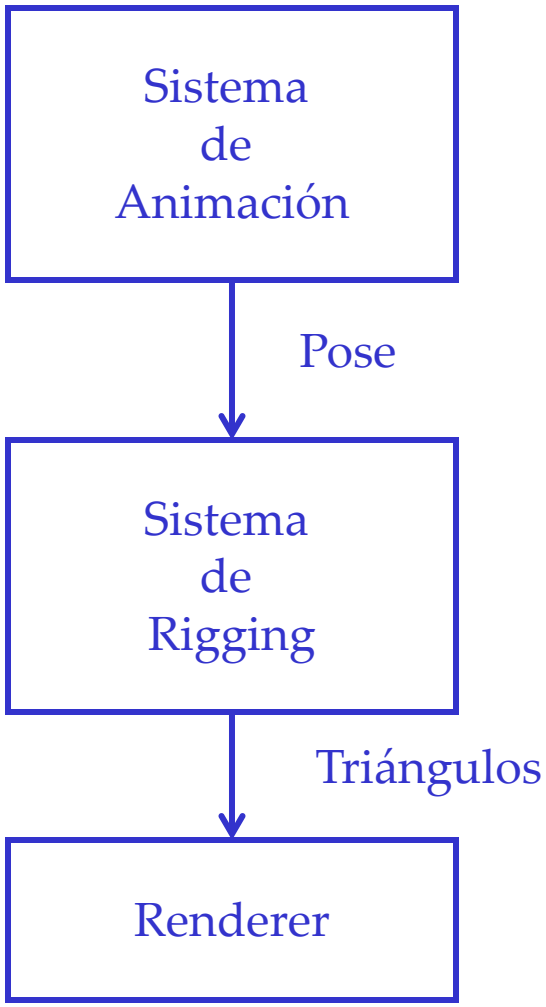

Universidad de Huelva

# Tema 13

# Canales y fotogramas clave

# Índice

## Índice




$$\mathbf{\Phi} = \begin{bmatrix} \phi_1 & \phi_2 & \dots & \phi_N \end{bmatrix}$$

- Cuando hablamos de una "animación", nos referimos a los datos necesarios para mostrar un *rig* sobre un rango de tiempo.
- Esto debe incluir información para especificar todos los valores de los GDL necesarios durante todo el intervalo de tiempo.
- A veces, esto se conoce como un "*clip*" o incluso una "*move*" (del inglés *movie*) (debido a que "animación" puede ser un termino ambiguo).

## Espacio de poses

- Si un personaje tiene N GDLs, entonces una pose puede ser vista como un punto en el espacio de poses N-dimensional

$$\mathbf{\Phi} = \begin{bmatrix} \phi_1 & \phi_2 & \dots & \phi_N \end{bmatrix}$$

- Una animación puede ser vista como un punto moviéndose a través del espacio de poses, o alternativamente como una curva fijada en el espacio de poses.

$$\mathbf{\Phi} = \mathbf{\Phi}(t)$$

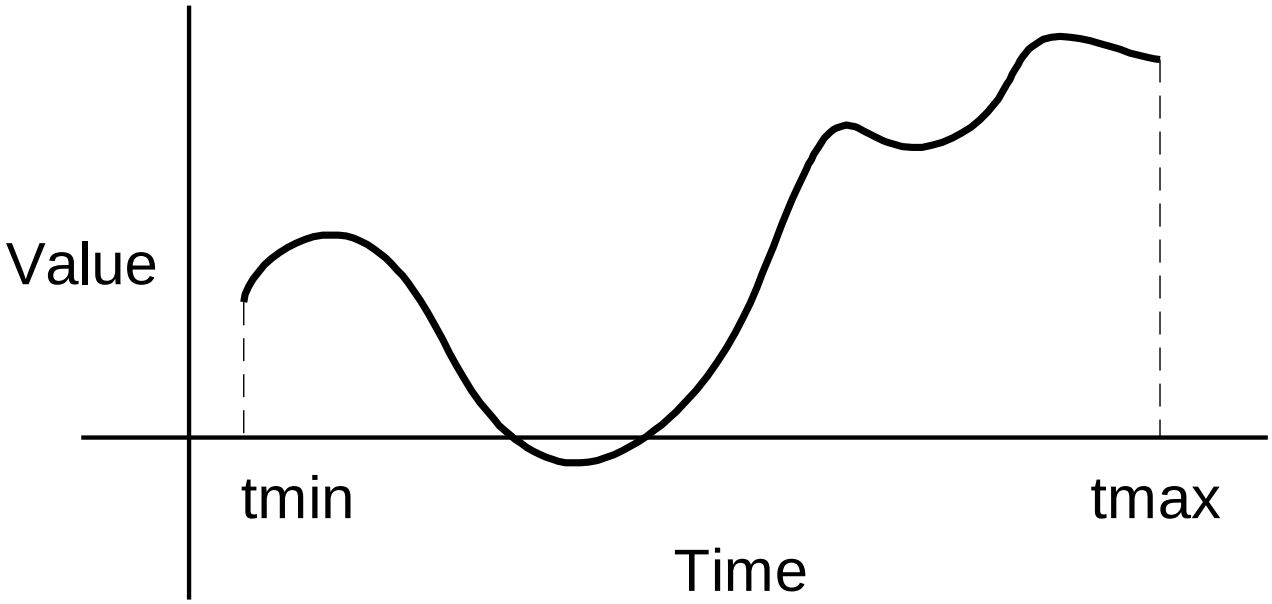
- Las animaciones '*One-shot*' son curvas abiertas, mientras que las animaciones '*loop animations*' forman una curva cerrada

- Generalmente, se puede pensar que una animación individual es como una curva continua, sin embargo podríamos tener también discontinuidades (cortes).

- Si toda la animación es una curva N-dimensional en el espacio pose, podemos separar esta en N curvas 1-dimensionales, uno para cada GDL.

$$\phi_i = \phi_i(t)$$

- A esto se le llama '*channels*' (canales).
- Un canal almacena el valor de una función escalar sobre algún dominio 1D (ya sea finito o infinito).
- Un canal se referirá a los datos pre-grabados o pre-animados para un GDL y no se refiere al caso más general de un GDL cambiando con el tiempo (el cual incluye la física, la animación de procedimiento ...).


Value
tmin
tmax
Time

- Como un canal representa un dato pre-grabado, la evaluación del canal para un particular valor debería siempre devolver el mismo resultado.
- Se permiten canales discontinuos en valor, pero no en tiempo.
- La mayoría de la veces, un canal será usado para representar un GDL cambiando durante el tiempo, pero ocasionalmente se usará la misma tecnología para relacionar alguna variable arbitraria con alguna otra variable arbitraria (por ejemplo, par de torsión (par motor) frente a la curva de RPM de un motor ...)

## Array de canales

- Una animación puede ser almacenada como un array de canales.
- Un medio sencillo de almacenar un canal es como una matriz de muestras regularmente espaciados en el tiempo.
- Usando esta idea, uno puede almacenar una animación como un array 2D de floats (NumGDLs x NumFrames).
- Sin embargo, si se quisiera utilizar algún otro medio de almacenamiento de un canal, se podría almacenar una animación como un conjunto de canales, donde cada canal se encarga de almacenar los datos si se quiere.

## Array de poses

- Un camino alternativo para almacenar una animación es como un array de poses.
- Esto también forma un array 2D de floats (NumFrames x NumGDLs).
- ¿Qué es mejor, poses o canales?

## Poses vs canales

- ¿Cual es mejor?
- Depende de los requerimientos.
- El resultado final:
  - Las poses son mas rápidas.
  - Los canales son mucho mas flexibles y pueden potencialmente usar menos memoria.

## Array de poses

- El método de matriz de poses se trata de la forma más rápida posible de reproducir datos de animación.
- Una 'pose' (vector de floats) es exactamente lo que se necesita con el fin de representar un *rig*.
- Los datos son contiguos en la memoria, y todo se puede acceder directamente desde una dirección.

## Array de canales

- Como cada canal se almacena de forma independiente, tienen la flexibilidad para aprovechar las diferentes opciones de almacenamiento y maximizar la eficiencia de la memoria.
- Además, en una situación de edición interactiva, nuevos canales pueden ser creados y manipulados de forma independiente.
- Sin embargo, necesitan ser evaluados de forma independiente para acceder al 'fotograma actual', lo que lleva tiempo e implica el acceso a memoria discontinua.

## Poses vs canales

- Array de poses es ideal si sólo se necesita para reproducir una animación relativamente simple y necesita el máximo rendimiento. Esto se corresponde con muchos videojuegos.
- Array de canales es esencial si se quiere flexibilidad para un sistema de animación o se está interesado en la generalidad sobre el rendimiento bruto.
- Array de canales también puede ser útil en situaciones de juego más sofisticadas o en los casos donde la memoria es más crítica que el rendimiento de la CPU (que no es raro).

- Como el método de array de poses es muy simple, no hay mucho más que decir al respecto.
- Por lo tanto, nos concentraremos en los canales en sus diversas técnicas de almacenamiento y manipulación.

## Continuidad temporal

- A veces, pensamos en animaciones como tener una velocidad de fotogramas en particular (es decir, 60 fps).
- A menudo es una mejor idea de pensar en ellos como un proceso continuo en el tiempo y no vinculado a un velocidad en particular. Algunas de las razones son:
  - Conversión de formato de vídeo (Film / NTSC / PAL)
  - Manipulación instantánea (estiramiento / contracción)
  - El desenfoque de movimiento (Motion blur)
- Ciertos efectos (y movimientos rápidos) pueden requerir que uno sea realmente consciente de fotogramas individuales, aunque …

## Almacenamiento

- Independientemente de si uno piensa en una animación como un proceso continuo o por tener puntos discretos, uno debe considerar métodos de almacenamiento de datos de animación.
- Algunos de estos métodos pueden requerir algún tipo de discretización temporal, mientras que otros no.
- Incluso cuando almacenamos un canal en incrementos de frames, todavía se puede pensar como una función continua interpolando el tiempo entre frames.

- Podemos pensar en términos de clase para describir el proceso de animación.

```
class AnimationClip
{
        void Evaluate(float time, Pose &p);
        bool Load(const char *filename);
};


class Channel
{
        float Evaluate(float time);
        bool Load(FILE*);
};
```

## Almacenamiento

- Hay varias maneras de almacenar canales. La mayoría de los enfoques utilizan un almacenamiento de frames "en bruto", o como curvas de interpolación a trozos (fotogramas clave o keyframes).
- Una tercera alternativa es como una expresión proporcionada por el usuario, que es sólo una función matemática arbitraria. En la práctica, esto no es muy común, pero puede ser útil en algunas situaciones.
- También se podría aplicar diversos esquemas de interpolación (como funciones de base radial), pero la mayoría de los métodos de canal están diseñados más en torno a la interactividad del usuario.

## Almacenamiento

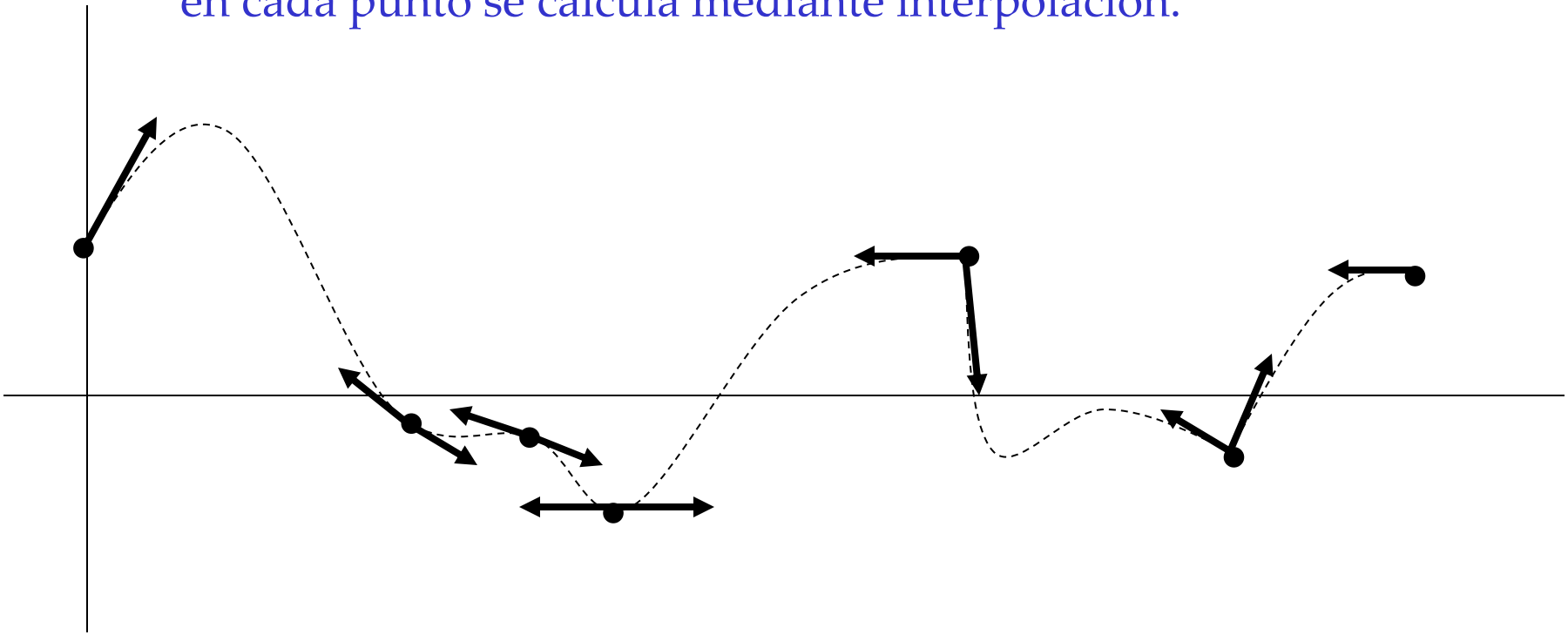
- A veces, los canales se almacenan simplemente como un conjunto de valores regularmente espaciados en el tiempo, en algún tipo de *frame rate* (velocidad de frame).
- Pueden usar interpolación lineal o suave para evaluar la curva entre los puntos de muestreo.
- Los valores son generalmente floats pero se puede comprimir más si se desea.
- La velocidad de los frames es generalmente similar a la tasa de fotogramas de reproducción final, pero podría ser menos si es necesario.

## Compresión de datos de canales

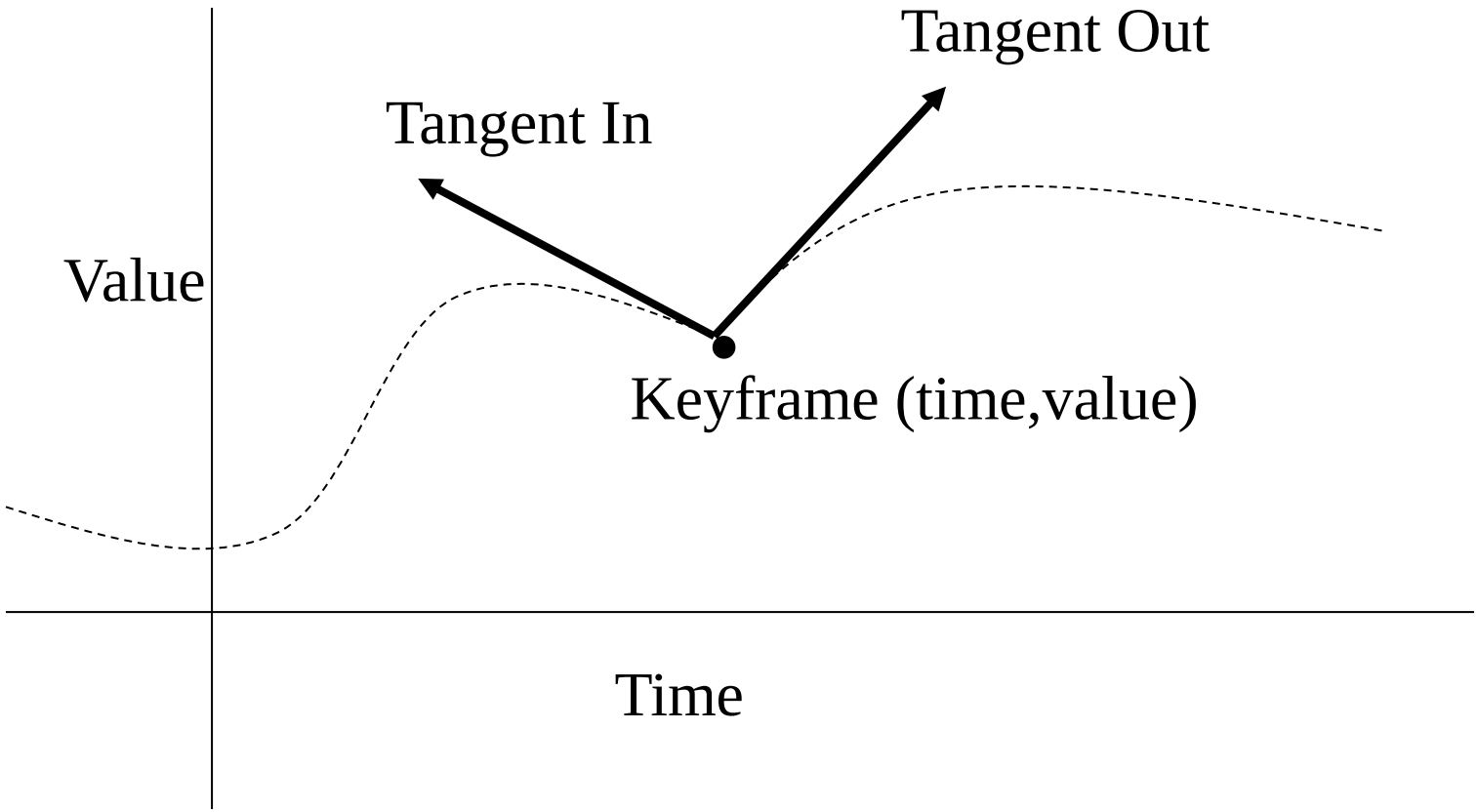
- Las rotaciones se pueden comprimir a 16 bits con razonable fidelidad.
- Las traslaciones se pueden comprimir similarmente si no van demasiado lejos del origen.
- También se puede almacenar un valor float mínimo y máximo por canal y almacenar un valor de punto fijo por frame que interpole entre min y max.
- La reducción de la velocidad de frames también puede ahorrar mucho espacio, pero sólo se puede hacer para animaciones suaves.
- Se podría utilizar un algoritmo automático para comprimir cada canal de forma individual sobre la base de las tolerancias especificadas por el usuario. Tambien se pueden comprimir 'Raw channels' usando algún tipo de *compresión delta*.

- Un canal puede ser almacenada como una secuencia de fotogramas clave (*keyframes*).
- Cada fotograma clave tiene un tiempo y un valor y por lo general una descripción de las tangentes en ese lugar.
- Las curvas de los *spans (espacios)* individuales entre las claves se definen por interpolación 1-D (por lo general a trozos de Hermite).

- Cada punto de la gráfica es un *keyframe* y define una posición y las tangentes a la izquierda y a la derecha.
- Dos *keyframes* consecutivos definen un span. El valor del canal en cada punto se calcula mediante interpolación.



- La información asociada a un keyframe es el instante, el valor del canal, la tangente a la izquierda y la tangente a la derecha.



- Los fotogramas clave se dibujan generalmente de manera que las tangentes de entrada apunten a la izquierda (anterior en el tiempo).
- La flecha dibujada es sólo para la representación visual y hay que recordar que si las dos flechas son exactamente opuestos, que en realidad significa que las tangentes son iguales.
- También recordar que sólo estamos tratando con curvas 1D ahora, así que la tangente en realidad tiene sólo una pendiente.

## ¿Por qué usar keyframes?

- Buena interfaz de usuario para el ajuste de curvas.
- Le da al usuario control sobre el valor del GDL y la velocidad del GDL.
- Definir una función perfectamente suave (si se desea).
- Puede ofrecer una buena compresión (no siempre).
- Cada sistema de animación ofrece algunas variaciones sobre el keyframing.
- Los videojuegos pueden considerar el uso de keyframes para compresión de la animación, aunque ello tenga un elevado coste de ejecución.

- Los canales de fotogramas clave son la base de la animación de propiedades (GDL) en muchos sistemas de animación comercial.
- Diferentes sistemas utilizan diferentes variaciones en los cálculos exactos, pero la mayoría se basan en algún tipo de curvas de Hermite cúbicas.

- Los fotogramas clave se pueden generar automáticamente a partir de los datos incluidos en el muestreo, tales como la captura de movimiento.
- Este proceso se llama "curva de ajuste", ya que implica la búsqueda de las curvas que se ajustan a los datos razonablemente bien.
- Algoritmos de ajuste permiten al usuario especificar las tolerancias que definen la calidad aceptable del ajuste.
- Esto permite la conversión bidireccional entre *keyframes* y los *raw formats*, aunque los datos podrían quedar ligeramente distorsionados con cada traducción.

- En términos de clase podríamos definir los datos necesarios para interpolar fotogramas clave como

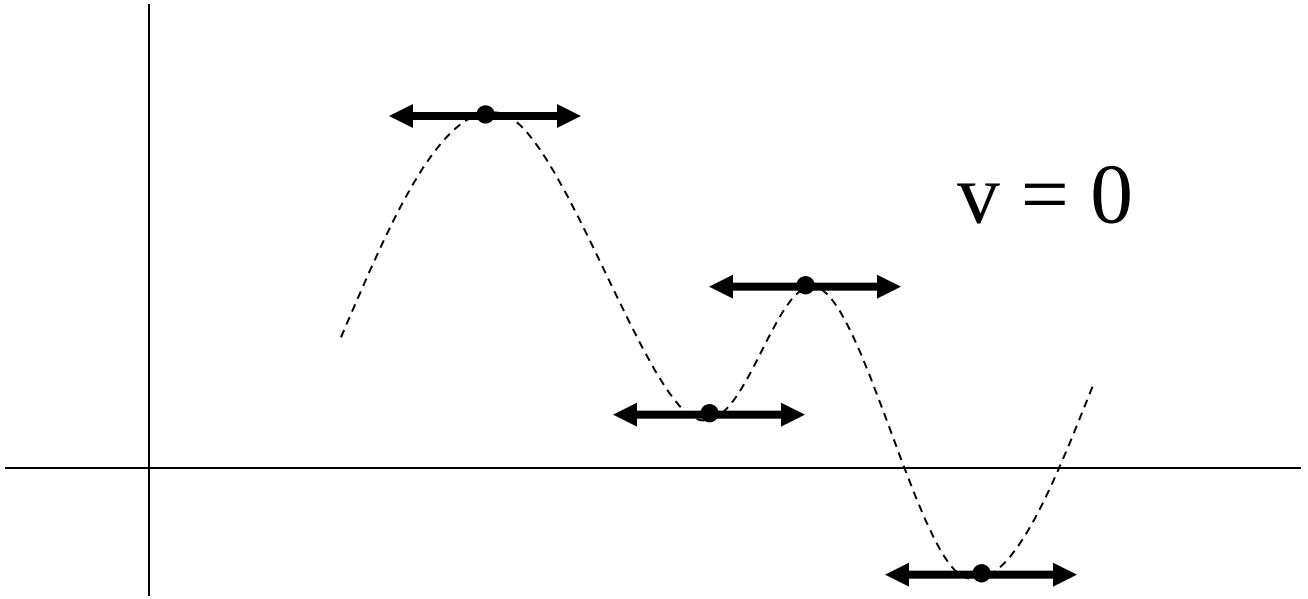
```
class Keyframe {
    float Time;
    float Value;
    float TangentIn,TangentOut;
    char RuleIn,RuleOut;        // Tangent rules
    float A,B,C,D;              // Cubic coefficients
}
```

- La estructura de datos para representar un canal completo podría ser:
  - Listas enlazadas
  - Listas doblemente enlazadas
  - Array

- En lugar de almacenar explícitamente los números para las tangentes, a menudo es más conveniente almacenar una "regla" y volver a calcular la tangente real cuando sea necesario.
- Por lo general, las reglas se almacenan separadas de las tangentes de entrada y salida.
- Reglas comunes para tangentes Hermite:
  - Flat (tangente = 0)
  - Lineal (puntos tangentes a la clave siguiente/anterior)
  - Smooth (ajustar automáticamente la tangente para resultados suaves)
  - Fijo (el usuario puede especificar arbitrariamente un valor)
- NOTA: La tangente es la tasa de cambio de la GDL (o velocidad). La denotaremos como 'v'.

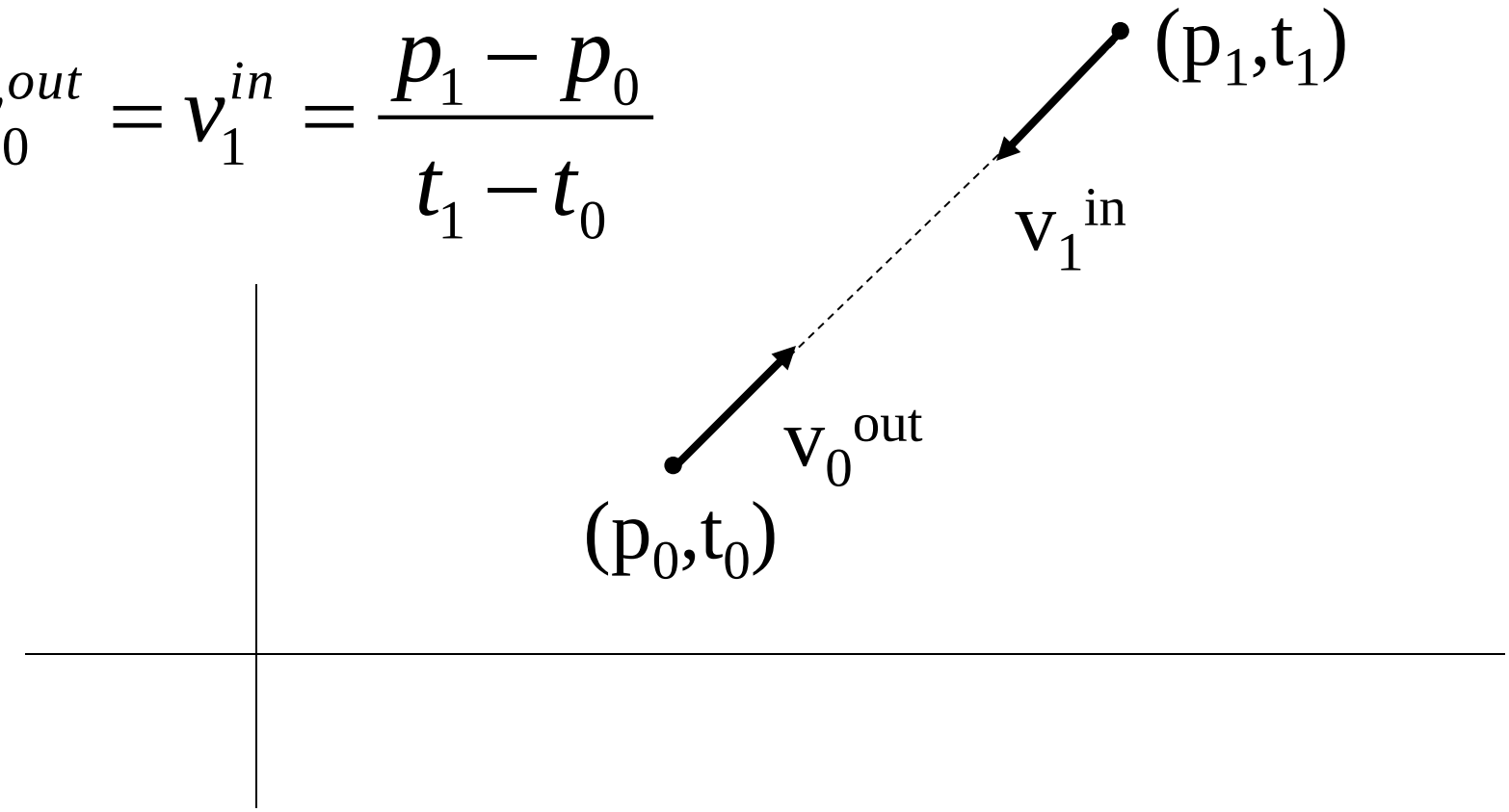
## Tangentes Flat

- Las tangentes Flat son particularmente útiles para hacer movimientos 'slow in' y 'slow out' (aceleración de una parada y desaceleración hasta parar)



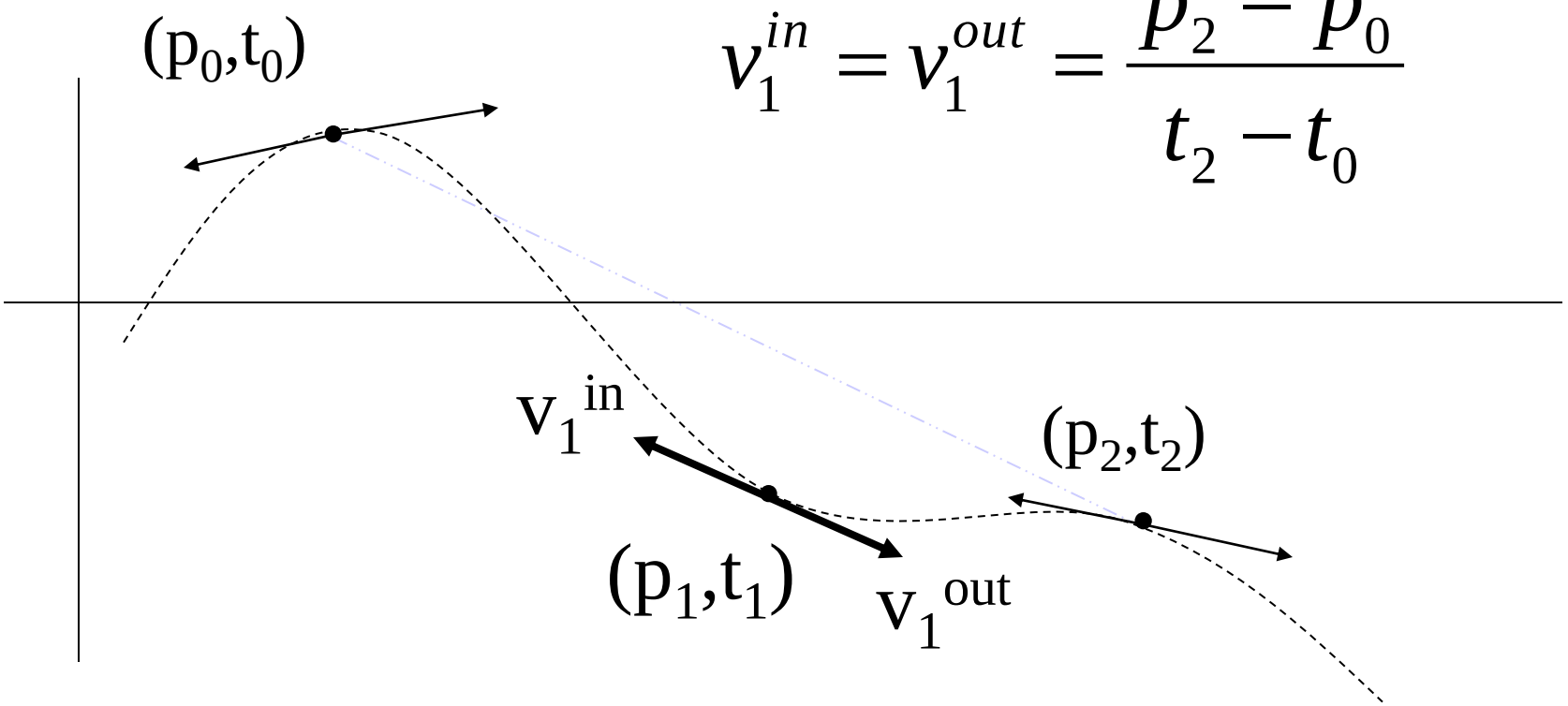
## Tangentes lineales

$$v_0^{out} = v_1^{in} = \frac{p_1 - p_0}{t_1 - t_0}$$



## Tangentes suaves (*smooth*)

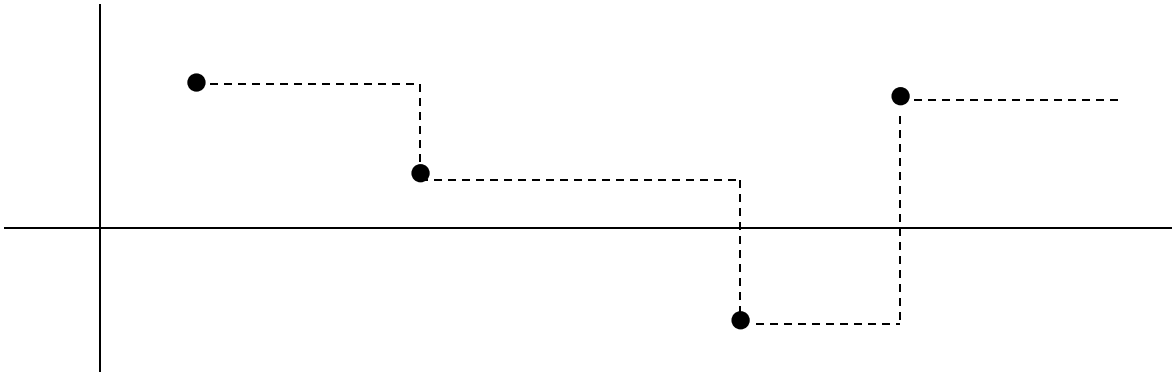
$$v_1^{in} = v_1^{out} = \frac{p_2 - p_0}{t_2 - t_0}$$




- Tenga en cuenta que esto no va a funcionar en la primera o la última tangente (sólo tiene que utilizar la regla lineal).

## Tangente 'paso' o *step*

- Este es un caso especial que sólo obliga a todo el span (arco) a una constante
- Esto requiere hackear los coeficientes cúbicos (a = b = c = 0, d = $p_0$)
- Sólo se puede especificar en la tangente de salida y anula cualquier regla es en la siguiente tangente entrante



## Interpolación cúbica de Hermite

- Los fotogramas clave se almacenan en su orden temporal.
- Entre cada dos frames clave sucesivos es un lapso de una curva cúbica.
- El *span* (sensibilidad) se define por el valor de los dos keyframes y la tangente de salida de la primera y entrante tangente de la segunda.
- Estos 4 valores se multiplican por la matriz de la base de Hermite y se convierten en coeficientes cúbicos para el *span*.
- Por simplicidad, los coeficientes se pueden almacenar en el primer keyframe para cada span.

## Interpolación cúbica de Hermite

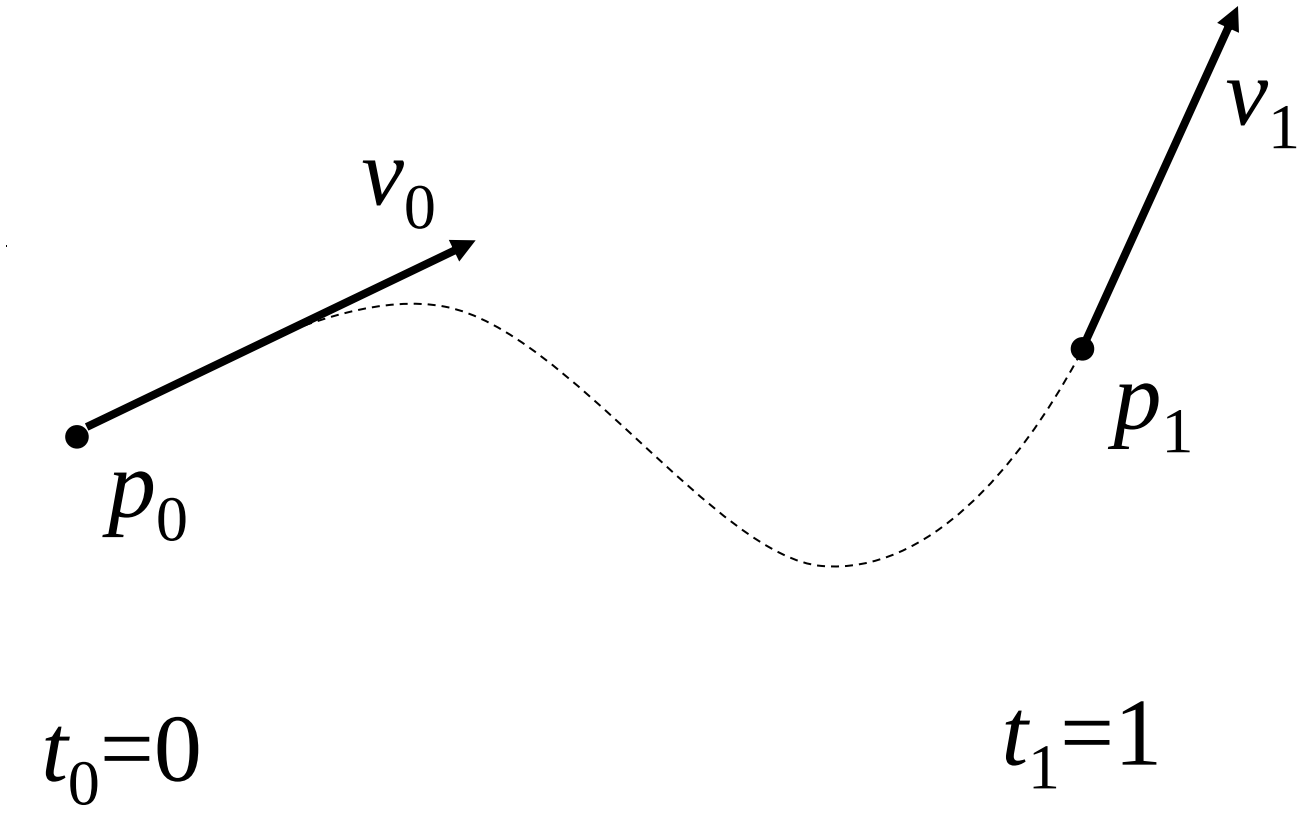
$$f(t) = at^3 + bt^2 + ct + d$$

$$\frac{df}{dt} = 3at^2 + 2bt + c$$

$$f(t) = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \cdot \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}$$

$$\frac{df}{dt} = \begin{bmatrix} 3t^2 & 2t & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}$$

## Interpolación cúbica de Hermite



- Queremos que el valor de la curva en t=0 sea f(0)=$p_0$, y en t=1, que sea f(1)=$p_1$
- Queremos que la derivada de la curva en t=0 sea $v_0$, y $v_1$ en t=1

---

$$f(0) = p_0 = a0^3 + b0^2 + c0 + d = d$$
$$f(1) = p_1 = a1^3 + b1^2 + c1 + d = a + b + c + d$$
$$f'(0) = v_0 = 3a0^2 + 2b0 + c = c$$
$$f'(1) = v_1 = 3a1^2 + 2b1 + c = 3a + 2b + c$$

$$p_0 = d$$
$$p_1 = a + b + c + d$$
$$v_0 = c$$
$$v_1 = 3a + 2b + c$$

$$\begin{bmatrix} p_0 \\ p_1 \\ v_0 \\ v_1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}$$

$$\begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix}^{-1} \cdot \begin{bmatrix} p_0 \\ p_1 \\ v_0 \\ v_1 \end{bmatrix}$$

$$\begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} p_0 \\ p_1 \\ v_0 \\ v_1 \end{bmatrix}$$

$$f(t) = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \cdot \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} p_0 \\ p_1 \\ v_0 \\ v_1 \end{bmatrix}$$

$$f(t) = \mathbf{t} \cdot \mathbf{B}_{Hrm} \cdot \mathbf{g}_{Hrm}$$
$$f(t) = \mathbf{t} \cdot \mathbf{c}$$

- Recordemos, esto asume que *t* varía entre 0 y 1

- Si $t_0$ es el tiempo de la primera clave y $t_1$ es el tiempo de la segunda, una interpolación lineal de estos tiempos por un parámetro *u* sería:

$$t = Lerp(u, t_0, t_1) = (1-u)t_0 + ut_1$$

- La inversa de esta operación nos da:

$$u = InvLerp(t, t_0, t_1) = \frac{t - t_0}{t_1 - t_0}$$

- Esto nos da un valor de 0…1 sobre el span donde se evaluará la ecuación cubica.
- Nota: $1/(t_1-t_0)$ puede ser precalculada para cada span.

- Las tangentes son generalmente expresadas como una pendiente de valor/tiempo
- Para normalizar los spans al rango 0 ... 1 , tenemos que corregir las tangentes
- Así que tenemos que escalarlos por $(t_1-t_0)$

- Para cada span se pre-calcula los coeficientes cúbicos:

$$\begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} p_0 \\ p_1 \\ (t_1 - t_0)v_0 \\ (t_1 - t_0)v_1 \end{bmatrix}$$

- Para evaluar la ecuación cubica para un span, se debe primero, convertir nuestro tiempo t a un valor de 0 .. 1 para el span (llamaremos a este parámetro *u*).

$$u = InvLerp(t, t_0, t_1) = \frac{t - t_0}{t_1 - t_0}$$

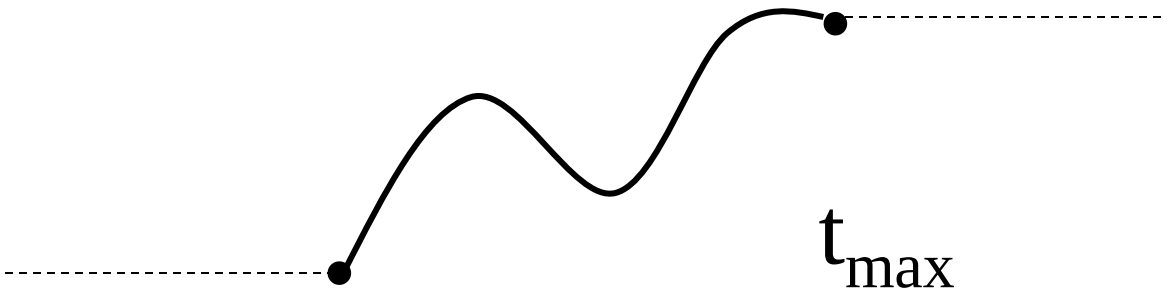
$$x = au^3 + bu^2 + cu + d = d + u(c + u(b + u(a)))$$

- Los dos principales cálculos de configuración que necesita llevar a cabo un canal de fotogramas clave son:
  - Computar tangentes desde reglas.
  - Computar coeficientes cúbicos desde tangentes y otros datos.
- Esto se puede hacer en dos pasos separados a través de las claves o combinarse en una sola pasada (pero hay que tener en cuenta que hay algunas dependencias si se hace en una sola pasada).

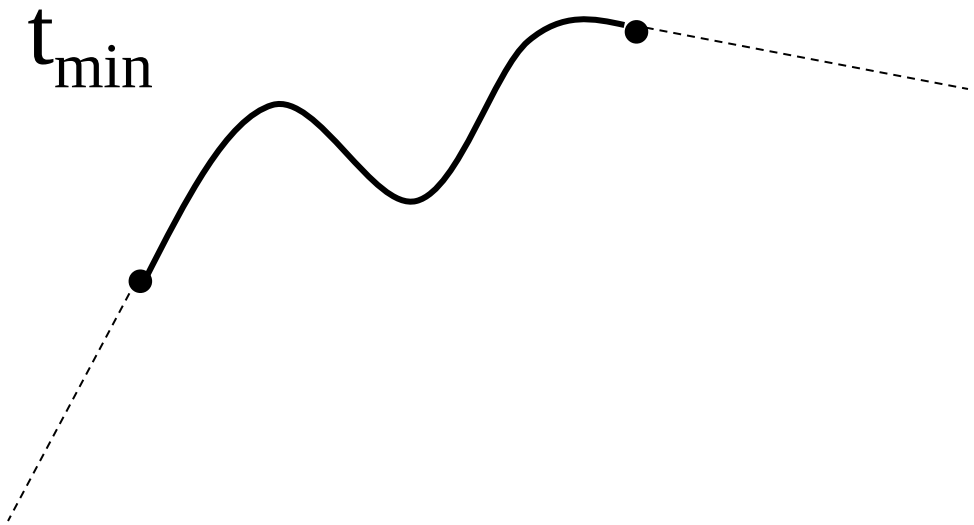
- Los canales pueden especificar modos de extrapolación para definir como las curvas es extrapolada antes $t_{min}$ y despues $t_{max}$
- Se pueden establecer modos de extrapolación diferentes para antes y después de los datos reales.
- Elecciones comunes:
  - Valor constante (primer / último valor clave)
  - Lineal (uso tangente en primera / última clave)
  - Cíclica (repetición de todo el canal)
  - Cíclica Offset (repetir con el valor del offset)
  - Rebote (repetición alterna hacia atrás y hacia delante)

- Hay que tener en cuenta que la extrapolación se aplica a todo el canal y no a las claves individuales.
- De hecho, la extrapolación no está directamente ligado a los keyframes y se puede utilizar para cualquier método de almacenamiento de canal (en bruto ...).

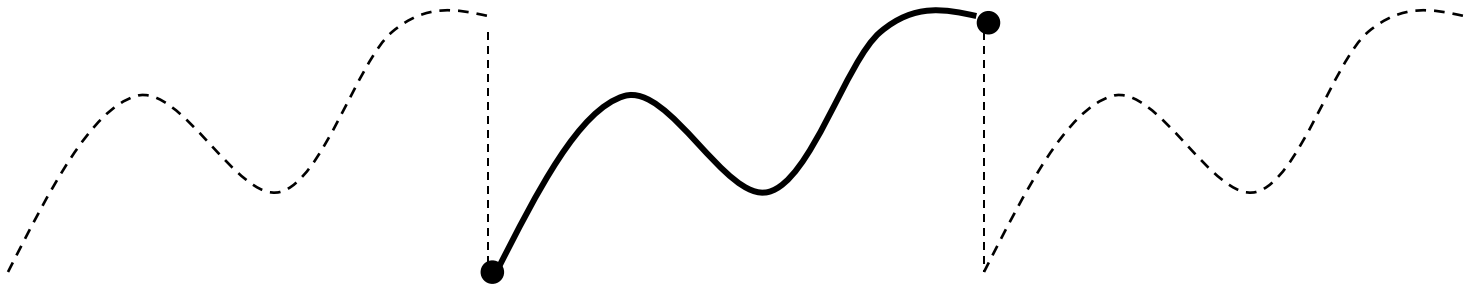
- Flat:



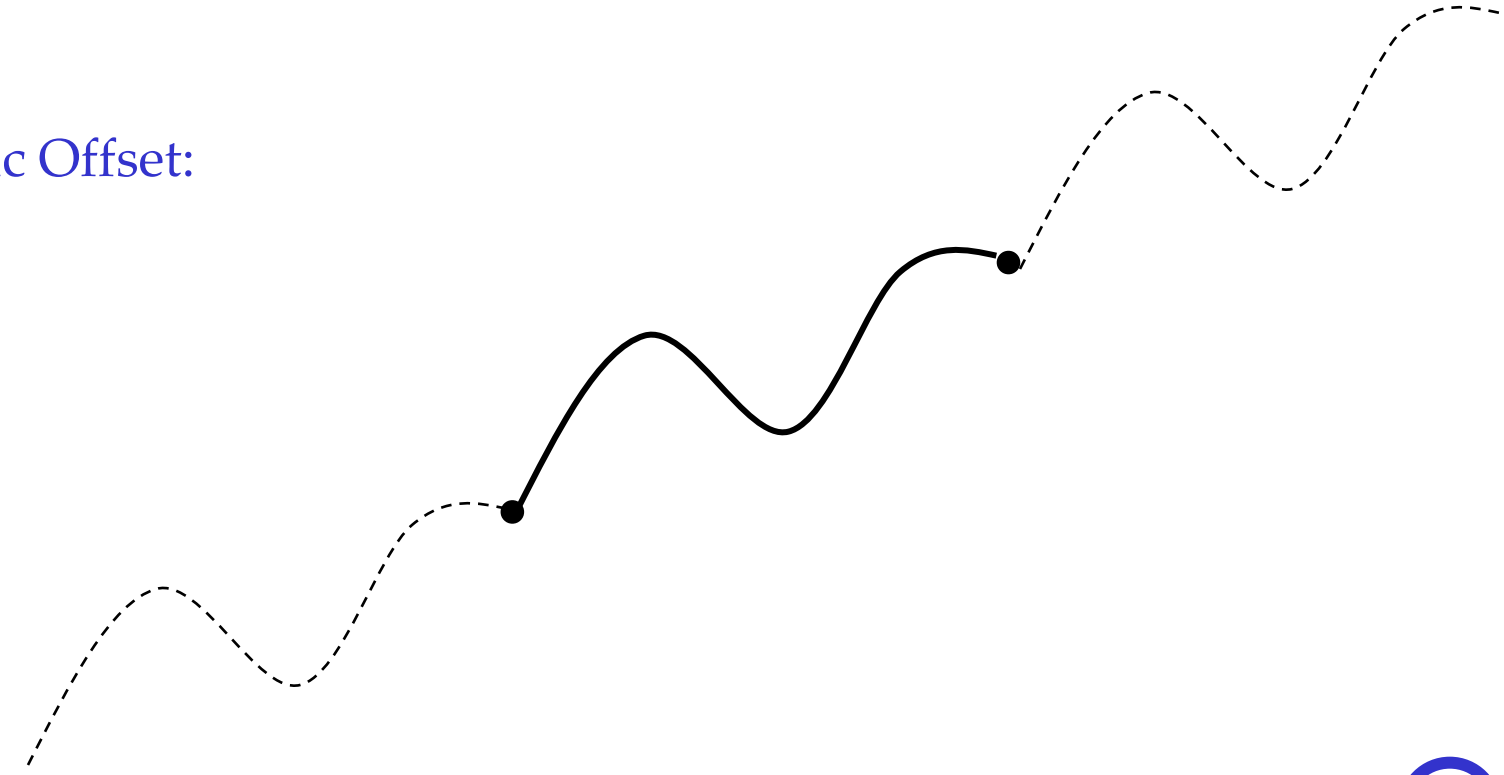

- Linear:



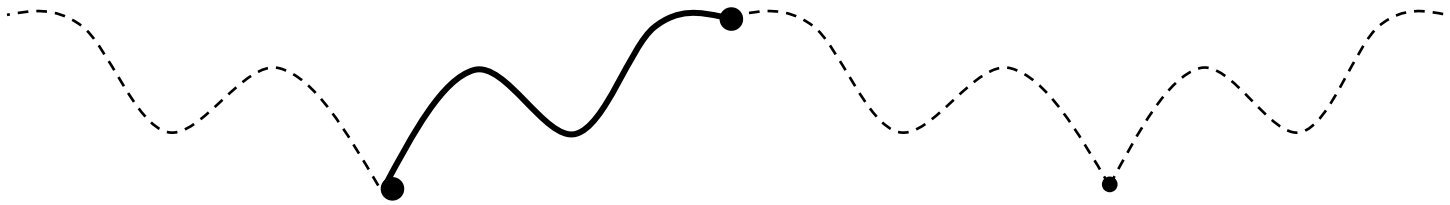
- Cyclic:



- Cyclic Offset:



- Bounce:



- La principal funcion para un canal es algo así como:

  float Channel::Evaluate(float time);

- Esta función será llamada muchas veces…
- Para un tiempo de entrada *t*, hay 4 casos que podemos considerar:
  - *t* está antes de la primera clave (usar extrapolation)
  - *t* está despues de la última clave (usar extrapolation)
  - *t* cae exactamente sobre alguna clave (devolver el valor de la clave)
  - *t* cae entre dos claves (evaluar ecuación cubica)

- La funcion Channel::Evaluate() necesita ser muy eficiente, como es llamada muchas veces durante la reproducción de animaciones
- Hay dos componentes principales para la evaluación:
  - Encontrar el propio span
  - Evaluar la ecuación cúbica para el span

- Para evaluar un canal en algún tiempo arbitrario *t*, tenemos que encontrar primero el *span* apropiado del canal y luego evaluar su ecuación.
- Si los *keyframes* están espaciados irregularmente hay que hacer una búsqueda del span correspondiente.
- Si los keyframes se almacenan como una lista enlazada, hay poco que podamos hacer excepto caminar a través de la lista en busca del span adecuado.
- Si se almacenan en un array, podemos utilizar una búsqueda binaria, lo que debería hacerse razonablemente bien.

## Búsqueda binaria

- Permite encontrar el span adecuado bastante rápido (log N) sin costo adicional de almacenamiento (las claves se asumen que se almacenan en una matriz en lugar de una lista).
- La búsqueda binaria se llama a veces "divide y vencerás" o ‘bisección’.
- Para un acceso aún más rápido se podrían utilizar algoritmos de hash, pero normalmente no es necesario ya que requieren de almacenamiento adicional.

## Búsqueda lineal

- Uno siempre puede simplemente recorrer las claves desde el principio y buscar el span adecuado.
- Este es una forma aceptable para empezar, ya que es importante para que las cosas funcionen correctamente antes de centrarse en la optimización.
- También puede ser una opción razonable para herramientas de edición interactivas que requerirían keyframes que se almacenen en una lista enlazada.
- Por supuesto, un algoritmo de bisección probablemente puede ser escrito en menos de una docena de líneas de código ...

## Acceso secuencial

- Si un personaje está reproduciendo una animación, a continuación, se accede a la secuencia de datos de canal.
- Hacer una búsqueda binaria para cada evaluación de canal para cada frame no es eficiente.
- Si hacemos un seguimiento de la clave que se ha accedido más recientemente para cada canal, entonces es muy probable que el siguiente acceso requerido ya sea la misma clave o la siguiente. Esto hace el acceso secuencial de frames clave potencialmente muy rápido.
- Sin embargo hay una trampa ...

## Acceso secuencial

- Considere un caso en el que tenemos un juego de video con 20 chicos malos corriendo. Todos ellos necesitan para acceder a los mismos datos de la animación (que sólo se deben almacenar una vez obviamente). Sin embargo, podrían estar cada uno con el acceso a los canales con un "tiempo" diferente.
- Por lo tanto, el código de nivel superior que reproduce animaciones necesita realizar un seguimiento de las claves más recientes en lugar de la solución más simple de sólo tener cada canal sólo almacenado un puntero a su clave más reciente.
- Por lo tanto, la clase de jugador de la animación tiene que hacer una considerable contabilidad, ya que tendrá que hacer un seguimiento de una clave más reciente para todos los canales en la animación.

- Si los coeficientes de interpolación están almacenados, se puede evaluar la ecuación cubica con 4 sumas y 4 multiplicaciones
- En otras palabras, la evaluación de la cúbica es prácticamente instantánea, mientras que saltar a través de la memoria tratando de localizar el span es mucho peor.
- Si somos capaces de aprovechar el acceso secuencial (que por lo general se puede) podemos reducir el cálculo de la ubicación del span a un número de operaciones muy pequeño.

- El canal siempre debe devolver un valor razonable, independientemente de lo que el tiempo t se aprobó en
  - Si no hay claves en el canal se devolvería 0
  - Si hay solo 1 clave, este debería retornar el valor de la clave
  - Si hay mas de una clave, debería evaluar la curva o usar una regla de extrapolación si está fuera de rango.
  - Como mínimo, la regla de extrapolación 'constante' debería ser usada, la cual devuelve el valor de la primera (o última) clave si está antes (o después) del rango de keyframe.
- Cuando creamos nuevas claves o modificamos el tiempo de una clave, uno necesita verificar que su tiempo permanece entre la clave anterior y después.