelo  
// Las animaciones, que se referencian con el array animaciones  
  
this.mixer = new THREE.AnimationMixer (modelo);  
this.actions = {};  
  
// Los AnimationAction se referencian en un diccionario  
// indexado por el nombre de la animación  
for (var i = 0; i < animaciones.length; i++) {  
    this.actions[animaciones[i].name] = this.mixer.clipAction (animaciones[i]);  
}
```

Configurando un AnimationAction

Activación y ajuste de velocidad

- Método `play()`
 - ▶ Activa la animación
- Método `stop()`
 - ▶ Desactiva la animación y la resetea
- Método `reset()`
 - ▶ Cuando se active la animación, empezará desde el principio
- Método `setEffectiveTimeScale (v)`, ajusta la velocidad
 - ▶ $v == 1$, la velocidad a la que fue creada la animación
 - ▶ $v > 1$, la animación transcurre más rápida
 - ▶ $0 < v < 1$, transcurre más lenta
 - ▶ $v == 0$, la animación se queda en pausa
 - ▶ $v < 0$, se mueve al revés, hacia *atrás*
- Método `wrap (v1, v2, s)`
 - ▶ Se modifica la velocidad desde $v1$ a $v2$ en s segundos

Configurando un AnimationAction

Repeticiones y situación al finalizar

- Método `setLoop (modo, n)`
 - ▶ modo == `THREE.LoopOnce`, la animación se realiza una sola vez
 - ▶ modo == `THREE.LoopRepeat`, la animación se realiza `n` veces (si no se indica el número se repite infinitamente)
 - ▶ modo == `THREE.LoopPingPong`, la animación se realiza `n` veces (o infinitamente) en modo ida y vuelta
- Atributo booleano `clampWhenFinished`
 - ▶ Indica si el modelo se queda en la posición del último frame cuando acaba (true) o en la posición del primer frame (false)



false



true

Configurando un AnimationAction

El *peso* de la animación

- Método `setEffectiveWeight (peso)`
 - ▶ Establece el peso de la animación, un valor entre 0 y 1
 - ▶ Es tenido en cuenta por el mixer en la mezcla final
- Método `fadeIn (s)`
 - ▶ Incrementa el peso gradualmente de 0 a 1 en `s` segundos
- Método `fadeOut (s)`
 - ▶ Decrementa el peso gradualmente de 1 a 0 en `s` segundos
- Método `crossFadeFrom (otraAnimacion, s)`
 - ▶ Incrementa el peso gradualmente de 0 a 1 en `s` segundos
 - ▶ Al mismo tiempo, decrementa el peso de `otraAnimacion`

Modelos glTF

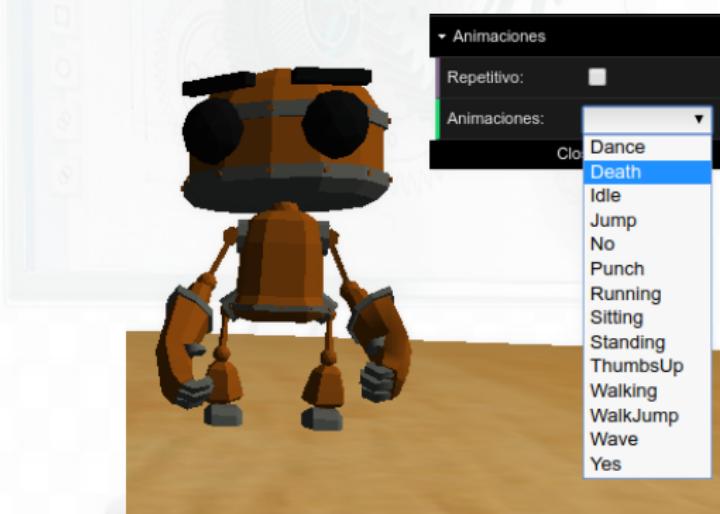
Su inclusión en una aplicación Three.js

- Además de cargar el modelo
 - ▶ Three.js dispone de un cargador para este formato
- Hay que averiguar qué animaciones tiene
 - ▶ Esta información se puede obtener del sitio donde se haya descargado el modelo
 - ▶ Analizando el propio modelo cargado también se puede obtener el listado de animaciones
- Entonces se pueden usar, solas, o combinadas
 - ▶ Una vez identificadas las animaciones, se procede a su activación, parada, combinación de varias, etc.

Modelos glTF

Identificando sus animaciones

- Crearemos un programa genérico que:
 - ▶ Cargue el modelo
 - ▶ Cree una interfaz de usuario con las animaciones
 - ▶ Nos permita ver cada animación interactuando con la interfaz



Conociendo un modelo glTF con animaciones

Algoritmo

- ① Cargar el modelo
 - ▶ Se obtienen por separado el modelo jerárquico y las animaciones
 - ▶ Cada animación se denomina clip
 - ② Se crea un AnimationMixer para este modelo
 - ③ Para cada clip, del que obtenemos también su nombre
 - a Se crea un AnimationAction
 - b El AnimationAction añade a un diccionario.
El diccionario está indexado por los nombres de los clips
 - c El nombre se añade a una lista desplegable,
para la interfaz de usuario
- Cuando el usuario selecciona un nombre de la lista,
se accede a su respectivo AnimationAction para lanzarla

Conociendo un modelo glTF con animaciones

Carga

- Se usa la clase GLTFLoader

```
import { GLTFLoader } from '../libs/GLTFLoader.js';
```

Ejemplo: Carga de un modelo glTF

```
var loader = new GLTFLoader();
loader.load( '../models/gltf/robot.glb', ( gltf ) => {
    // El modelo está en el atributo scene
    var model = gltf.scene;
    // Y las animaciones en el atributo animations
    var animations = gltf.animations;

    // No olvidarse de colgar el modelo de la escena (directa o indirectamente)
    this.add( model );

    // Se llama al método que crea el AnimationMixer
    // y el diccionario de AnimationAction
    this.createActions( model, animations );

    // Se llama al método que crea la interfaz de usuario
    this.createGUI( gui, str );

}, undefined, ( e ) => { console.error( e ); }
);
```

Conociendo un modelo glTF con animaciones

Creación del diccionario de acciones

Ejemplo: Creando la estructura THREE para las animaciones

```
createActions (model, animations) {  
    // Se crea un mixer para este modelo  
    this.mixer = new THREE.AnimationMixer (model);  
  
    // Se crea el diccionario de AnimationAction y el array de nombres  
    this.actions = {};  
    this.clipNames = [];  
  
    for (var i = 0; i < animations.length; i++) {  
        // Se toma una animación de la lista de animaciones del archivo gltf  
        var clip = animations[i];  
  
        // Se incorpora el clip al mixer y obtenemos su AnimationAction  
        var action = this.mixer.clipAction (clip);  
  
        // Se añade el AnimationAction al diccionario con su nombre  
        this.actions[clip.name] = action;  
  
        // Se añade el nombre a la lista de nombres, para la interfaz  
        this.clipNames.push (clip.name);  
    }  
}
```

Conociendo un modelo glTF con animaciones

Creación de la interfaz de usuario

Ejemplo: Creando la interfaz de usuario

```
createGUI (gui, str) {  
  
    this.guiControls = {  
        // En este campo estará la lista desplegable de animaciones del archivo  
        current : 'Animaciones',  
        // Este campo nos permite ver cada animación una sola vez o repetidamente  
        repeat : false  
    }  
  
    // Creamos y añadimos los controles de la interfaz de usuario  
    var folder = gui.addFolder (str);  
    var repeatCtrl = folder.add (this.guiControls, 'repeat').name('Repetitivo:');  
    var clipCtrl = folder.add (this.guiControls, 'current')  
        .options(this.clipNames).name('Animaciones:');  
  
    // Cada vez que uno de los controles de la interfaz de usuario cambie,  
    // llamamos al método que lance la animación elegida  
    clipCtrl.onChange (() => {  
        this.fadeToAction (this.guiControls.current, this.guiControls.repeat);  
    });  
    repeatCtrl.onChange (() => {  
        this.fadeToAction (this.guiControls.current, this.guiControls.repeat);  
    });  
}
```

Conociendo un modelo glTF con animaciones

El método que lanza la animación

Ejemplo: El método que lanza la animación

```
fadeToAction (name, repeat) {  
    // Referenciamos la animación antigua y la nueva actual  
    var previousAction = this.activeAction;  
    this.activeAction = this.actions[ name ];  
  
    // Reseteamos la nueva animación, elimina cualquier rastro de la ejecución anterior  
    this.activeAction.reset();  
    // La nueva comenzará mientras la anterior se para  
    // Se emplea un 10% del tiempo de la nueva en esa transición  
    this.activeAction.crossFadeFrom (previousAction, this.activeAction.time/10 );  
    // Hacemos que la animación se quede en su último frame cuando acabe  
    this.activeAction.clampWhenFinished = true;  
    // Ajustamos su peso al máximo, ya que queremos ver la animación en su plenitud  
    this.activeAction.setEffectiveWeight( 1 );  
    // Se establece el número de repeticiones  
    if (repeat) {  
        this.activeAction.setLoop (THREE.Repeat);  
    } else {  
        this.activeAction.setLoop (THREE.LoopOnce);  
    }  
    // Una vez configurado el accionador, se lanza la animación  
    this.activeAction.play();  
}
```

Usando un modelo glTF con animaciones

Método

- Una vez conocido el modelo, y las animaciones que tiene
- Solo queda usarlas en la aplicación donde se vaya a incorporar dicho modelo
 - ▶ Se carga el modelo
 - ▶ Se almacenan sus AnimationAction en un diccionario indexado por los nombres de las animaciones
 - ★ Igual que cuando se hizo el análisis de la animación
 - ★ Permite acceder a ellas por su nombre
 - ▶ Se manipulan los AnimationAction según necesidad
 - ★ `stop()` para parar una animación
 - ★ `play()` y ajuste de peso y velocidad para activarla
 - ★ A veces se tienen varias animaciones activas (`play`) y se va *jugando* con los pesos.

Usando un modelo glTF con animaciones

Ejemplo

- Suponemos que tenemos activas en el robot las animaciones “Idle”, “Walking” y “Wave”
- Si se quiere que el robot esté en reposo
 - ▶ El peso de “Idle” es 1, y los pesos de “Walking” y “Wave” son 0
- En un momento dado, se quiere que el robot empiece a caminar

Robot: De reposo a caminando

```
// En 0.5 segundos el robot pasa de reposo a caminando
this.actions["Idle"].fadeOut (0.5);
this.actions["Walking"].fadeIn (0.5);
```

- Ahora se quiere que empiece a saludar

Robot: Añadiendo el saludo

```
// Inicia el saludo sin detener otras animaciones
this.actions["Wave"].fadeIn (0.5);
```

Animaciones en modo aditivo

Planteamiento

- Suponer que se tienen dos animaciones con el peso máximo, por ejemplo, caminar y saludar
- Ambas animaciones actúan sobre los brazos del robot
 - ▶ Saludar “tira” del brazo hacia arriba
 - ▶ Caminar “tira” del brazo en otras direcciones
 - El saludo no lo hace igual estando en reposo que caminando



Animaciones en modo aditivo

Solución

- Consiste en poner la animación de saludar en modo aditivo

Robot: Saludar en modo aditivo

```
// Cuando se están extrayendo las animaciones del modelo
// para construir los AnimationAction

this.actions = {};
for (var i = 0; i < animations.length; i++) {
    var clip = animations[i];

    // Se identifica la animación que se va a poner en modo aditivo

    if (clip.name == "Wave") {

        // Se pone en modo aditivo

        THREE.AnimationUtils.makeClipAdditive (clip);
    }

    // Se construye su AnimationAction y se añade al diccionario

    this.actions[clip.name] = this.mixer.clipAction (clip);
}
```



Animación

Francisco Velasco Anguita

Dpto. Lenguajes y Sistemas Informáticos
Universidad de Granada

Sistemas Gráficos

Grado en Ingeniería Informática
Curso 2021-2022