Animación

Sistemas Gráficos

Grado en Ingeniería Informática

Curso 2019/2020

1. Introducción

Cuando se habla de animación en el contexto de un sistema gráfico, nos estamos refiriendo a trasmitirle al usuario **la ilusión de que algo en la escena que está observando cambia a lo largo del tiempo**. Puede ser que un personaje está caminando, que la luz se aclara, que algún material se transforma de un color en otro, etc.

Al igual que ocurre en el cine, esta ilusión se consigue mostrándole al espectador una secuencia de imágenes estáticas (fotografías) a una determinada velocidad medida en imágenes por segundo.

Pero ¿por qué el espectador percibe movimiento en vez de una secuencia de fotografías estáticas? Se debe a un fenómeno que se ha conocido tradicionalmente como **persistencia de la visión**, mediante el cual, el cerebro, al procesar las imágenes procedentes de la retina, si estas imágenes se reciben a una cierta velocidad y hay poca variación entre ellas, el cerebro las percibe como una sola imagen en movimiento y no como varias imágenes estáticas.

Hoy día, las velocidades que se usan son **24 imágenes por segundo** en el cine y 29.97 imágenes por segundo en televisión (sistema NTSC), aunque en Europa y Sudamérica se usa el sistema PAL para TV que proyecta 25 imágenes por segundo.

Así, si mostráramos a la velocidad adecuada cada uno de los dibujos del ratón Mickey (figura 1), veríamos el salto de dicho personaje Disney de una manera continua.

Figura 1: Diversas imágenes fijas de un personaje que proyectadas a la velocidad adecuada harían ver que el salto se produce de manera continua.

1.1. Clasificación

Esta técnica no solo la ha usado el cine que en definitiva fotografía escenas interpretadas por actores y actrices. De hecho, el uso de la palabra animación es más propio de la creación de películas a partir de escenas dibujadas, como la mostrada en la figura 2. Lo que conocemos precisamente como *dibujos animados*.

Figura 2: Un fotograma de una conocida serie de animación. Un dibujo 2D.

En este sentido, se puede establecer una clasificación atendiendo principalmente a los medios que se usan para obtener los fotogramas (también llamados frames) que van a constituir la animación.

Animación convencional

Se llama así si todo se hace a mano, el trazado de los dibujos, el coloreado, etc. Se usa principalmente para animación 2D donde todos los dibujos tienen apariencia plana. Los

escenarios pueden trasmitir profundidad debido a la perpectiva pero son **dibujos 2D hechos a mano**, siendo esta su principal desventaja.

Sin embargo, este tipo de animación posee una gran ventaja: la expresividad que un buen dibujante puede darle a los personajes, obteniendo animaciones de gran calidad artística.

Animación asistida por ordenador

No es más que una animación convencional, es decir, basada en dibujos 2D pero que en algunas fases del proceso se ha usado el ordenador. Por ejemplo, en el coloreado. Así, se mantienen las ventajas de la animación convencional y se atenúan en parte sus desventajas al poder **automatizar ciertos procesos reiterativos**.

Animación por ordenador

Ahora el ordenador no es una herramienta que ayuda en ciertas tareas del proceso de la animación, sino que adquiere más protagonismo. Los escenarios y personajes se han modelado por ordenador, los parámetros que definen cada movimiento se han configurado en los modelos y **es el ordenador el que en base a toda la información que se le ha dado genera la secuencia de imágenes que dan lugar a la animación**.

Figura 3: En una imagen generada por ordenador es más fácil obtener elementos que transmiten la tridimensionalidad de la escena como la perspectiva y el volumen. Además de poder mostrar otra calidad de imagen gracias al tratamiento de la luz o como se muestran los diversos materiales.

Ha permitido crear animaciones 3D (figura 3) donde la sensación de espacio tridimensional ya no solo se consigue con la perspectiva, sino que el shading (combinación de tonos claros y oscuros en una superficie) contribuye a trasmitir el volumen que tienen los distintos elementos de la escena. Por no hablar de otra calidad de imagen gracias al tratamiento de la

luz, la representación de distintos materiales, manejo de transparencias, brillos, reflejos, refracciones, etc. Elementos que, dibujarlos a mano, sería bastante más complicado.

Por ejemplo, si se compara la cabeza de Homer Simpson en las figuras 2 y 3, se puede observar como el shading de la imagen 3D ayuda a que el observador aprecie mejor el volumen de la testa del personaje. Otro ejemplo lo tenemos en la boca de Moe Szyslak.

Sin embargo, frente a esta gran ventaja con respecto a los otros tipos de animación nos encontramos con una desventaja también importante y es la falta de expresividad de los personajes. Ya no solo a nivel de la expresividad en el rostro de un personaje, sino en movimientos más básicos como es caminar. El resultado es que se obtienen personajes que se mueven como robots en vez de como humanos.

Con el paso del tiempo este problema se ha ido superando gracias al **Motion capture**, que no es más que calcar la expresividad en el rostro o movimientos de un actor humano en un modelo de ordenador.

La técnica consiste en poner unos dispositivos en el cuerpo del actor y captar cómo se mueven esos dispositivos para con esa información controlar el movimiento del modelo de ordenador (figura 4).

Figura 4: El movimiento de los dispositivos que hay en el traje del actor se usa para definir el movimiento en el modelo de ordenador.

A fin de cuentas, lo que marca la diferencia y le da expresividad y naturalidad a la animación son esos pequeños movimientos que se hacen con unas partes del cuerpo mientras se realiza un movimiento principal con otras partes.

Observar algo tan habitual en un humano como es caminar. Podemos decir que todos caminamos igual, primero un pie, luego el otro, etc. sin embargo, todos tenemos un caminar distinto que incluso nos puede llevar a reconocer a una persona a la que no le hemos visto la cara simplemente por la forma de caminar. **Son esas diferencias sutiles lo que da naturalidad a una animación por ordendador** y es lo que se consigue con la técnica de motion capture.

Esta técnica de motion capture no solo se usa para los movimientos más amplios de

cuerpo y extremidades, sino que también se usa para captar y reproducir los movimientos de una cara y poder dotarle de emociones al modelo de ordenador (figura 5).

Figura 5: Gracias a capturar los pequeños movimientos de los dispositivos (puntos verdes) que hay en la cara de Zoe Saldana se consigue que Neytiri se muestre tan enfadada como Zoe.

En cualquier caso, en una animación por ordenador no desaparece por completo el trabajo del humano, ni queda dedicado exclusivamente a un trabajo técnico de programación. Al contrario, el trabajo de un humano artista sigue siendo una parte fundamental de una animación por ordenador.

En una animación convencional, o asistida por ordenador, podemos hablar de dos tipos de profesionales directamente relacionados con el movimiento de los personajes:

- **Animador:** Es quién define el movimiento en base a unas poses clave. En la figura 6.a sería el animador el que ha definido que Mickey dé ese salto con esas posturas concretas de su cuerpo y no otras. Define un movimiento en base a unos **dibujos clave**.
- **Intercalador:** Es el que, a partir de los dibujos clave que le proporciona el animador y conociendo el tiempo que debe durar ese movimiento, realiza los dibujos intermedios para exista una transición progresiva entre cada dos dibujos clave (figura 6.b).

a) b)

Figura 6: El animador define un movimiento en base a unos dibujos clave (a) y los intercaladores realizan los dibujos intermedios.

En una animación por ordenador la profesión del animador se mantiene, es quien decide los movimientos en base a posiciones clave, aunque ahora dichas posiciones las define dándole unos valores concretos a unos parámetros del modelo en unos instantes de tiempo concretos.

Sería la profesión del intercalador la que quedaría sustituida por **el ordenador**, que es quien **interpola el valor de los parámetros del modelo en base a los valores definidos por el animador para las escenas clave**.

En la figura 7 un animador humano ha definido un movimiento en el que una columna verde va de un extremo al otro del suelo inclinándose mientras se desplaza. Lo ha hecho indicando que posición e inclinación (parámetros) tiene la columna en los instantes de tiempo t_0 y t_1.

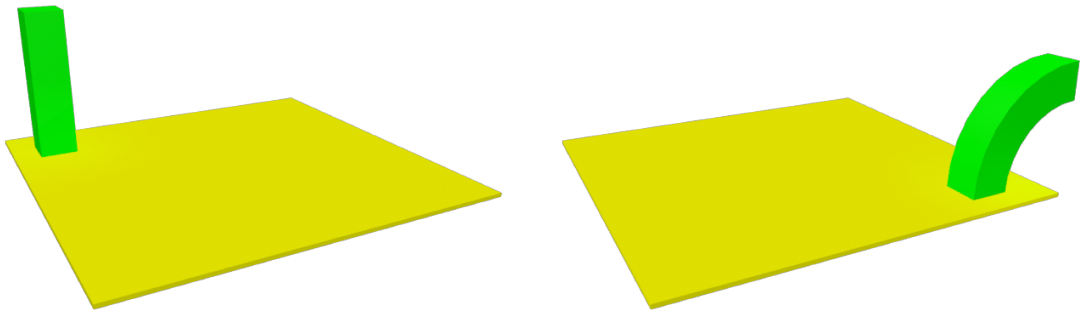


Figura 7: El animador define un movimiento en base a unas escenas clave indicando valores de parámetros concretos en instantes de tiempo concretos.

Cuando se le pide al ordenador que realice la animación, renderiza las imágenes intermedias, figura 8. Para ello, le va dando a los parámetors involucrados valores que obtiene por interpolación entre los valores extremos que ha definido el animador.

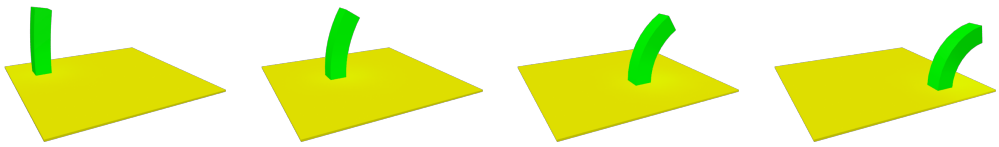


Figura 8: El ordenador obtiene las imágenes intercaladas interpolando parámetros.

1.2. Etapas en una animación por ordenador

Un cortometraje (y por supuesto un largometraje) de animación por ordenador requiere de varias etapas en las que se va avanzando desde el esbozado de un guion hasta la grabación final pasando por diversas tareas que le van dando forma a la animación. Todas estas etapas son explicadas en la asignatura de 4^o dedicada en exclusiva a animación por ordenador.

Sin embargo, en este tema de esta asignatura si debo insistir en dos tareas que son fundamentales. Da igual la envergadura del proyecto, ya sea un producción que aspire a competir en los Goya o una pequeña animación en una parte de un videojuego. Esas tareas son el guion y el esquema de la historia.

Guion (Script)

Es la tarea principal de la animación, lo que determina todo lo demás, y es ¿qué se quiere contar? Cualquier película con un mal guion será una mala película, aunque técnicamente sea perfecta o esté plagada de efectos especiales. Una película que no sabes que te están contando, que no terminas de saber cuál es la historia no dejará mucha huella en ti.

En cada animación, es fundamental tener claro por qué se hace, qué se quiere contar, qué idea se quiere transmitir al espectador. Y a partir de ahí ir concretando todo lo demás para llegar a ese fin.

Esquema de la historia (Storyboard)

Se trata de un resumen gráfico, en viñetas, de la historia. Una especie de comic donde visualmente se muestra la película. Figura 9.

Figura 9: En un conjunto de viñetas se tiene la primera representación gráfica de la película que hasta ese momento solo estaba en la cabeza de su creador.

No se trata de un mero documento para mostrar la idea, es una herramienta de trabajo, sobre el storyboard se toman decisiones de cara a perfilar la película, que escenas va a tener, en qué orden, etc. De hecho, una vez que se tienen grabados los diálogos de los personajes, se graba una primera versión de la película superponiendo los diálogos sobre la imagen fija de cada viñeta según corresponda. Es lo que se conoce como Bobina Leyca. Con el visionado de esa primera versión de la película, se puede decidir realizar modificaciones en una etapa muy temprana de la producción y cuando aún no se ha gastado nada de dinero en actores, escenarios, medios técnicos, etc.

En esta url tenéis un ejemplo de bobina leyca:
`https://youtu.be/r8mrfks3rWI`

Como podéis observar, no solo se muestra qué pasa, sino también cuándo pasan las cosas, y a qué ritmo o velocidad ocurren. Y no solo se muestra qué realizan los personajes, también sirve para definir que planos de cámara se usan, cómo se suceden los diferentes planos, etc.

Sobre el storyboard también se hacen anotaciones sobre si se va a hacer zoom, si la luz va a aumentar u oscurecerse, si debe ocurrir algún sonido de fondo, etc. En definitiva, es el documento que contiene el diseño de toda la película, en muchos aspectos de esta.

En cualquier caso, en toda animación por pequeña que sea, como parte fundamental, se debe tener claro lo siguiente:

1. Qué se quiere contar con esa animación.
2. Qué elementos hay que mover o modificar para transmitir ese mensaje. O en qué animaciones más pequeñas se va a descomponer una animación mayor.
3. Cuánto va a durar cada acción.

1.3. Modos de implementar la animación

Una vez se ha diseñado la animación, mediante el guion y el storyboard, se pasa a implementarla. Que en el contexto de esta asignatura, pasa por programar cómo se van a ir modificado los parámetros que varían en la animación según va avanzando el tiempo.

Se puede hablar de tres modos de implementación:

- Animación procedural
  No hay ningún tipo técnica o métodología. Simplemente hay un método que se va a llamar para cada frame (update()) que se encarga de modificar lo que tenga que modificar en función del tiempo que transcurrió desde la última vez que se ejecutó.
- Mediante escenas clave
  Aquí si existe una metodología, basada en definir que valores concretos van a tener diferentes parámetros en los instantes inicial y final de la animación. E indicar cuánto tiempo debe transcurrir entre esos instantes iniciales y finales.
  Alguna biblioteca o clase se encarga de interpolar esos valores en cada frame.

- Mediante caminos
  Este modo es apropiado cuando simplemente se quiere que un elemento recorra una determinada trayectoria.
  Tan solo habría que definir la línea de esa trayectoria y vincular la posición de la figura móvil a dicha trayectoria, además de decidir a qué velocidad se va a mover.

2. Animación procedural

Es el tipo de animación que hemos estado haciendo hasta el momento en prácticas. Cada clase que tiene elementos que se mueven tiene un método `update()` que se encarga de modificar los parámetros que correspondan según el tiempo que ha transcurrido desde la última vez que se ejecutó.

Aunque hasta ahora lo hemos usado de una manera muy básica. Por ejemplo, en el código de figura 10 simplemente se hace girar una figura con respecto al eje Y 0.1 radianes en cada frame, sin ningún control del inicio del giro, de la parada o velocidad del mismo.

```
update() {
  this.figura.rotation.y += 0.1;
}
```

Figura 10: La implementación más básica de una animación.

Una implementación más elaborada de una animación procedural requeriría medir el tiempo, para ello se puede usar la clase `THREE.Clock`, llevar anotado cuándo empezó la animación y en cada `update` medir cuánto tiempo ha transcurrido desde la última vez, o desde el inicio de la animación, para en función de eso, modificar unos parámetros u otros y decidir en cuánto se modifica cada parámetro.

Puede haber incluso una clase `Game` o `Animation` que sería la responsable gestionar las animaciones y llamar a los correspondientes métodos `update` de cada objeto que hubiera que animar.

En el código de ejemplo de la figura 11 se observa como el método `update` de la clase `Game` lo que hace es almacenar en una variable de clase (`Game.deltaTime`) el tiempo que ha transcurrido desde la última vez, y para cada objeto que esté en la lista `gameObjects` llama a su correspondiente método `update`.

Cuando un objeto requiere animación se añade a esa lista, y cuando no, se quita de la lista. En cada frame, solo son llamados los métodos `update` de los objetos que están en

dicha lista. Y cada método `update` de cada objeto ya hará las modificaciones que tenga que hacer según se haya programado.

```
update() {
  Game.deltaTime = this.clock.getDelta();

  this.gameObjects.forEach (function (gameObject) {
    gameObject.update();
  });
}
```

Figura 11: La implementación más básica de una animación.

Con respecto a los métodos `update` de cada clase, tienen toda la responsabilidad de la animación, deben saber qué movimiento está haciendo la figura en ese momento y en qué fase o estado del movimiento se encuentra. Pensar que un movimiento global, caminar, implica movimientos parciales, movimientos independientes de piernas, y pies en un determinado orden.

Es decir, debe saber en qué momento ocurre cada cosa y no menos importante, a qué velocidad.

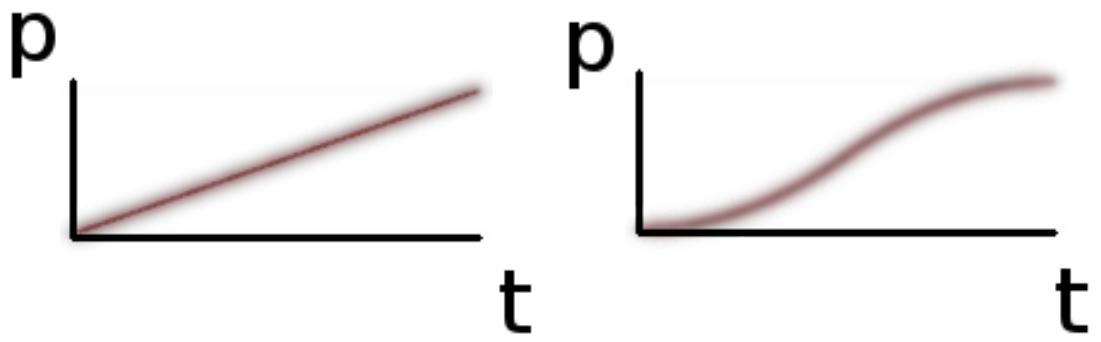
2.1. Curvas de función

Para el control de la velocidad nos ayudamos de las curvas de función. Una función, que recibe como entrada un valor de tiempo y proporciona como salida el valor de un parámetro. Y según varíe el parámetro en función del tiempo, tendremos que el parámetro varía con una u otra velocidad.

La gráfica de la figura 12.a se corresponde con un parámetro que cambia a velocidad constante. La gráfica es una recta. A incrementos de tiempo iguales entre sí, le corresponden incrementos del parámetro iguales entre sí.

En cambio, la gráfica de la figura 12.b se observa como al principio del movimiento, para valores de t bajos, el parámetro apenas cambia o varía muy poco. Luego, en la zona media, la gráfica muestra una mayor variación en el parámetro para volver a ralentizarse la modificación del parámtro cuando t va llegando a su valor más alto.

Es la típica gráfica que controlaría la velocidad de un coche que sale de un semáforo y se detiene en el siguiente semáforo. El coche sale de parado, va cogiendo velocidad poco a poco hasta que llega a la velocidad a la que se va circular, para después, progresivamente ir frenando poco a poco hasta detenerse.



a) b)

Figura 12: Graficas de función que muestran: a) un movimiento a velocidad constante y b) un clásico movimiento lento al salir, lento al llegar.

La mejor forma de interpretar estas gráficas es pensar que son gráficas que indican la velocidad con la que cambia un parámetro con respecto al tiempo, y que esa velocidad viene marcada por la pendiente de la gráfica en cada momento. Y la pendiente en un punto viene determinada por la recta tangente a la gráfica en dicho punto.

Veámoslo de nuevo. En la gráfica de la figura 12.a la pendiente de la gráfica siempre es la misma, luego la velocidad de cambio del parámetro con respecto al tiempo siempre es la misma. Esa figura se mueve a velocidad constante.

En cambio, en la gráfica de la figura 12.b la pendiente de la gráfica al principio es cero, luego la gráfica va aumentando su pendiente poco a poco hasta que en la zona central de la gráfica, la pendiente en prácticamente la misma durante un tramo para después, poco a poco hacerse cero de nuevo. Es decir, la velocidad empieza siendo cero para ir aumentando (acelerando) poco a poco hasta que llega a un tramo en la que la velocidad es prácticamente la misma para después, poco a poco ir disminuyendo (frenando o desacelerando) hasta hacerse cero.

Aquellos que lo necesitéis, repasar en los apuntes de otras asignaturas, lo relacionado con la pendiente de la curva de una función.

La gráfica de función de variación de un parámetro cualquiera puede parecer compleja, pero en realidad no es más que una concatenación de fragmentos de gráficas simples como las que se muestran en la figura 13. Ya que en definitiva, si lo pesamos bien, una figura, con respecto a la velocidad con la que se mueve solo puede estar en uno de estos 4 estados: detenido, acelerando, velocidad constante, o decelerando. De hecho, la gráfica del movimiento que hemos mencionado antes (figura 12.b) con respecto al coche que iba de un semáforo a otro no es más que la concatenación de fragmentos de las gráficas aceleración, velocidad constante y deceleración. En ese orden.

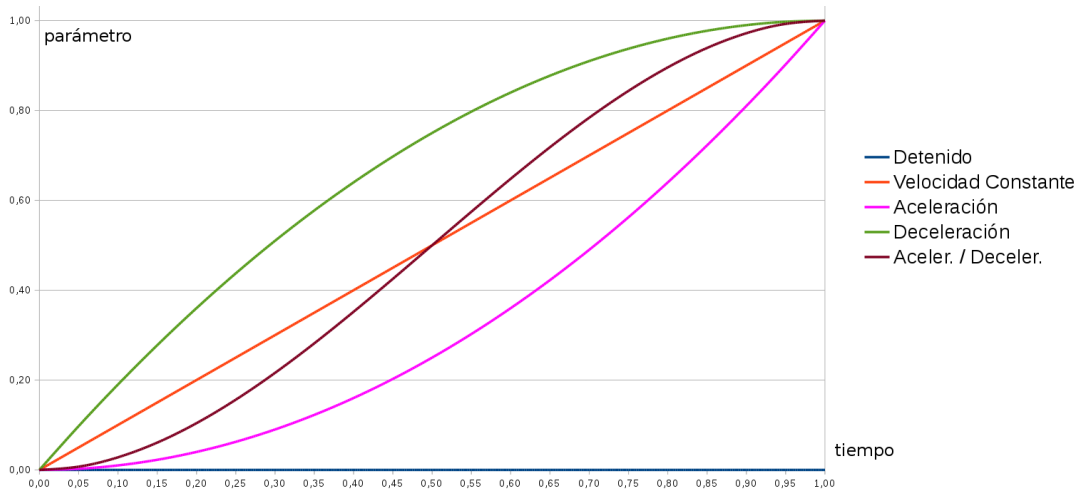


Figura 13: Graficas de función típicas para los 4 estados posibles de una velocidad: detenido (la pendiente siempre es cero, tangente horizontal), velocidad constante (la pendiente siempre es la misma), aceleración (la tangente cada vez va tomando una pendiente mayor) y delecera-ción (la tangente cada vez va teniendo una pendiente menor). También se muestra una gráfica aceleración/deceleración que es una concatenación de fragmentos de las anteriores.

Con todo lo dicho ya estamos en condiciones de implementar una animación en donde un parámetro varíe a una velocidad no constante. Para ello definiremos variables auxiliares que tomando el tiempo que nos proporciona el sistema, el cual siempre es constante, obtengamos una variable que represente también el tiempo que ha transcurrido en una animación pero que vaya variando según la velocidad que deseamos.

Partimos de las siguientes premisas:

- Sabemos que animación queremos hacer, y eso implica:
- Sabemos que un parámetro p va a variar entre v_0 y v_1
- Eso va a ocurrir entre el tiempo t_0 y el tiempo t_1

Con esos tiempos concretos de inicio y fin, calculamos un valor de tiempo λ normalizado que varía entre 0 y 1. Es decir, cuando $\lambda = 0$ estamos al principio de la animación ($t = t_0$) y cuando $\lambda = 1$ estamos al final de la animación ($t = t_1$).

λ se calcula así,

$$\lambda = \frac{t - t_0}{t_1 - t_0}$$

no es más que un valor que nos indica qué porcentaje de la animación llevamos.

Con eso, la expresión $p = v_0 + \lambda \cdot (v_1 - v_0)$ nos daría el valor del parámetro en cada momento cumpliendo con que cuando $\lambda = 0, p = v_0$, y cuando $t = t_1, p = v_1$, aunque eso sí, a velocidad constante, ya que el tiempo t siempre avanza a velocidad constante y tal como hemos calculado λ, también avanza a velocidad constante.

Para tener una variación de velocidad distinta solo debemos definir una función $f(\lambda)$ que tenga la gráfica de variación de velocidad deseada. Las únicas condiciones que debe cumplir $f(\lambda)$ son:

$$f(0) = 0$$

$$f(1) = 1$$

Que no es más que decir que, por mucho que varíe la velocidad en medio, en los extremos debemos estar siempre al $0\,\%$ y al $100\,\%$ respectivamente.

Algunos ejemplos de funciones $f(\lambda)$ para distintos tipos de velocidad son los siguientes.

- Velocidad constante: $f(\lambda) = \lambda$
- Aceleración: $f(\lambda) = \lambda^2$
- Deceleración: $f(\lambda) = -\lambda^2 + 2 \cdot \lambda$
- Aceleración, seguido de deceleración: $f(\lambda) = -2 \cdot \lambda^3 + 3 \cdot \lambda^2$

En concreto, esas funciones se corresponden con las gráficas de la figura 13.

Así, **las expresiones para calcular el valor de un parámetro** p que debe variar entre v_0 cuando $t = t_0$ y v_1 cuando $t = t_1$ con unos determinandos cambios de velocidad son:

$$\lambda = \frac{t - t_0}{t_1 - t_0}$$

$$p = v_0 + f(\lambda \cdot)(v_1 - v_0)$$

Dependiendo los cambios de velocidad de cómo esté definida $f(\lambda)$.

2.2. Velocidad independiente del ordenador

Lo que nunca debe ocurrir es que la misma animación se vea más rápido en un ordenador que sea más rápido o que esté menos cargado y se muestre más lenta en un ordenador más lento o que tenga una carga de procesos mayor.

Se debe controlar la velocidad a la que se modifican los parámetros para que la animación siempre vaya igual de rápida, con independencia del ordenador o del momento en que se ejecute.

Si aplicamos las expresiones de la sección anterior no habrá ningún problema ya que en esas expresiones se está teniendo en cuenta el tiempo del sistema y precisamente se están usando funciones que definen una velocidad.

El problema surgirá si implementamos una animación según una expresión del tipo

$$figura.position.x \quad += \quad 1;$$

Donde un ordenador que sea capaz de renderizar 30 frames por segundo hará avanzar esa figura a 30 unidades por segundo pero un ordenador que sea capaz de renderizar 60 frames por segundo hará avanzar esa figura a 60 unidades por segundo, el doble de rápido.

Para evitar eso recurrimos a la ya conocida ecuación de la cinemática

$$e = v \cdot t$$

El espacio que recorre un objeto móvil es el resultado de multiplicar la velocidad a la que se mueve por el tiempo que está en movimiento.

En realidad, cuando hacemos $figura.position.x \quad += \quad 1;$ estamos implementando la ecuación de la cinemática

$$e = e' + \Delta e$$

al espacio que ya teníamos le añadimos un incremento de espacio ($e \quad += \quad \Delta e$).

Pues ese incremento de espacio no puede ser un valor fijo, sino que se debe calcular teniendo en cuenta la velocidad deseada y el tiempo que ha transcurrido desde la última vez que se aplicó un incremento.

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

Así, en un ordenador que sea muy rápido y que ejecute esa ecuación más veces, calculará un Δt más pequeño y hará incrementos más pequeños. Y un ordenador que sea más lento, la ejecutará menos veces, calculará un Δt más grande y hará incrementos mayores.

El ordenador rápido hará muchos incrementos, pero pequeños y el ordenador lento hará menos incrementos pero más amplios. El resultado es que el usuario observará que la figura se desplaza a la misma velocidad.

Una vez conocido esto, la implementación en un lenguaje no tiene mayor problema. Al construir la figura guardamos el tiempo actual en el momento de la construcción y definimos la velocidad deseada, en unidades/segundo para movimientos lineales y en radianes/segundo para movimientos giratorios. Ver el código de la figura 14.

```
// Al crear el objeto medimos el tiempo actual nedido en milisegundos
this.tiempoAnterior = Date.now();

// Se tiene en un atributo la velocidad
//    (expresada como unidades / segundo)
this.velocidad = 10;

// El valor de la velocidad tambien podria leerse de una GUI
```

Figura 14: Control de la velocidad independiente del ordenador. Parte a implementar en el constructor.

Y en el método `update` solo tenemos que medir el incremento de tiempo para calcular el incremento de espacio según la velocidad y actualizar el parámetro en consecuencia. Ver el código de la figura 15.

```
// En el metodo update(), que se ejecuta en cada frame y actualiza el objeto
var tiempoActual = Date.now(); // Recuerda, son milisegundos

var segundosTranscurridos = (tiempoActual-this.tiempoAnterior)/1000;

// Se divide por 1000 para pasar el tiempo a segundos.
// La velocidad se mide en unidades/segundo
//    porque es más intuitivo que medirla en unidades/ms

this.objeto.position.x += this.velocidad * segundosTranscurridos;

// No olvidarse de que el tiempo actual de ahora
//    será el tiempo anterior del siguiente frame
this.tiempoAnterior = tiempoActual;
```

Figura 15: Control de la velocidad independiente del ordenador. Parte a implementar en el actualizador.

3. Animación mediante escenas clave

Es el modo de implementar animaciones más usado. Deriva directamente de la animación convencional, que como veíamos al principio del tema, se basa en que el animador

define momentos importantes en los movimientos, escenas clave, y los intercaladores realizan las imágenes intermedias que completan la animación.

En el caso de la animación por ordenador el procedimiento es el mismo salvo que en vez de dibujar, **las escenas clave son definidas en base a darle valores concretos a los parámetros involucrados en cada animación** para cada instante de tiempo en el que se sitúe cada escena clave.

Así, la animación a realizar, en caso de ser compleja se descompone en movimientos más simples. Para cada movimiento simple se determinan las dos escenas clave que lo definen: una situación que se corresponde con el inicio del movimiento y otra que se corresponde con el fin del movimiento. La definición de dicho movimiento se completa indicando el tiempo que debe transcurrir entre la escena clave de inicio y la escena clave de fin. También se le puede configurar la curva de función que controlará los cambios de velocidad del movimiento.

Una vez definidos todos estos movimientos parciales, la animación global se obtiene encadenando movimientos parciales.

3.1. Implementación en Three.js

Se usará la biblioteca Tween.js, la cual se ha proporcionado para las prácticas en Prado y que también es descargable de `https://github.com/tweenjs/tween.js`

Acordaos de incluirla en el `index.html`.

Veamos su uso con un ejemplo. Se desea desplazar una figura entre la posición (0,300,0) y la posición (400, 50, 0); que emplee en ese movimiento 2 segundos y que realice ese desplazamiento a un ritmo "lento al salir, rápido en medio, y lento al llegar". Las dos escenas clave son las que se corresponden con que el parámetro posición de la figura tenga esas coordenadas al principio y al final del movimiento respectivamente.

La figura se construye como es habitual.

```
this.figura = new THREE.Mesh ( . . . );
```

Una vez tenemos pensada la animación, se necesitan dos variables de tipo diccionario definidas sobre los mismos parámetros pero con distintos valores, y en este caso, como solo se modifican dos parámetros, la posición x e y de la figura, las variables diccionario tendrán solo dos parámetros, que pueden llamarse igual x e y aunque no es obligatorio que se llamen igual.

Las dos variables diccionario con los mismos parámetros pero con distintos valores,

como os podéis imaginar, representan a la escena clave inicio del movimiento y a la escena clave fin del movimiento.

```
var origen  = { x :   0, y : 300 };
var destino = { x : 400, y :  50 };
```

En el origen del movimiento, los parámetros valen (0,300) y en el final del movimiento, los parámetros valen (400, 50).

Ya solo nos queda definir el movimiento, configurarle el tiempo, los cambios de velocidad y usar los parámetros de esas variables auxiliares para modificar los parámetros de la figura a animar.

En la definición del movimiento intervienen la variable diccionario origen, la variable diccionario destino y el tiempo que debe trascurrir entre origen y destino, expresado en milisegundos.

```
this.movimiento = new TWEEN.Tween(origen).to(destino, 2000);
```

Le configuramos los cambios de velocidad usando la constantes que la biblioteca TWEEN proporciona.

```
this.movimiento.easing(TWEEN.Easing.Quadratic.InOut);
```

Si se quiere que una animación tenga velocidad constante bastaría con no configurar la animación con el método `easing` o llamarlo con el parámetro `TWEEN.Easing.Linear.None`. Si se quiere que solo sea acelerado, en la constante se cambia `InOut`, por `In`. Es decir, la constante sería `TWEEN.Easing.Quadratic.In`, si se quiere que solo sea decelerado, se cambia `InOut` por `Out`. Si se quiere una aceleración mayor, se puede cambiar `Quadratic` por `Cubic`, `Quartic`, `Quintic` o `Exponential`. Por ejemplo, si se quisiera un movimiento solo acelerado pero con una aceleración muy enérgica la constante sería `TWEEN.Easing.Exponential.In`.

Por supuesto, tenemos que indicar cómo vamos a modificar los parámetros de la figura a partir de los parámetros del diccionario del objeto de animación tween.

Se usa la referencia `that` para acceder a la figura y la variable diccionario, `origen` en este caso, para acceder a los parámetros del diccionario.

```
var that = this;
this.movimiento.onUpdate (function () {
  that.figura.position.x = origen.x;
  that.figura.position.y = origen.y;
});
```

Cada vez que se ejecute la animación se quedarán los parámetros del diccionario con sus valores finales. En previsión de que se quiera volver a ejecutar esta animación, es recomendable programar que dichos valores de los parámetros del diccionario se restablezcan a sus valores iniciales. Lo hacemos así:

```
this.movimiento.onComplete (function () {
  origen.x =   0;
  origen.y = 300;
});
```

Ya tenemos la animación definida, que por cierto, puede hacerse en el constructor. Es decir, en el constructor pueden definirse todas las animaciones que se estime que van a ser necesarias a la espera de que sean lanzadas por otros métodos cuando sea requerido.

Cuando se quiera lanzar la animación solo es necesario ejecutar

```
this.movimiento.start();
```

Solo nos quedaría una última cosa, en el método `update` de la clase principal de la aplicación, en vez de llamar a los métodos `update` de las figuras, se llamaría exclusivamente al método `update` de TWEEN, es decir:

```
// En el update de la clase principal
TWEEN.update();
```

Es la clase `TWEEN` la que se encarga de gestionar las animaciones que están activas en cada momento y llamar a las funciones `onUpdate` de dichas animaciones.

Otros métodos para configurar las animaciones Tween

Por defecto, cada animación se ejecuta una sola vez cada vez que se lanza, pero si se desea que se ejecute varias veces seguidas se puede programar con el método **`repeat`**.

```
this.unaAnimacion.repeat (2);
// La animacion anterior, cuando se lance se ejecutara 3 veces.
//    La vez por defecto y 2 veces mas.

this.otraAnimacion.repeat (Infinity);
// Esta animacion, cuando se lance se ejecutara indifinidamente
```

Si se desea que una animación sea de ida y vuelta, se programará con el método **`yoyo(true)`**.

```
this.otraAnimacion.yoyo (true);
// Esta animacion, cuando se lance se ejecutara dos veces
//   Una vez desde origen a destino y otra vez desde destino a origen
```

El método `repeat(Infinity)` se puede combinar con el método `yoyo(true)`. Imaginad que queréis programar la animación del péndulo de un reloj de péndulo. Ver el código de la figura 16.

```
var penduloReloj = // Se crea este Mesh como se estime conveniente
this.reloj.add (penduloReloj); // Y se anade al grafo

// Variables diccionario con un parametro
var vaivenInicio = { x : -0.5 };
var vaivenFinal  = { x :  0.5 };

// Definicion de la animacion
var animacionPendulo = new TWEEN.Tween (vaivenInicio).to (vaivenFinal, 1000)
  .easing (TWEEN.Easing.Quadratic.InOut)
  .repeat (Infinity)
  .yoyo (true)
  .onUpdate (function () {
     penduloReloj.rotation.z = vaivenInicio.x;
  })
  .start();
```

Figura 16: Programación del movimiento de vaivén del péndulo de un reloj.

Todos los métodos de un objeto Tween se pueden encadenar, y en cualquier orden, como se muestra en el código. En esta ocasión, como se quiere que la animación esté funcionando desde el principio, se lanza desde el propio constructor encadenando el método `start` como último método en la cadena de métodos de configuración. Mientras la aplicación se esté ejecutando, el péndulo estará oscilando medio radian hacia cada lado, empleando 1 segundo en cada recorrido y su velocidad será más lenta en los extremos del recorrido y más rápida en el centro.

Cuando se quiera parar una animación solo hay que enviarle el mensaje **`stop()`**.

Por último, un método muy útil, que nos permite encadenar una animación a otra, de manera que cuando una finalice comience la otra. El método **`chain`**.

```
var animacion1 = new TWEEN.Tween ( ... );
var animacion2 = new TWEEN.Tween ( ... );
var animacion3 = new TWEEN.Tween ( ... );
var animacion4 = new TWEEN.Tween ( ... );

animacion1.chain (animacion2);
animacion1.chain (animacion3);
animacion2.chain (animacion4);

animacion1.start();
```

Figura 17: Una vez se han definido y encadenado varias animaciones, solo hay que lanzar la primera. Las siguientes irán ejecutándose según finalicen las anteriores.

Según el código de la figura 17, se ejecutará primero la animación 1, cuando finalice se lanzarán en paralelo las animaciones 2 y 3, y cuando finalice la 2 se lanzará la 4.

Con este mecanismo es fácil realizar animaciones complejas si se dividen en partes más simples y se encadenan adecuadamente.

4. Animación mediante caminos

Esta técnica se utiliza cuando una figura debe moverse siguiendo una trayectoria arbitraria. Consiste en definir dicha trayectoria, normalmente mediante una curva spline y se hace que tanto la posición como la orientación de la figura sigan dicha curva.

En la url `https://threejs.org/examples/#webgl_geometry_extrude_splines` tenéis un ejemplo. En dicho ejemplo lo que se anima siguiendo un camino es una cámara, lográndose el efecto de una cámara subjetiva en el vagón de una montaña rusa. Para ver dicha cámara subjetiva, en la GUI del ejemplo, tenéis que hacer clic en la opción *animationView*.

La curva spline se define usando la clase `CatmullRomCurve3` pasándole la lista de puntos de paso definidos como `Vector3`. El resultado es una línea curva que pasa por todos los puntos pero de una manera suave, sin ángulos, como se observa en la figura 18.

Si se desea dibujar la línea spline, hay que tener en cuenta que la visualización de una línea se realiza aproximándola mediante una polilinea de segmentos rectos. Es similar a cuando se define una esfera, que matemáticamente es una superficie curva y se visualiza aproximándola mediante un poliedro de poligonos planos, en una cantidad mayor o menor dependiendo de la

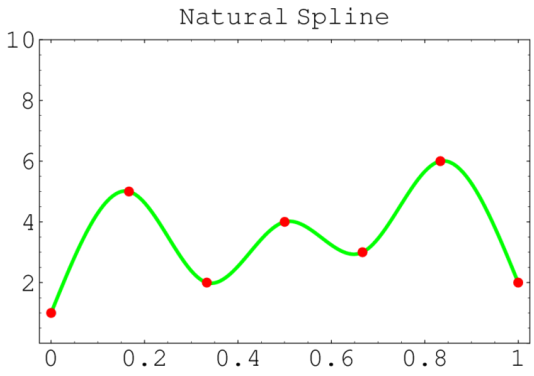


Figura 18: La curva spline se caracteriza por pasar por todos los puntos de control de una manera suave.

resolución. Por tanto, en la visualización de una curva spline se debe indicar una resolución. Tenéis un ejemplo en el código de la figura 19.

```
// Se define el spline con unos pocos puntos de control
var spline = new THREE.CatmullRomCurve3 ([
  new THREE.Vector3 (0, 0, 0), new THREE.Vector3 (0, 1, 0), ... ]);

// Si se quiere visualizar, se visualiza como una linea de segmentos rectos
var geometryLine = new THREE.Geometry();

// Se toman los vertices del spline, en este caso 100 muestras
geometryLine.vertices = spline.getPoints(100);

// Se crea una linea visible con un material de linea
var material = new THREE.LineBasicMaterial ({color: 0xff0000});
var visibleSpline = new THREE.Line (geometryLine, material);
escena.add (visibleSpline);
```

Figura 19: La curva se define con unos pocos puntos de control, pero para visualizarla se le debe dar una resolución más o menos alta.

Si se quiere usar esa línea spline para que una figura se mueva por dicha trayectoria, se debe ir evaluando la curva para obtener una posición de la curva en cada momento e ir colocando la figura en dicha posición. También, usando la tangente de la curva en dicha posición, se debe actualizar la orientación de la figura. Pensad en un coche que circula por una carretera, no solo cambia su posición, sino también su orientación siguiendo las curvas de la carretera, unas veces el morro del coche apunta hacia el norte, otras veces al sureste, etc.

Se van a necesitar, por tanto, dos puntos, ver la figura 20. Un primer punto en la curva para posicionar la figura (el punto azul), y un segundo punto en la dirección de la tangente para que la figura *mire* a dicho punto y así se oriente adecuadamente (el punto rojo).

Visto esto, el código para implementar una animación mediante caminos, es bastante sencillo, figura 21

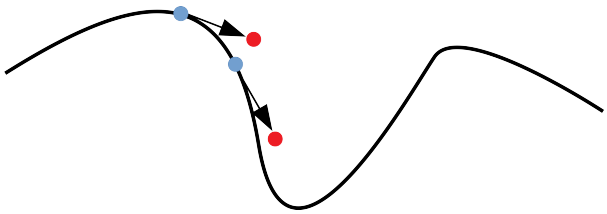


Figura 20: La posicion de la figura en cada instante viene marcada por los puntos azules. La orientación de la figura viene marcada por los puntos rojos, que se obtienen sumándole a los puntos azules, la tangente de la curva en dichos puntos.

```
// Se necesita un parametro entre 0 y 1
//    Representa la posicion 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;
// t siempre va a estar entre 0 y 1

// Se coloca y orienta el objeto a animar
// Se obtiene del spline el punto que se encuentra en el lugar t de la curva
// Es el punto azul en la imagen anterior
var posicion = spline.getPointAt (t);

// Se usa para actualizar el atributo position de la figura mediante una copia
object.position.copy (posicion);

// Se obtiene del spline la tangente en el lugar t de la curva
var tangente = spline.getTangentAt (t);
// Se le suma al punto anterior para obtener el punto rojo
posicion.add (tangente);

// Se le dice a la figura que mire al punto azul
object.lookAt (posicion);

// IMPORTANTE: Lo que se alinea con la tangente es la Z positiva del objeto
```

Figura 21: La figura del ejemplo, `object`, recorre un determinado camino, `spline`, en un ciclo infinito, empleando 20 segundos en cada ciclo.

Combinando técnicas

En una aplicación, no todas las animaciones deben hacerse con la misma técnica. Obviamente, hay que usar unas u otras según interese.

Por ejemplo, imaginad que se quiere animar una figura para que recorra una trayectoria pero se desea que empiece desde parado y vaya acelerando poco a poco para que cuando se aproxime al final vaya decelerando progresivamente hasta detenerse (animación por caminos). Se puede usar una animación Tween (animación por escenas clave) cuyo parámetro del diccionario varíe entre 0 y 1. Se configura la velocidad (método easing) para que cambie según esos

requisitos. Y se usa ese parámetro, que varía entre 0 y 1 con esa característica de velocidad, para posicionar y orientar la figura en la trayectoria.

Se pueden tener las animaciones Tween (animación por escenas clave) previamente creadas y que desde una clase que controla las animaciones de la aplicación (animación procedural), se van lanzando y deteniendo las animaciones previamente definidas.

Se puede tener una animación Tween que representa a una animación compleja que tiene solo un parámetro en el diccionario que varía entre 0 y 1. Ese parámetro no es más que un valor que nos va a indicar qué porcentaje de esa animación global a transcurrido, y en función de eso lanzar o detener otras animaciones más sencillas que forman parte de la animación compleja.

En fin, no son más que un conjunto de técnicas para usarlas adaptándolas y combinándolas según el problema concreto que haya que resolver en cada caso.