



ANIMACIÓN POR ORDENADOR

Tema 4

La animación como cambio.
Fotogramas clave. Intercalado.
Funciones de movimiento. Orientación.



CONTENIDO

1. La animación como cambio.
2. Fotogramas clave.
3. Un repaso al álgebra lineal.
4. Intercalado. Interpolaciones.
5. Interpretación del movimiento: las CURVAS DE ANIMACIÓN.
6. Orientación. Cuaternios (*Quaternions*)



INTERCALADO. INTERPOLACIONES





FUNCIONES POLINÓMICAS

Lineal:

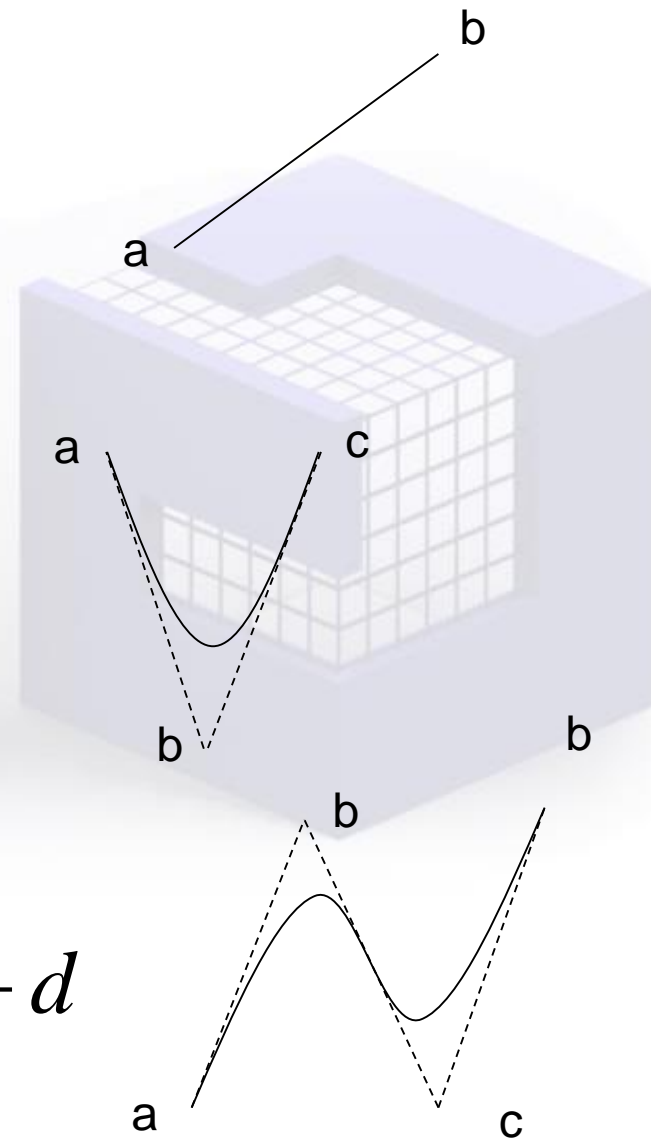
$$f(t) = at + b$$

Cuadrática:

$$f(t) = at^2 + bt + c$$

Cúbica:

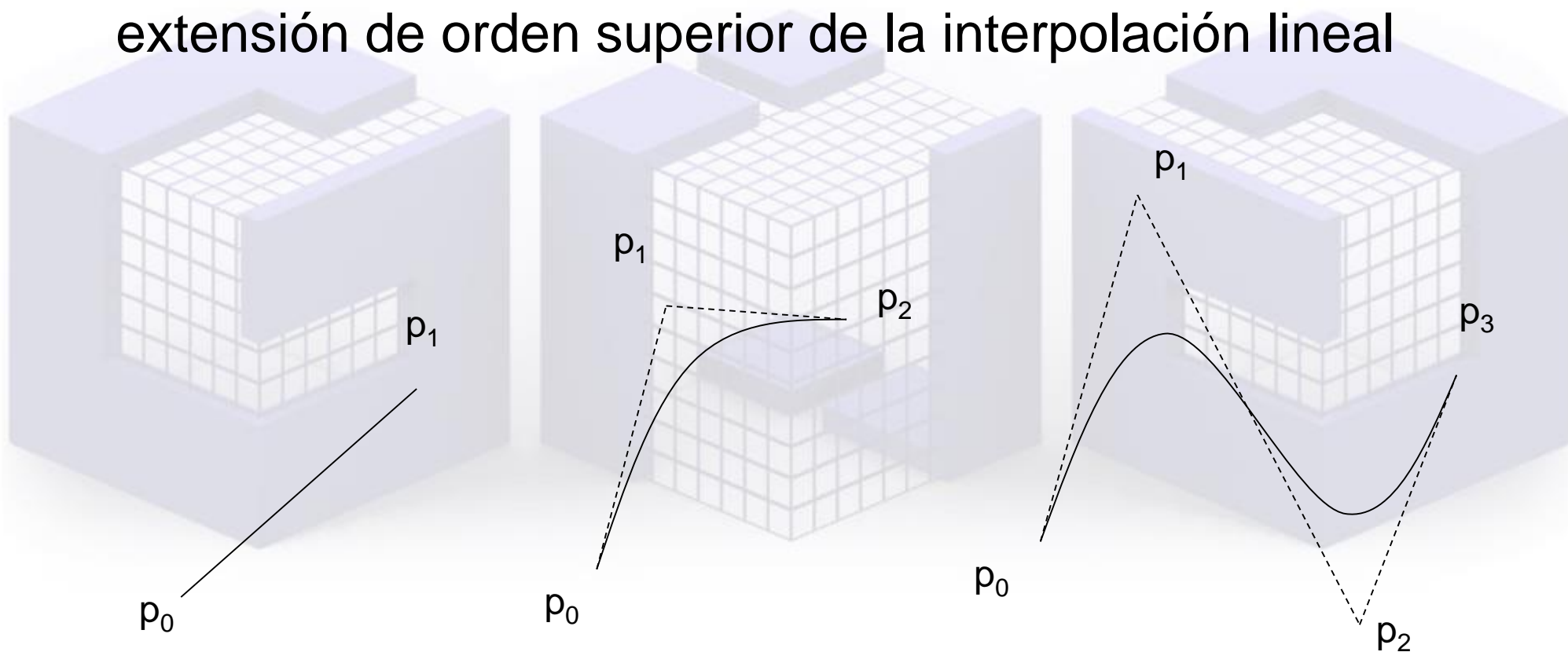
$$f(t) = at^3 + bt^2 + ct + d$$





CURVAS DE BÉZIER

Las curvas de Bézier pueden considerarse como una extensión de orden superior de la interpolación lineal





FORMULACIÓN DE LAS CURVAS DE BÉZIER

Hay varias maneras de formular las curvas de Bézier matemáticamente.

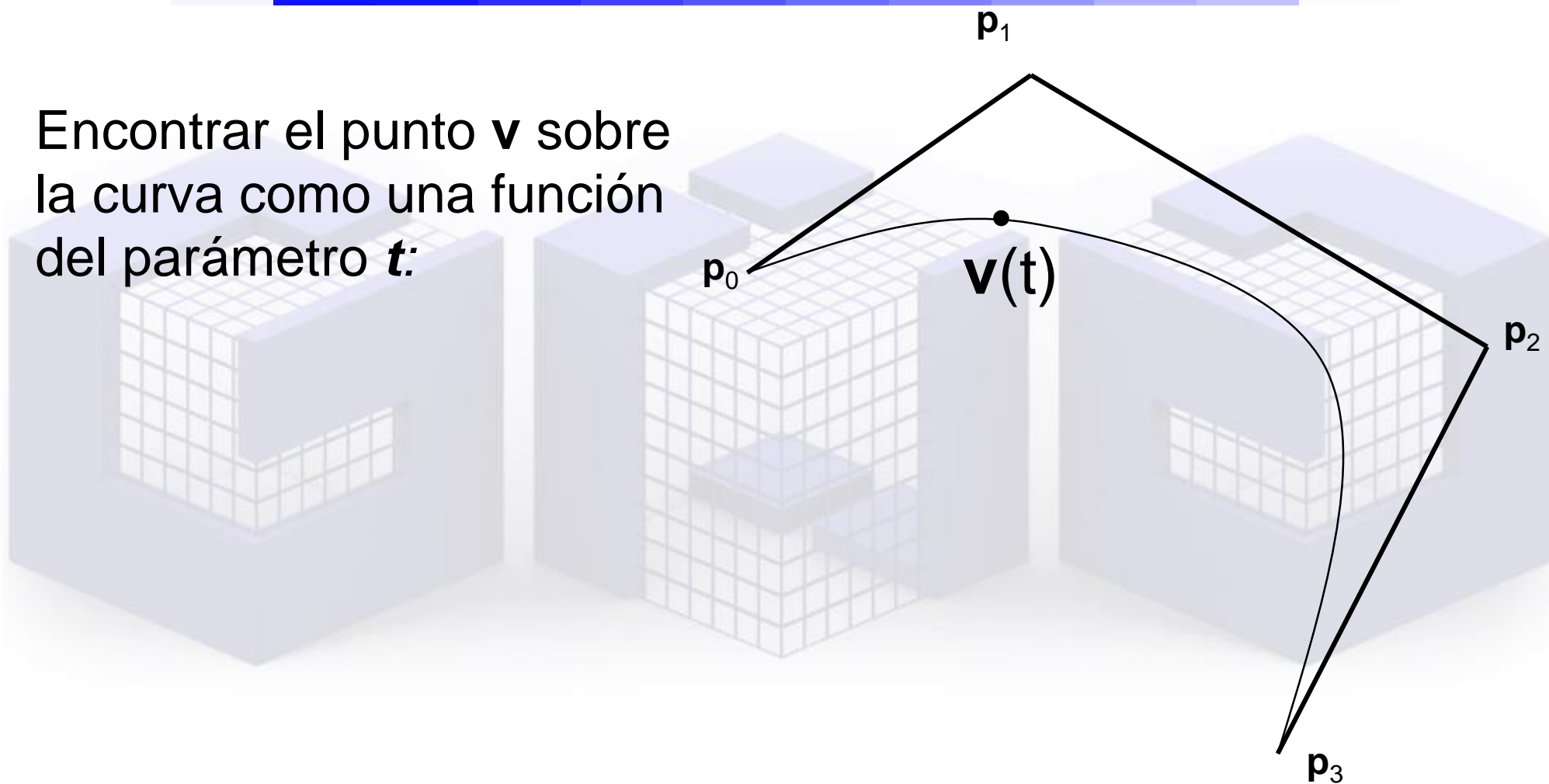
- Algoritmo de *de Castlejau* (interpolaciones lineales recursivas)
- Polinomios de *Bernstein* (funciones que definen la influencia de cada punto de control como una función de t)
- Ecuaciones cúbicas (ecuación cúbica general)
- Forma matricial

En la práctica, la forma matricial es la más útil en animación por ordenador, pero los demás son importantes para entender cómo funcionan.



CURVA DE BÉZIER

Encontrar el punto \mathbf{v} sobre la curva como una función del parámetro t :

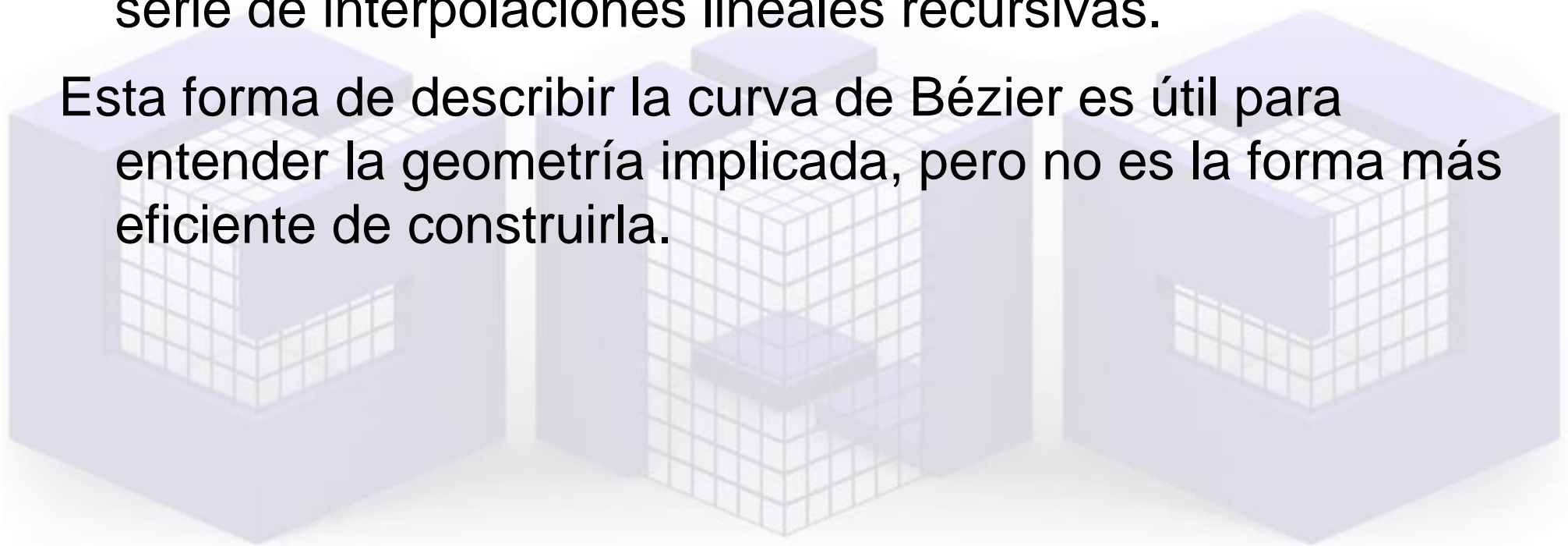




ALGORITMO DE *DE CASTELJAU*

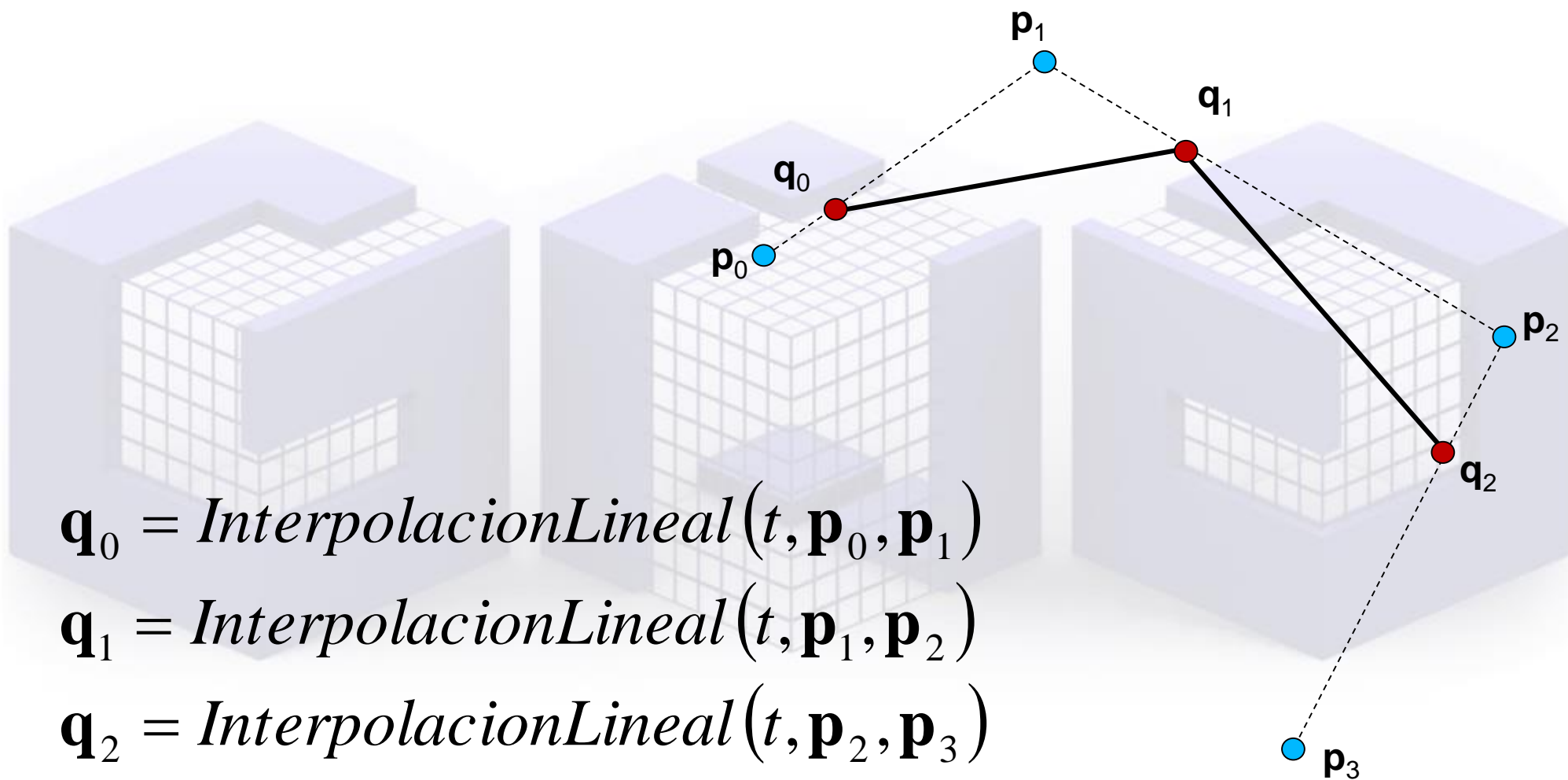
El algoritmo de *de Casteljau* describe la curva como una serie de interpolaciones lineales recursivas.

Esta forma de describir la curva de Bézier es útil para entender la geometría implicada, pero no es la forma más eficiente de construirla.



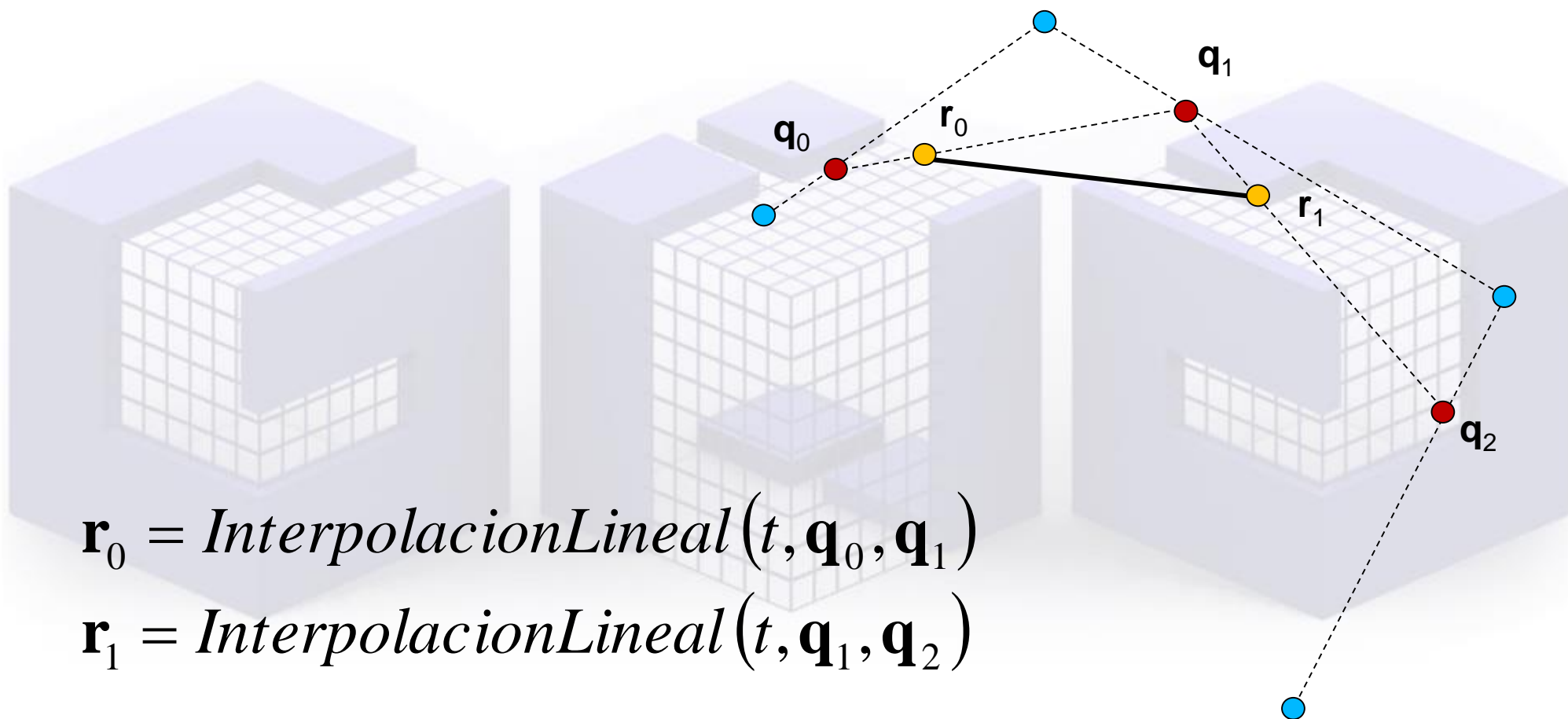


ALGORITMO DE DE CASTELJAU



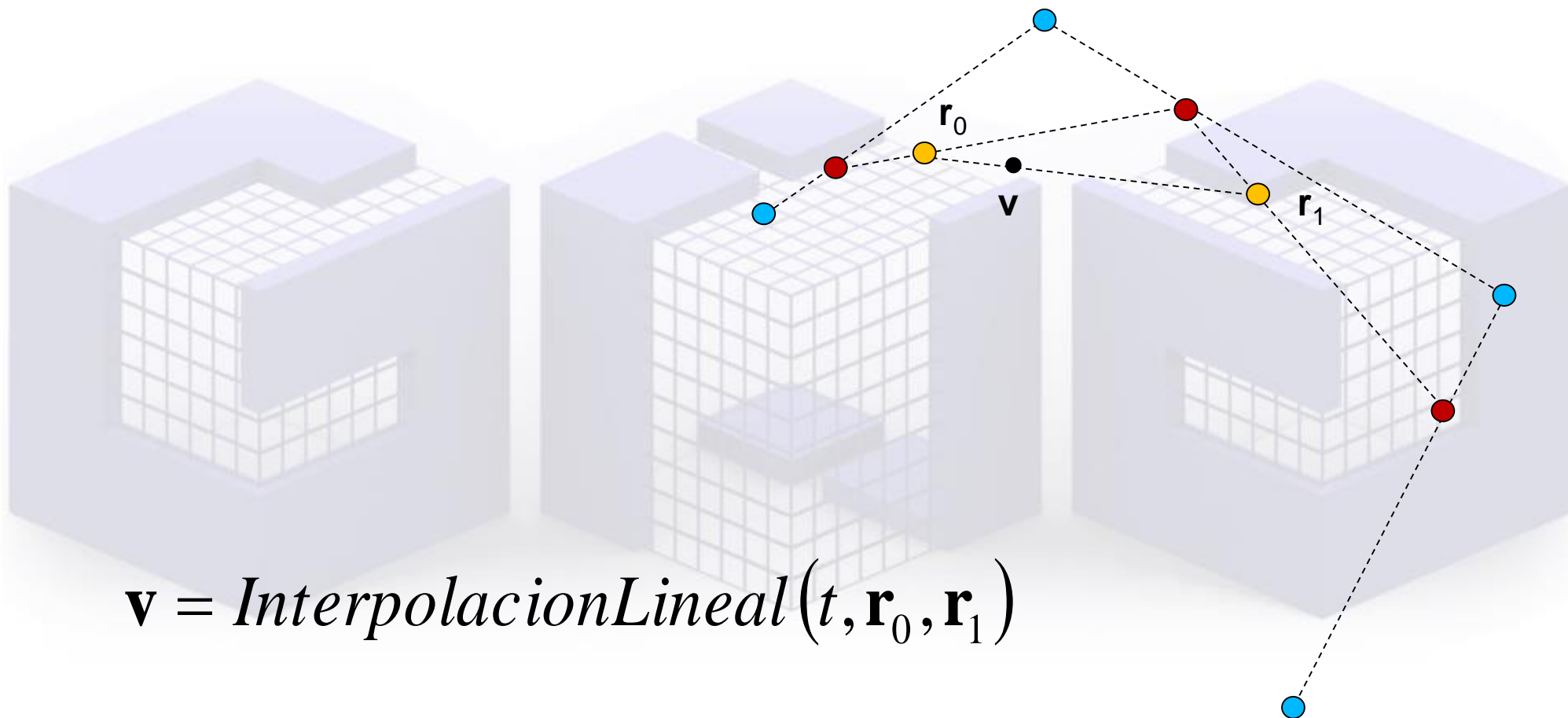


ALGORITMO DE DE CASTELJAU





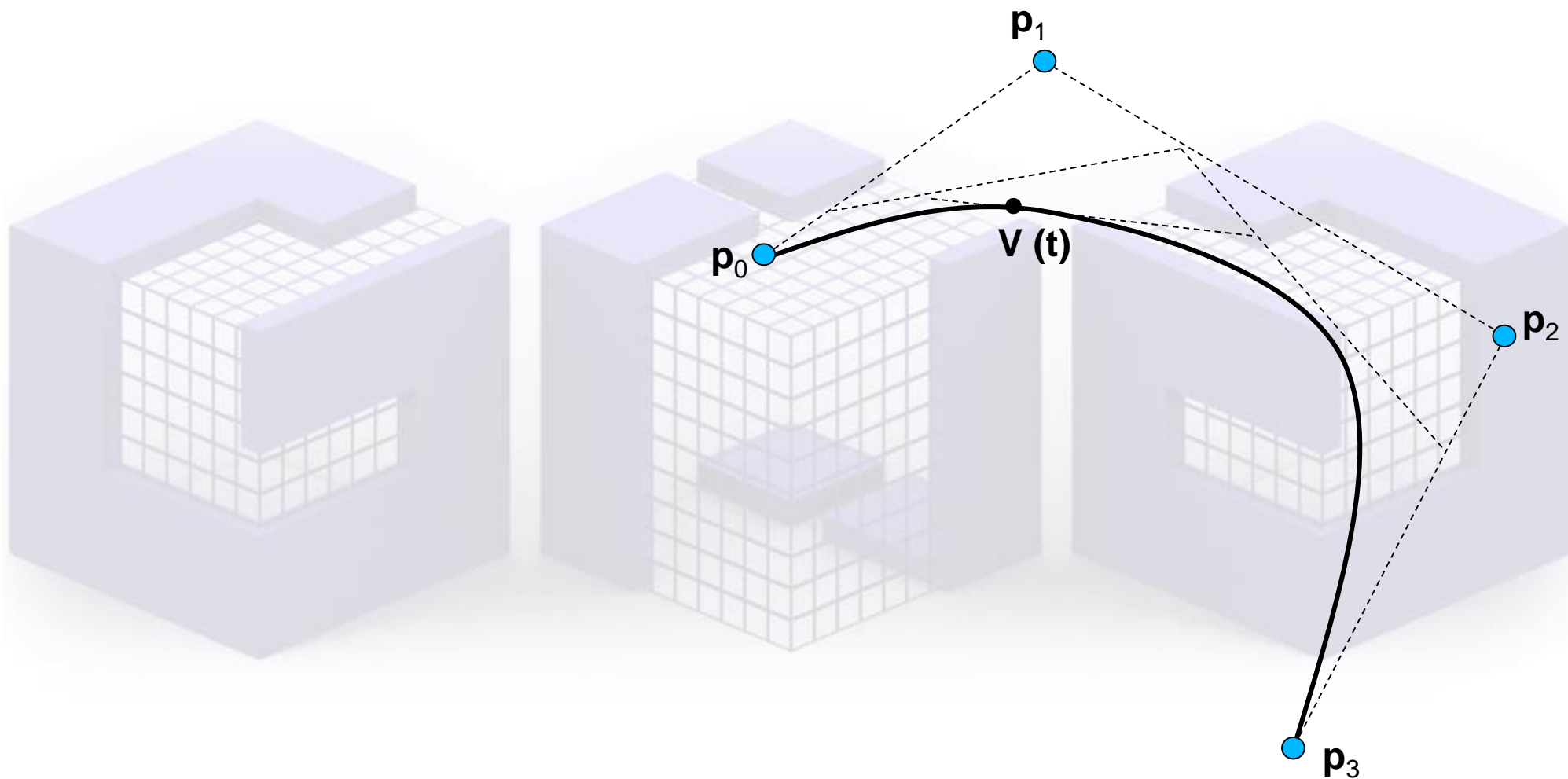
ALGORITMO DE DE CASTELJAU



$$\mathbf{v} = \text{InterpolacionLineal}(t, \mathbf{r}_0, \mathbf{r}_1)$$

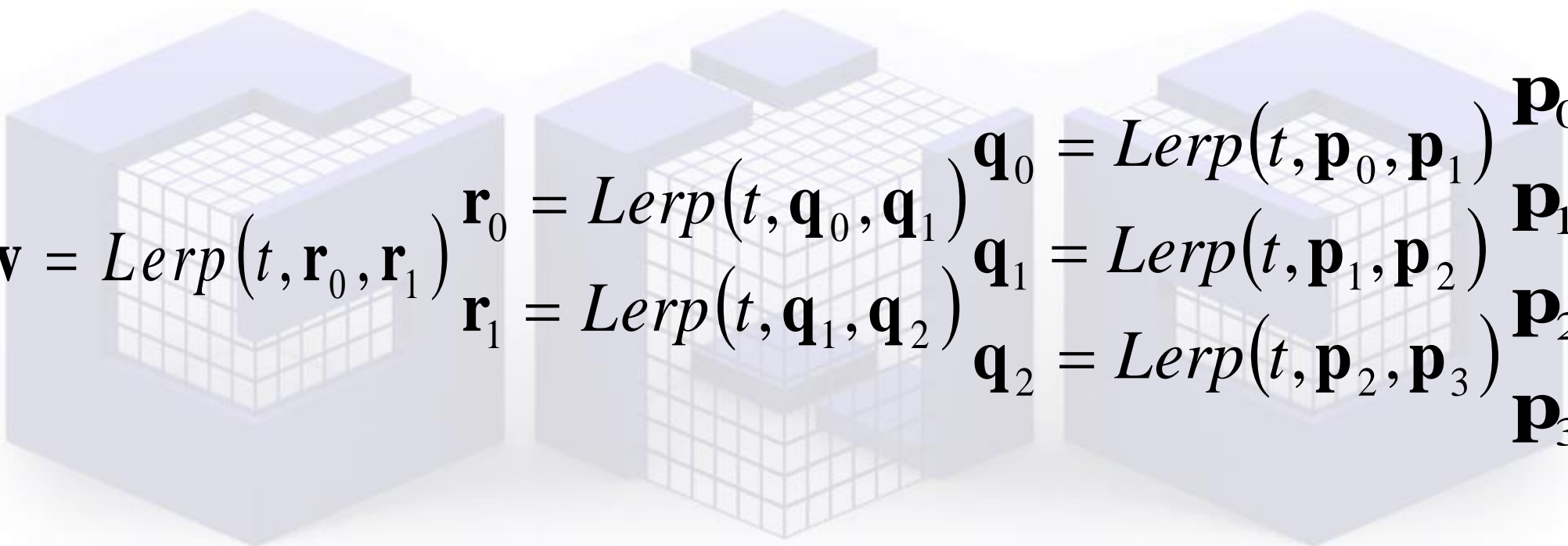


CURVA DE BÉZIER





INTERPOLACIÓN LINEAL RECURSIVA


$$\mathbf{v} = \text{Lerp}(t, \mathbf{r}_0, \mathbf{r}_1)$$
$$\mathbf{r}_0 = \text{Lerp}(t, \mathbf{q}_0, \mathbf{q}_1)$$
$$\mathbf{r}_1 = \text{Lerp}(t, \mathbf{q}_1, \mathbf{q}_2)$$
$$\mathbf{q}_0 = \text{Lerp}(t, \mathbf{p}_0, \mathbf{p}_1)$$
$$\mathbf{q}_1 = \text{Lerp}(t, \mathbf{p}_1, \mathbf{p}_2)$$
$$\mathbf{q}_2 = \text{Lerp}(t, \mathbf{p}_2, \mathbf{p}_3)$$
$$\mathbf{p}_0$$
$$\mathbf{p}_1$$
$$\mathbf{p}_2$$
$$\mathbf{p}_3$$

$$\text{Lerp}(t, \mathbf{a}, \mathbf{b}) = (1 - t)\mathbf{a} + t\mathbf{b}$$



POLINOMIOS DE BERNSTEIN

- Partiendo del algoritmo de *de Casteljau*, expandimos las expresiones y las agrupamos en funciones polinómicas sobre t , multiplicando el resultado por los puntos en el polígono de control
- La generalización de esta expresión nos da la forma de ***Bernstein*** de la curva Bézier
- Esto nos da una mayor comprensión de lo que ocurre en la curva:
 - Podemos ver la influencia de cada punto del polígono de control como una función de t
 - Vemos que las funciones base suman 1 para cualquier valor de t , lo que indica que la curva de Bezier es una media convexa de los puntos de control



EXPANDIENDO LAS INTERPOLACIONES LINEALES

$$\mathbf{q}_0 = \text{Lerp}(t, \mathbf{p}_0, \mathbf{p}_1) = (1-t)\mathbf{p}_0 + t\mathbf{p}_1$$

$$\mathbf{q}_1 = \text{Lerp}(t, \mathbf{p}_1, \mathbf{p}_2) = (1-t)\mathbf{p}_1 + t\mathbf{p}_2$$

$$\mathbf{q}_2 = \text{Lerp}(t, \mathbf{p}_2, \mathbf{p}_3) = (1-t)\mathbf{p}_2 + t\mathbf{p}_3$$

$$\mathbf{r}_0 = \text{Lerp}(t, \mathbf{q}_0, \mathbf{q}_1) = (1-t)((1-t)\mathbf{p}_0 + t\mathbf{p}_1) + t((1-t)\mathbf{p}_1 + t\mathbf{p}_2)$$

$$\mathbf{r}_1 = \text{Lerp}(t, \mathbf{q}_1, \mathbf{q}_2) = (1-t)((1-t)\mathbf{p}_1 + t\mathbf{p}_2) + t((1-t)\mathbf{p}_2 + t\mathbf{p}_3)$$

$$\begin{aligned} \mathbf{v} = \text{Lerp}(t, \mathbf{r}_0, \mathbf{r}_1) = & (1-t)((1-t)((1-t)\mathbf{p}_0 + t\mathbf{p}_1) + t((1-t)\mathbf{p}_1 + t\mathbf{p}_2)) \\ & + t((1-t)((1-t)\mathbf{p}_1 + t\mathbf{p}_2) + t((1-t)\mathbf{p}_2 + t\mathbf{p}_3)) \end{aligned}$$



POLINOMIOS DE BERNSTEIN

$$\mathbf{v} = (1-t)((1-t)((1-t)\mathbf{p}_0 + t\mathbf{p}_1) + t((1-t)\mathbf{p}_1 + t\mathbf{p}_2)) \\ + t((1-t)((1-t)\mathbf{p}_1 + t\mathbf{p}_2) + t((1-t)\mathbf{p}_2 + t\mathbf{p}_3))$$

$$\mathbf{v} = (1-t)^3 \mathbf{p}_0 + 3(1-t)^2 t \mathbf{p}_1 + 3(1-t) t^2 \mathbf{p}_2 + t^3 \mathbf{p}_3$$

$$\mathbf{v} = (-t^3 + 3t^2 - 3t + 1)\mathbf{p}_0 + (3t^3 - 6t^2 + 3t)\mathbf{p}_1 \\ + (-3t^3 + 3t^2)\mathbf{p}_2 + (t^3)\mathbf{p}_3$$



POLINOMIOS DE BERNSTEIN CÚBICOS

$$\mathbf{v} = (-t^3 + 3t^2 - 3t + 1)\mathbf{p}_0 + (3t^3 - 6t^2 + 3t)\mathbf{p}_1 \\ + (-3t^3 + 3t^2)\mathbf{p}_2 + (t^3)\mathbf{p}_3$$

$$\mathbf{v} = B_0^3(t)\mathbf{p}_0 + B_1^3(t)\mathbf{p}_1 + B_2^3(t)\mathbf{p}_2 + B_3^3(t)\mathbf{p}_3$$

$$B_0^3(t) = -t^3 + 3t^2 - 3t + 1$$

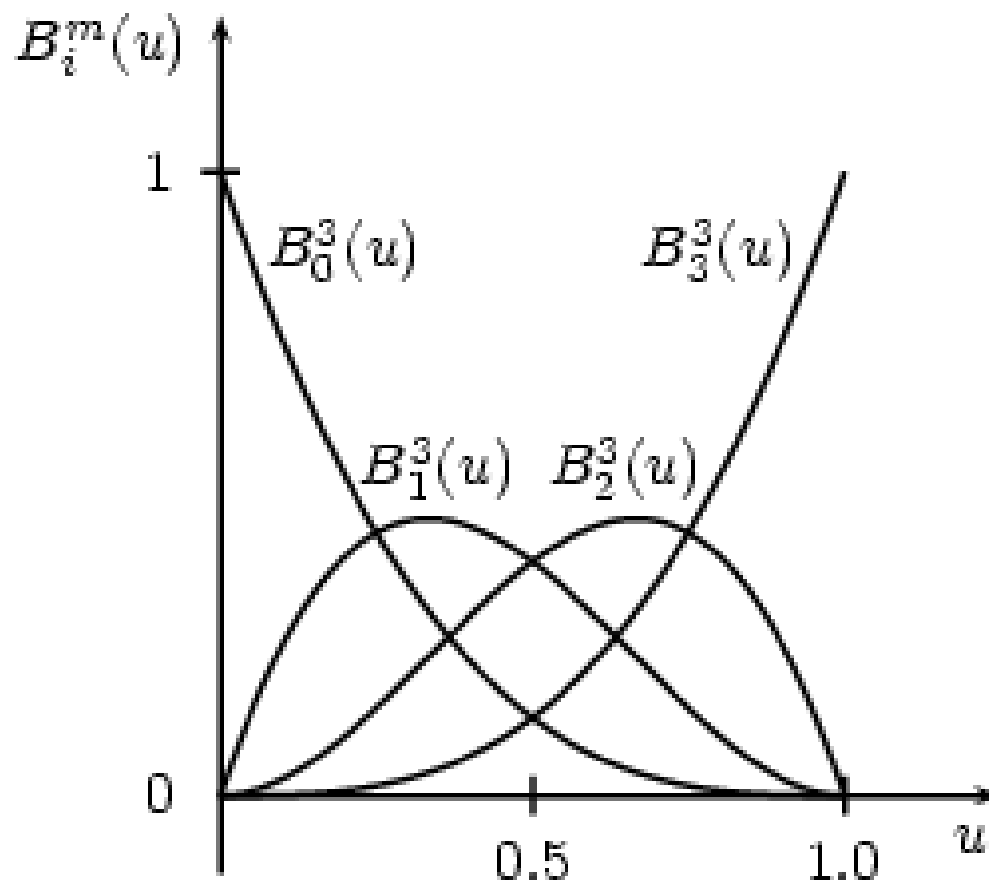
$$B_1^3(t) = 3t^3 - 6t^2 + 3t$$

$$B_2^3(t) = -3t^3 + 3t^2$$

$$B_3^3(t) = t^3$$



POLINOMIOS DE BERNSTEIN





POLINOMIOS DE BERNSTEIN

$$B_0^3(t) = -t^3 + 3t^2 - 3t + 1$$

$$B_1^3(t) = 3t^3 - 6t^2 + 3t$$

$$B_2^3(t) = -3t^3 + 3t^2$$

$$B_3^3(t) = t^3$$

$$B_0^2(t) = t^2 - 2t + 1$$

$$B_1^2(t) = -2t^2 + 2t$$

$$B_2^2(t) = t^2$$

$$B_0^1(t) = -t + 1$$

$$B_1^1(t) = t$$

$$B_i^n(t) = \binom{n}{i} (1-t)^{n-i} (t)^i$$

$$\binom{n}{i} = \frac{n!}{i!(n-i)!}$$

$$\sum B_i^n(t) = 1$$



POLINOMIOS DE BERNSTEIN

Curva de Bézier mediante polinomios de Bernstein:

$$B_i^n(t) = \binom{n}{i} (1-t)^{n-i} (t)^i$$

$$\mathbf{v}(t) = \sum_{i=0}^n B_i^n(t) \mathbf{p}_i$$



BÉZIER COMO ECUACIÓN CÚBICA

- Si reagrupamos la ecuación por términos de exponentes de t , obtenemos la forma cúbica estándar de Bézier
- Esta forma es ideal para una evaluación rápida, ya que todos los términos constantes (**a , b , c , d**) se pueden precalcular
- La forma de ecuación cúbica *oculta* la geometría de entrada en la expresión, pero dicha geometría siempre se puede extraer fuera de los coeficientes cúbicos



BÉZIER COMO ECUACIÓN CÚBICA

$$\mathbf{v} = (-t^3 + 3t^2 - 3t + 1)\mathbf{p}_0 + (3t^3 - 6t^2 + 3t)\mathbf{p}_1 \\ + (-3t^3 + 3t^2)\mathbf{p}_2 + (t^3)\mathbf{p}_3$$

$$\mathbf{v} = (-\mathbf{p}_0 + 3\mathbf{p}_1 - 3\mathbf{p}_2 + \mathbf{p}_3)t^3 + (3\mathbf{p}_0 - 6\mathbf{p}_1 + 3\mathbf{p}_2)t^2 \\ + (-3\mathbf{p}_0 + 3\mathbf{p}_1)t + (\mathbf{p}_0)1$$



BÉZIER COMO ECUACIÓN CÚBICA

$$\mathbf{v} = (-\mathbf{p}_0 + 3\mathbf{p}_1 - 3\mathbf{p}_2 + \mathbf{p}_3)t^3 + (3\mathbf{p}_0 - 6\mathbf{p}_1 + 3\mathbf{p}_2)t^2 + (-3\mathbf{p}_0 + 3\mathbf{p}_1)t + (\mathbf{p}_0)1$$

$$\mathbf{v} = \mathbf{a}t^3 + \mathbf{b}t^2 + \mathbf{c}t + \mathbf{d}$$

$$\mathbf{a} = (-\mathbf{p}_0 + 3\mathbf{p}_1 - 3\mathbf{p}_2 + \mathbf{p}_3)$$

$$\mathbf{b} = (3\mathbf{p}_0 - 6\mathbf{p}_1 + 3\mathbf{p}_2)$$

$$\mathbf{c} = (-3\mathbf{p}_0 + 3\mathbf{p}_1)$$

$$\mathbf{d} = (\mathbf{p}_0)$$



FORMA MATRICIAL

- Podemos reescribir las ecuaciones en forma matricial
- Esto nos da una notación compacta y muestra cómo las diferentes formas de curvas cúbicas pueden ser relacionadas
- También es una forma muy eficiente, ya que puede aprovechar las características de hardware para el uso de matrices 4×4 ...



FORMA MATRICIAL CÚBICA

$$\mathbf{v} = \mathbf{a}t^3 + \mathbf{b}t^2 + \mathbf{c}t + \mathbf{d}$$

$$\mathbf{v} = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \cdot \begin{bmatrix} \mathbf{a} \\ \mathbf{b} \\ \mathbf{c} \\ \mathbf{d} \end{bmatrix}$$

$$\mathbf{a} = (-\mathbf{p}_0 + 3\mathbf{p}_1 - 3\mathbf{p}_2 + \mathbf{p}_3)$$

$$\mathbf{b} = (3\mathbf{p}_0 - 6\mathbf{p}_1 + 3\mathbf{p}_2)$$

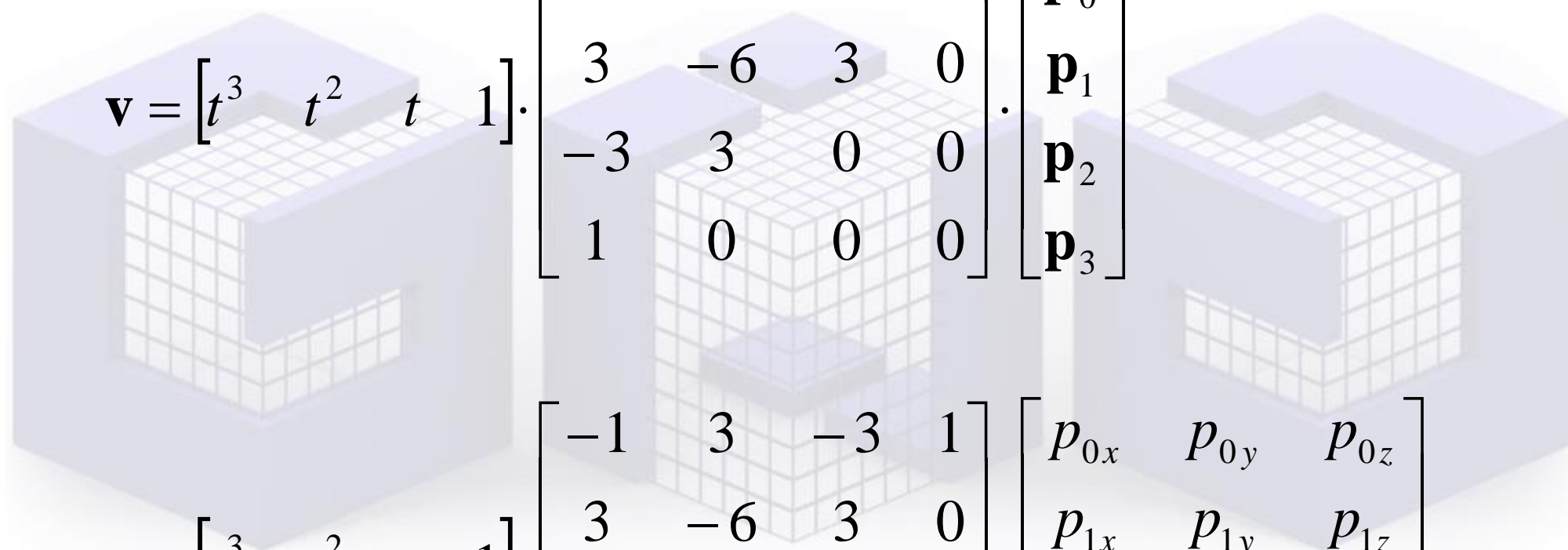
$$\mathbf{c} = (-3\mathbf{p}_0 + 3\mathbf{p}_1)$$

$$\mathbf{d} = (\mathbf{p}_0)$$

$$\begin{bmatrix} \mathbf{a} \\ \mathbf{b} \\ \mathbf{c} \\ \mathbf{d} \end{bmatrix} = \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} \mathbf{p}_0 \\ \mathbf{p}_1 \\ \mathbf{p}_2 \\ \mathbf{p}_3 \end{bmatrix}$$



FORMA MATRICIAL CÚBICA


$$\mathbf{v} = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \cdot \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} \mathbf{p}_0 \\ \mathbf{p}_1 \\ \mathbf{p}_2 \\ \mathbf{p}_3 \end{bmatrix}$$
$$\mathbf{v} = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \cdot \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} p_{0x} & p_{0y} & p_{0z} \\ p_{1x} & p_{1y} & p_{1z} \\ p_{2x} & p_{2y} & p_{2z} \\ p_{3x} & p_{3y} & p_{3z} \end{bmatrix}$$



FORMA MATRICIAL

$$\mathbf{v} = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \cdot \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} p_{0x} & p_{0y} & p_{0z} \\ p_{1x} & p_{1y} & p_{1z} \\ p_{2x} & p_{2y} & p_{2z} \\ p_{3x} & p_{3y} & p_{3z} \end{bmatrix}$$

$$\mathbf{v} = \mathbf{t} \cdot \mathbf{B}_{Bez} \cdot \mathbf{G}_{Bez}$$

$$\mathbf{v} = \mathbf{t} \cdot \mathbf{C}$$



DERIVADAS

Podemos calcular la derivada (tangente/pendiente) de una curva de forma sencilla:

$$\mathbf{v} = \mathbf{a}t^3 + \mathbf{b}t^2 + \mathbf{c}t + \mathbf{d}$$

$$\frac{d\mathbf{v}}{dt} = 3\mathbf{a}t^2 + 2\mathbf{b}t + \mathbf{c}$$

$$\mathbf{v} = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \cdot \begin{bmatrix} \mathbf{a} \\ \mathbf{b} \\ \mathbf{c} \\ \mathbf{d} \end{bmatrix}$$

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



FORMA DE HERMITE

- Veamos una forma alternativa de escribir una curva cúbica
- En vez de definirla en base a 4 puntos de control como en una curva de Bézier, vamos a definirla mediante una posición y una tangente (*velocidad*), tanto en el inicio como en el final de la curva (\mathbf{p}_0 , \mathbf{p}_1 , \mathbf{v}_0 , \mathbf{v}_1)





CURVAS HERMITE

Queremos que el valor de la curva en $t=0$ sea p_0 y p_1 en $t=1$

Queremos que la derivada de la curva en $t=0$ sea v_0 y v_1 en $t=1$

$$p_0 = a0^3 + b0^2 + c0 + d = d$$

$$p_1 = a1^3 + b1^2 + c1 + d = a + b + c + d$$

$$v_0 = 3a0^2 + 2b0 + c = c$$

$$v_1 = 3a1^2 + 2b1 + c = 3a + 2b + c$$



CURVAS HERMITE

$$\mathbf{p}_0 = \mathbf{d}$$

$$\mathbf{p}_1 = \mathbf{a} + \mathbf{b} + \mathbf{c} + \mathbf{d}$$

$$\mathbf{v}_0 = \mathbf{c}$$

$$\mathbf{v}_1 = 3\mathbf{a} + 2\mathbf{b} + \mathbf{c}$$

$$\begin{bmatrix} \mathbf{p}_0 \\ \mathbf{p}_1 \\ \mathbf{v}_0 \\ \mathbf{v}_1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} \mathbf{a} \\ \mathbf{b} \\ \mathbf{c} \\ \mathbf{d} \end{bmatrix}$$



FORMA MATRICIAL DE LAS CURVAS HERMITE

$$\begin{bmatrix} \mathbf{a} \\ \mathbf{b} \\ \mathbf{c} \\ \mathbf{d} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix}^{-1} \cdot \begin{bmatrix} \mathbf{p}_0 \\ \mathbf{p}_1 \\ \mathbf{v}_0 \\ \mathbf{v}_1 \end{bmatrix}$$

$$\begin{bmatrix} \mathbf{a} \\ \mathbf{b} \\ \mathbf{c} \\ \mathbf{d} \end{bmatrix} = \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} \mathbf{p}_0 \\ \mathbf{p}_1 \\ \mathbf{v}_0 \\ \mathbf{v}_1 \end{bmatrix}$$



FORMA MATRICIAL DE LAS CURVAS HERMITE

$$\mathbf{v} = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \cdot \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} p_{0x} & p_{0y} & p_{0z} \\ p_{1x} & p_{1y} & p_{1z} \\ v_{0x} & v_{0y} & v_{0z} \\ v_{1x} & v_{1y} & v_{1z} \end{bmatrix}$$

$$\mathbf{v} = \mathbf{t} \cdot \mathbf{B}_{Hrm} \cdot \mathbf{G}_{Hrm}$$

$$\mathbf{v} = \mathbf{t} \cdot \mathbf{C}$$



CURVAS HERMITE

- Las curvas *Hermite* son otra forma geométrica de la definición de una curva cúbica
- Vemos que, en última instancia, se trata de otra forma de generar coeficientes cúbicos
- Podemos convertir una curva Bézier a/de una forma Hermite con la siguiente relación:

$$\mathbf{C} = \mathbf{B}_{Bez} \cdot \mathbf{G}_{Bez} = \mathbf{B}_{Hrm} \cdot \mathbf{G}_{Hrm}$$



FOTOGRAMAS CLAVE



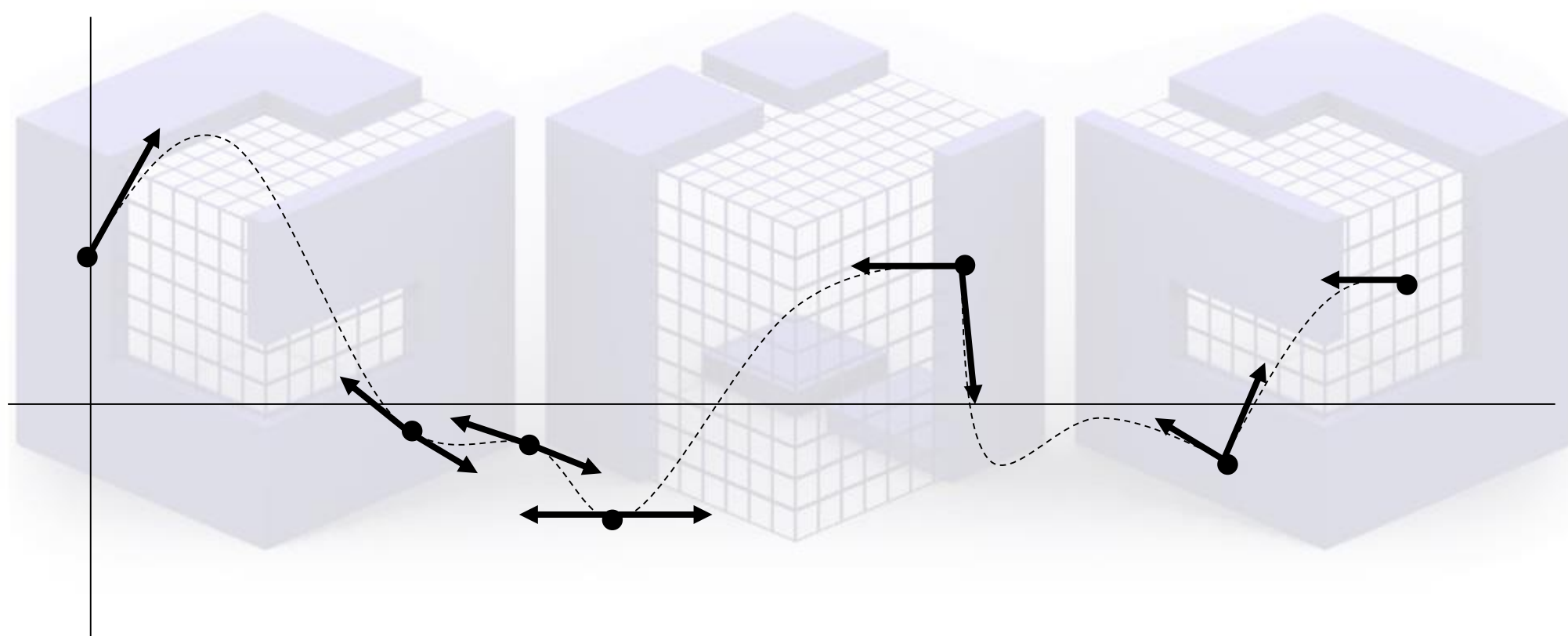


CANALES DE *FRAMES* CLAVES

- Un canal puede almacenarse como una secuencia de fotogramas clave
- Cada fotograma clave corresponde a un tiempo y un valor y, por lo general, las tangentes en ese instante
- Las curvas de los tramos individuales entre las claves se definen por interpolación (por lo general, trozos de *Hermite*)

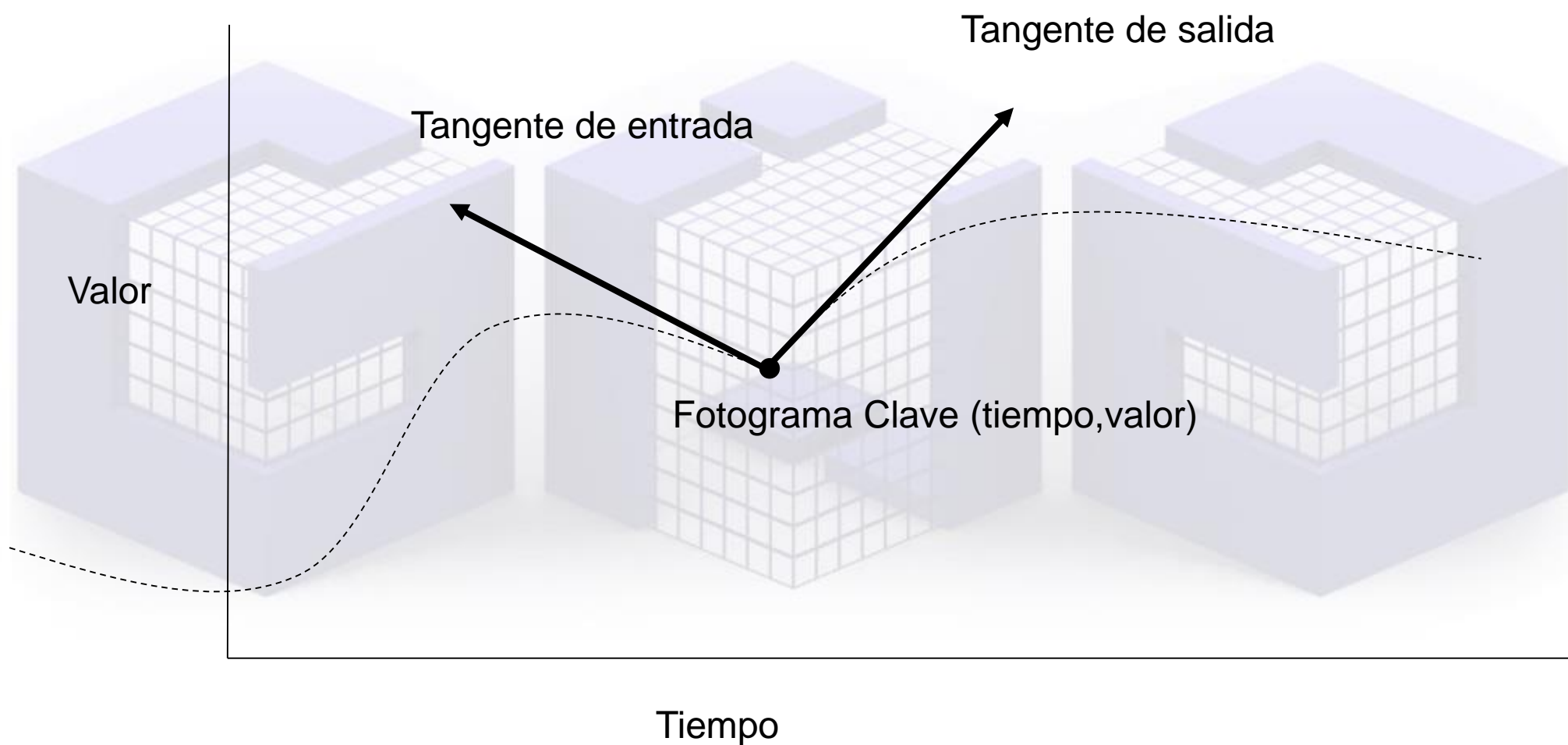


CANALES DE *FRAMES* CLAVES





FOTOGRAMA CLAVE





TANGENTE EN FOTOGRAMA CLAVE

- Los fotogramas clave se representan generalmente de manera que las tangentes de entrada apuntan a la izquierda (antes en el tiempo)
- La flecha dibujada es sólo para la representación visual y hay que recordar que si las dos flechas son exactamente opuestas, en realidad significa que las tangentes son iguales (*misma pendiente y velocidad*)
- También recordar que estamos usando solo curvas 1D, por tanto, la tangente en realidad sólo es solo una pendiente



¿POR QUÉ USAR FOTOGRAMAS CLAVE?

- Son una buena interfaz de usuario para el ajuste de curvas
- Le dan al usuario control sobre el valor de DOFs y la velocidad de cambio
- Definen función perfectamente suaves (si se desea)
- Puede ofrecer buena compresión (no siempre)
- Cada sistema de animación ofrece alguna variación en los fotogramas clave
- Los videojuegos pueden considerar los fotogramas clave para realizar compresión de los datos, a pesar de que tengan un coste en el rendimiento



ANIMANDO CON FOTOGRAMAS CLAVE

Los *canales* de fotogramas clave son la base de la animación de grados de libertad (DOFs) en muchos sistemas de animación comercial

Diferentes sistemas utilizan diferentes variaciones en los cálculos, pero la mayoría se basan en algún tipo de curvas de Hermite cúbicas



ESTRUCTURA DE DATOS PARA FOTOGRAMAS CLAVE

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

Data Structures:

Linked list

Doubly linked list

Array



TANGENTES

- En lugar de almacenar las tangentes de forma explícita, a menudo es más conveniente almacenar una "regla" que las defina y calcular la tangente real cuando sea necesario .
- Por lo general, estas reglas se almacenan de forma separada para las tangentes de entrada y desalida
- Algunas reglas comunes para las tangentes de *Hermite* son:
 - *Constante* (*tangente = 0*)
 - *Lineal* (*tangente apunta al siguiente/anterior valor clave*)
 - *Smooth* (*ajuste automático para curva suave*)
 - *Fijo* (*el usuario puede especificar un valor concreto*)
- Recuerde que la tangente es igual al ratio de cambio del valor del DOF representado (es decir, la velocidad)



COEFICIENTES CÚBICOS

- Los fotogramas clave se almacenan en orden de su tiempo.
- Entre cada dos fotogramas clave sucesivos hay un tramo de curva cúbica.
- Cada tramo está definido por el valor de los dos fotogramas clave y la tangente de salida del primero y la entrante del segundo.
- Estos 4 valores se multiplican por la matriz base de *Hermite* y se convierten en coeficientes cúbicos para el tramo.
- Por simplicidad, se pueden almacenar en el primer fotograma clave



MODOS DE EXTRAPOLACIÓN

En los *canales* de animación se pueden especificar "modos de extrapolación" para definir cómo se extrapola la curva antes y después de t_{min} y t_{max}

Opciones típicas:

- *Valor constante* (mantener el primer / último valor de la clave)
- *Lineal* (usa tangente en primera y última clave)
- *Cíclica* (repetición de todo el canal)
- *Cíclica Offset* (repetir todo el canal con un valor de offset)
- *Rebote* (repetición hacia atrás y hacia delante)



EXTRAPOLACIÓN

Por lo general, los modos de extrapolación se pueden definir de forma separada para antes y después del tiempo donde está establecida la animación

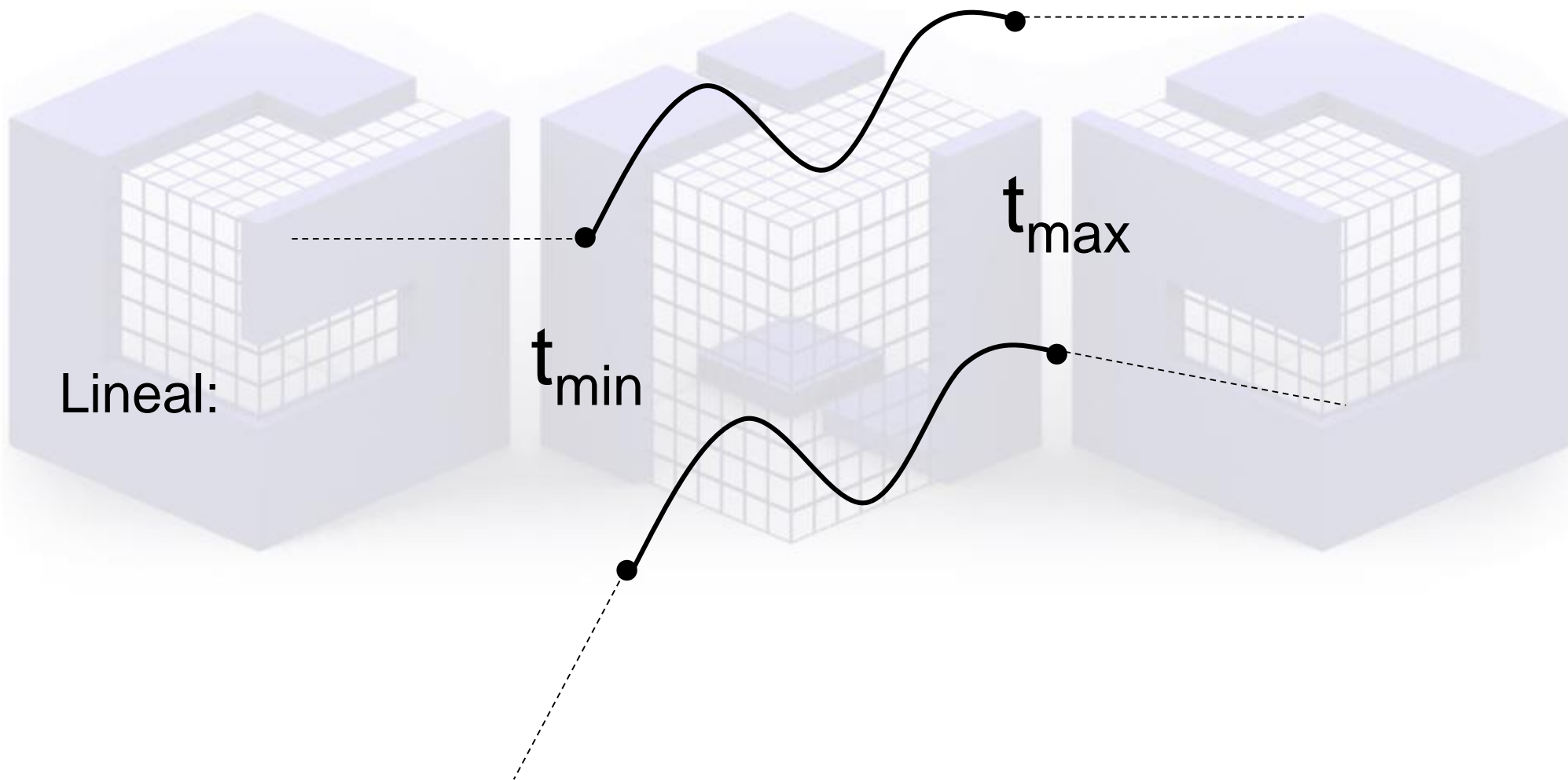
Tenga en cuenta que la extrapolación se aplica a todo el canal y no a las claves individuales

De hecho, la extrapolación no está directamente ligada a los fotogramas clave y se puede utilizar para cualquier método de almacenamiento de canal (por ejemplo, en bruto ...)



EXTRAPOLACIÓN

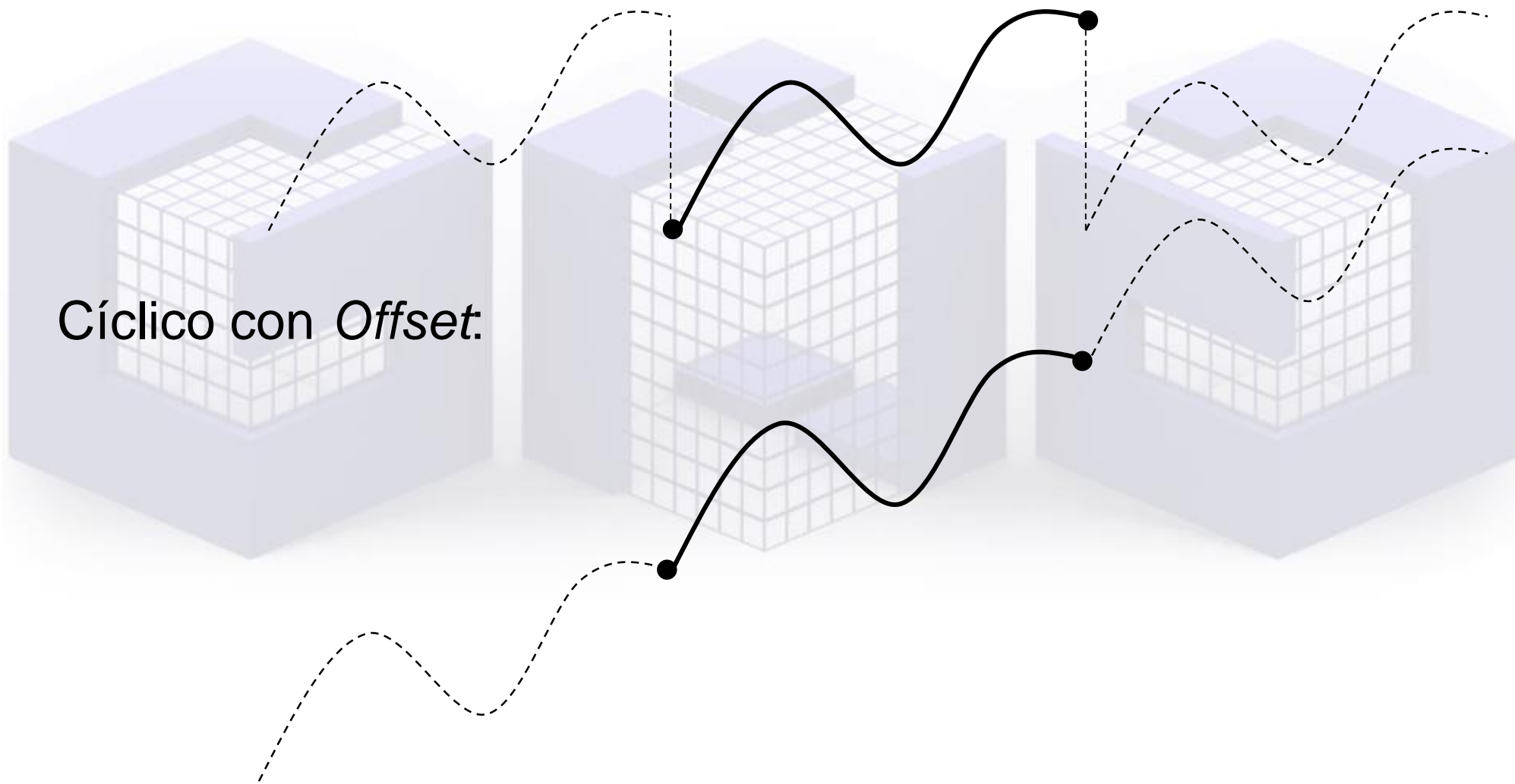
Constante:





EXTRAPOLACIÓN

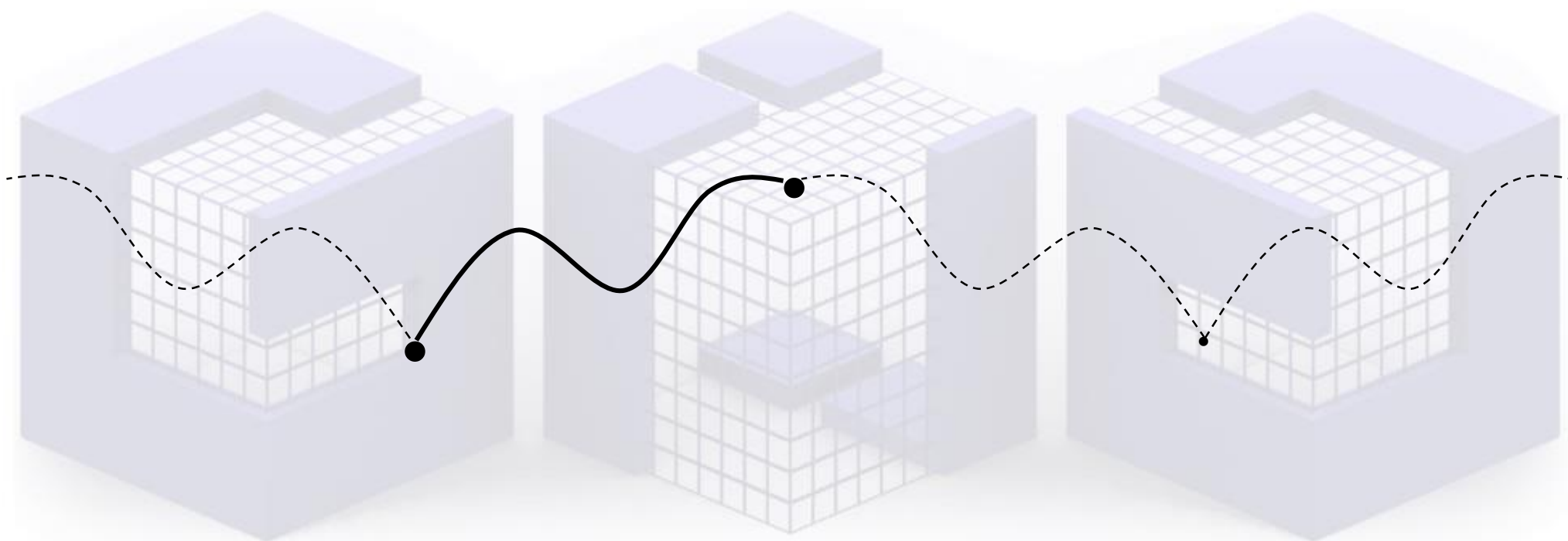
Cíclico:





EXTRAPOLACIÓN

Rebote:





EVALUACION DE LOS FRAMES CLAVE

La principal función en tiempo de ejecución para un canal será:

float Channel::Evaluar (float t);

Esta función se llamará continuamente en ejecución para realizar la animación.

En cada instante de tiempo t , hay 4 casos a considerar:

- t está antes de la primera clave (usamos extrapolación)
- t está después de la última clave (usamos extrapolación)
- t cae exactamente en alguna clave (devolvemos valor clave)
- t cae entre dos claves (evaluamos la ecuación cúbica)



ACCESO ALEATORIO

- Para evaluar un canal en algún instante arbitrario t , tenemos que encontrar primero el tramo correspondiente del canal y luego evaluar su ecuación
- Como los fotogramas clave pueden estar espaciados irregularmente, tenemos que realizar una búsqueda
- Si los fotogramas clave se almacenan como una lista enlazada, hay poco que podamos hacer excepto recorrerla lista en busca del tramo correspondiente
- Si se almacenan en un array, podemos utilizar una búsqueda binaria, lo que debería funcionar razonablemente bien



ACCESO SECUENCIAL

- Cuando reproducimos la animación de un personaje, debemos acceder a la secuencia de datos de canal de forma secuencial.
- Hacer una búsqueda binaria para cada evaluación para cada clave no es eficiente
- En este caso, si hacemos un seguimiento de la clave a la que se ha accedido más recientemente para cada canal, es muy probable que el siguiente acceso requerirá que sea la misma clave o la siguiente
- Esto hace que el acceso secuencial a fotogramas clave sea potencialmente lo más rápido.
- Sin embargo hay un problema...



ACCESO SECUENCIAL

- Consideremos un caso en el que tenemos un juego con 20 personajes corriendo.
- Todos ellos necesitan seguramente acceder a los mismos datos de la animación (que sólo se deberían almacenar una vez obviamente)
- Sin embargo, cada uno podría estar accediendo a ellos con un tiempo diferente (para evitar que todos se muevan a la vez).
- El código que reproduce las animaciones necesita realizar un seguimiento de las claves más recientes en cada canal para cada personaje en lugar de la solución más simple de sólo tener un puntero a su clave más reciente.
- Por lo tanto, la clase reproductora de la animación tiene que hacer un cálculo importante, ya que tendrá que hacer un seguimiento de la clave más reciente para **todos** los canales.



AJUSTE DE CURVAS

- Los fotogramas clave se pueden generar automáticamente a partir de datos registrados, como por ejemplo mediante captura de movimiento.
- Este proceso se realiza mediante un *ajuste de curvas*, ya que implica la búsqueda de las curvas que se ajustan mejor a los datos
- Los algoritmos de ajuste permiten al usuario especificar las tolerancias que definen la calidad del ajuste.
- Esto permite la conversión bidireccional entre fotogramas clave y formatos *brutos*, aunque los datos podrían quedar ligeramente alterados con cada conversión y el resultado habrá que *limpiarlo*



EVALUANDO CANALES DE FOTOGRAMAS CLAVE

- Como se ha mencionado, con el fin de evaluar cada canal, primero hay que encontrar el tramo apropiado por algún tipo de algoritmo de búsqueda.
- Una vez que conocemos el tramo, tenemos que reparametrizar el tiempo para convertirlo en un intervalo $0 \dots 1$ que es el usado en cada tramo.
- Hacemos esto mediante la *interpolación lineal inversa*



INTERPOLACIÓN LINEAL INVERSA

Si t_0 es el tiempo en la primera clave y t_1 es el tiempo en la segunda clave, la interpolación de esos tiempos en base a un parámetro u será:

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

La inversa de dicha operación viene dada por la expression:

$$u = InvLerp(time, t_0, t_1) = (time - t_0) / (t_1 - t_0)$$

Esto nos da un valor de u entre $0 \dots 1$ en el tramo donde ahora evaluaremos la ecuación cúbica

Nota: $1/(t_1 - t_0)$ puede precalcularse para mejorar la eficiencia



EVALUANDO LA ECUACIÓN CÚBICA

Evaluar la ecuación cúbica es sencillo:

$$x = Au^3 + Bu^2 + Cu + D$$

O su version rápida:

$$x = D + u * (C + u * (B + u * (A)))$$



PRECALCULO DE LAS CONSTANTES

Los coeficientes cúbicos son constantes y pueden ser precalculados

Los términos **$(t_1 - t_0)$** sirven para normalizar la pendiente sobre el rango $0 \dots 1$ de **t** en lugar del rango real:

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