# Splines: Curvas y Superficies

**Introducción al dibujo de curvas de aproximación e interpolación por computador.**

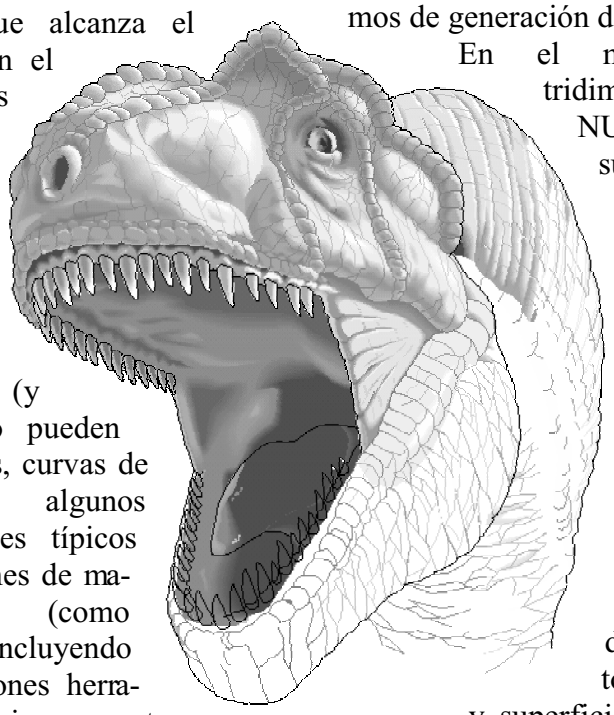
Carlos González Morcillo (*Carlos.Gonzalez@uclm.es*)

## Índice

# 1. Introducción.

En este texto se intenta dar una introducción al estudio de las curvas de interpolación generadas por ordenador. El estudio de las curvas y superficies data de la época de los primeros computadores. Se intentaba simular los fenómenos físicos reales con la mayor perfección posible.

Lo que menos impera en el mundo real son las líneas rectas. Tanto la forma de los objetos que nos rodean, como las trayectorias de los móviles, obedecen a líneas y superficies curvas.



La dimensión que alcanza el trabajo con curvas en el mundo de los computadores es enorme. Todos los programas de dibujo vectorial soportan el trabajo con varios tipos de curva de interpolación (y aproximación), como pueden ser Splines, B-Splines, curvas de Béizer... Incluso algunos programas comerciales típicos de diseño con imágenes de mapas de píxeles (como Photoshop), están incluyendo en sus últimas versiones herramientas para el trabajo con este tipo de curvas. La imagen que ilustra estas líneas fue generada con un programa de diseño vectorial, Macromedia FreeHand.

A nivel industrial, desde hace años se trabaja con curvas splines y B-Splines. Un ingeniero de Renault, dio nombre a un tipo de curvas ampliamente utilizadas hoy en día, las curvas de Béizer. Desde el diseño de carrocerías para automóviles, pasando por cascos para embarcaciones hasta llegar a zapatillas deportivas de competición, se podría decir que todo el sector del diseño utiliza este tipo de curvas. Incluso se utilizan en las imprentas actuales (las fuentes que utiliza $T_EX$ son generadas mediante curvas de Béizer y Splines antes de mandarlas a la célula de impresión). Incluso, cuando buscaba información en Internet para preparar estos apuntes, he encontrado proyectos de Robótica industrial y micro-robótica que utilizan splines para calcular trayectorias óptimas en movimientos. Los videojuegos y en el mundo de la DemoScene no se quedan sin aplicación directa de estas curvas; el cálculo de la posición de las cámaras que captan la escena y la elaboración de algunos efectos en Demos (deformaciones sobre todo) se consiguen aplicando algoritmos de generación de splines.

En el modelado de objetos tridimensionales se utilizan NURBS, que son superficies generadas mediante Splines y B-Splines. Prácticamente todos los paquetes de diseño y animación tridimensional soportan este tipo de trabajo (Blender, 3DStudioMAX, Maya...).

Como cabe esperar, en este texto no se pueden dar todos los detalles del desarrollo de todos los tipos de curvas y superficies que se utilizan en el diseño por computador. Muchas cosas se han quedado en el tintero, por falta de tiempo. Se podría dedicar todo un cuatrimestre al estudio de este área y nos faltarían muchas cosas por ver.

Me he centrado en el estudio de un tipo de curvas, las splines cúbicas naturales. Se ha desarrollado por completo el análisis matemático de este tipo de curvas junto con un programa de ejemplo implementado en C. Si queréis profundizar en el algoritmo de generación de algún tipo de curva, recomiendo los dos primeros libros citados en la bibliografía (*Watt* y *Farin*).

## 2. Representación paramétrica

La representación paramétrica es mucho más utilizada en la generación de gráficos por ordenador que la representación implícita. Sobre todo es especialmente útil en la creación de curvas y superficies.

Ambas representaciones (paramétrica e implícita) son formas analíticas. ¿Por qué la forma paramétrica es «mejor» que la implícita?. Veamos un ejemplo...

Supongamos que queremos dibujar un círculo con centro en el origen de coordenadas y radio la unidad. En su forma implícita (la que estamos acostumbrados a utilizar), se expresaría con una función cartesiana del tipo:

$$x^2+y^2=1^2 \qquad z=0$$

Frente a su representación paramétrica:

$$R(u) = (\cos(\frac{u}{2\pi}), \text{sen}(\frac{u}{2\pi}), 0)$$

Al dibujar el círculo, en forma paramétrica tendremos una manera sencilla de conectar puntos que estén en el círculo. variando u entre 0 y 1 (en la forma paramétrica, la variable u se define siempre dentro de ese intervalo), incrementando el parámetro u poco a poco desde 0 hasta 1, conseguiremos una serie de puntos que forman parte del contorno de la circunferencia. Bastará con tomar los intervalos lo suficientemente pequeños para que, con una serie de líneas rectas, podamos dibujar la circunferencia sin que se noten los segmentos.

Sin embargo, la representación con coordenadas cartesianas no es buena para utilizarla en la implementación en un ordenador. Para cada punto, habría que resolver dos raíces cuadradas (lo que nos da un coste computacional muy alto).Además, no sólo es ineficiente sino que tampoco es implementable; ya que las raices cuadradas nos dan más de una solución para la ecuación, por lo que habría que descartar la solución que no vale.

En general, en la representación implícita (o no paramétrica), las curvas y superficies

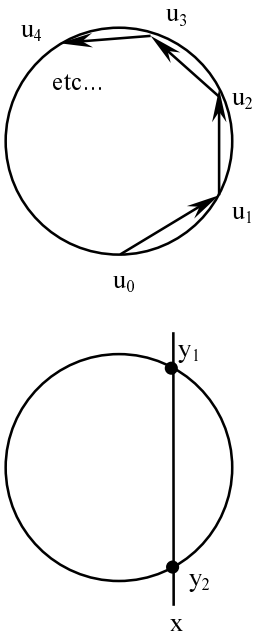



**Figura 1**. En la circunferencia superior se observa la construcción a base de unir puntos obtenidos de la ecuación en forma paramétrica, variando u. La circunferencia inferior muestra el problema principal de la representación implícita; en este caso, a cada valor de x le corresponden 2 valores de y.

presentan el problema de múltiples soluciones en su ecuación, por lo que el trazado no es posible.

Podemos expresar cada una de las 3 coordenadas cartesianas de una curva en función del parámetro u, definido normalmente en el intervalo [0,1], de forma general:

$$P(u) = (x(u), y(u), z(u))$$

Las superficies son objetos tridimensionales. Las posiciones en una superficie se pueden describir con dos parámetros *u* y *v*. De forma general, una superficie se puede representar con la función paramétrica:

$$P(u,v) = (x(u,v), y(u,v), z(u,v))$$

donde los valores de las coordenadas cartesianas x,y,z se expresan como funciones de los parámetros u,v. Al igual que ocurría en las curvas, con frecuencia es posible definir la función paramétrica de modo que los parámetros u,v estén dentro del intervalo [0,1].

## 3. Representaciones de Spline.

Hace años en dibujo técnico, una spline era una banda flexible de madera que se utilizaba para hacer pasar una curva por un conjunto de puntos. Para ello, el delineante colocaba una serie de pesos sobre esta «regla» y trazaba luego la curva deseada.

Esa curva obtenida podemos expresarla de forma matemática con una función polinómica cúbica cuyas primera y segunda derivadas son continuas a través de las distintas secciones de la curva.

En la jerga del diseño asistido por computador, el término «curva de spline» se refiere a cualquier curva compuesta que se forma con secciones polinómicas que satisfacen ciertas condiciones específicas de continuidad en la frontera de cada intervalo.

Existen varias clases de especificaciones de splines que se usan en las aplicaciones gráficas. Cada especificación difiere del resto en el polinomio particular que utiliza junto con las condiciones de frontera (o condiciones de extremos) que requiere.

Recordemos que una función polinómica de n-ésimo grado se define como:

$$y = a_0 + a_1 x + a_2 x^2 + \ldots + a_{n-1} x^{n-1} + a_n x^n$$

donde n es un entero no negativo y $a_k$ son constantes. Siendo $a_n \neq 0$, si n=2 la función es cuadrática; si n=3 la función tenemos un polinomio cúbico... (si n=1 tendremos la ecuación de una recta).

Dado un conjunto de puntos, podemos ajustar una curva que pase por ellos. Se construye una sección curva (con un polinomio cúbico) entre cada par de puntos dado. Así, cada sección de curva se define como:

$$x = a_{x_0} + a_{x_1} u + a_{x_2} u^2 + a_{x_3} u^3$$

$$y = a_{y_0} + a_{y_1} u + a_{y_2} u^2 + a_{y_3} u^3$$

Que se podría escribir de forma equivalente como:

$$x(u) = a_x u^3 + b_x u^2 + c_x u + d_x$$

$$y(u) = a_y u^3 + b_y u^2 + c_y u + d_y$$

donde u varía entre 0 y 1. Los valores de u se determinan según las condiciones de frontera en las secciones curvas.

- Dos secciones de curva adyacentes deben tener igual coordenada en los extremos.
- Tendremos que adaptar la pendiente de cada par de curvas adyacentes para obtener una curva global (formada por la unión de varias secciones de curvas) contínua y suave.

## 4. Interpolación y aproximación de splines.

Especificamos una curva spline al proporcionar un conjunto de puntos (que será una serie de coordenadas) a los que denominaremos puntos de control.

Estos puntos de control se ajustarán después con funciones polinómicas contínuas de una de las siguientes maneras:

a) La curva realiza **interpolación** del conjunto de puntos de control cuando las secciones polinómicas se ajustan de modo que la curva pasa a través de cada punto de control.
b) La curva realiza una **aproximación** al conjunto de puntos de control cuando los polinomios se ajustan a la trayectoria general del punto de control sin pasar necesariamente a través de ningún punto de control.

Una curva spline se define y se modifica con operaciones sobre sus puntos de control. Los paquetes de CAD (como Autocad) pueden insertar puntos de control adicionales para ayudar al diseñador en el modelado.

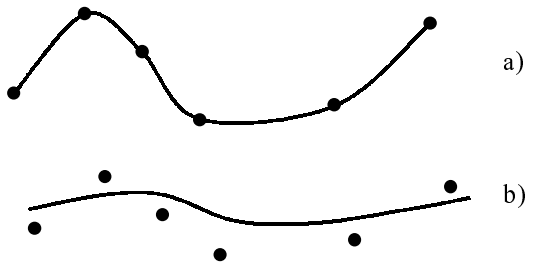



**Figura 2.** a) Interpolación, b) Aproximación

## 5. Splines contra otros métodos de interpolación.

Habitualmente, en cursos de análisis numérico se explican métodos de interpolación más o menos sencillos que pueden servir para satisfacer ciertos problemas.

Un método clásico de interpolación es el polinomio de Lagrange. El término general de la fórmula de interpolación de Lagrange puede escribirse de la siguiente forma:

$$P(x) = \sum_{i=1}^{n} y_i \prod_{\substack{j=1 \\ j \neq i}}^{n} \frac{x - x_j}{x_i - x_j}$$

Vamos a realizar un ejemplo; aproximaremos la función

$$y = \frac{12}{x^2 + 2x + 5}$$

mediante el polinomio de Lagrange y también mediante splines cúbicas naturales. En la figura 3, podemos observar los resultados del experimento. En la zona superior están las gráficas resultantes de Lagrange y en la zona inferior las resultantes de splines. La gráfica punteada es la exacta a la función original.

Centrándonos en el polinomio de Lagrange; con 4 puntos el polinomio conseguido es de tercer grado (la aproximación es mala; únicamente se limita a pasar por los puntos).

Con cinco puntos, el polinomio es de cuarto grado; la aproximación solo es buena entre el segundo y tercer punto (empezando desde la izquierda). Con 10 puntos, aparecen unas ondulaciones lejos del máximo de la función. Este fenómeno, se denomina «fenómeno de Runge», y aumenta su actividad cuando se incrementa el número de puntos.

Usando splines cúbicas, la interpolación conseguida con 10 puntos es prácticamente perfecta.

Conclusiones:

- Hay que realizar muchas operaciones aritméticas en otros métodos de interpolación (como el de Lagrange), ya que habrá 2 bucles entrelazados, uno para el sumatorio y otro para el productorio.
- Si queremos añadir o suprimir un punto al conjunto de datos, habrá que volver a hacer todos los cálculos (este problema también lo presentan las splines naturales, aunque otras splines no lo tienen).
- En ciertos casos, un polinomio de grado alto puede desviar mucho la curva que pasa por los puntos dados (fenómeno de Runge).

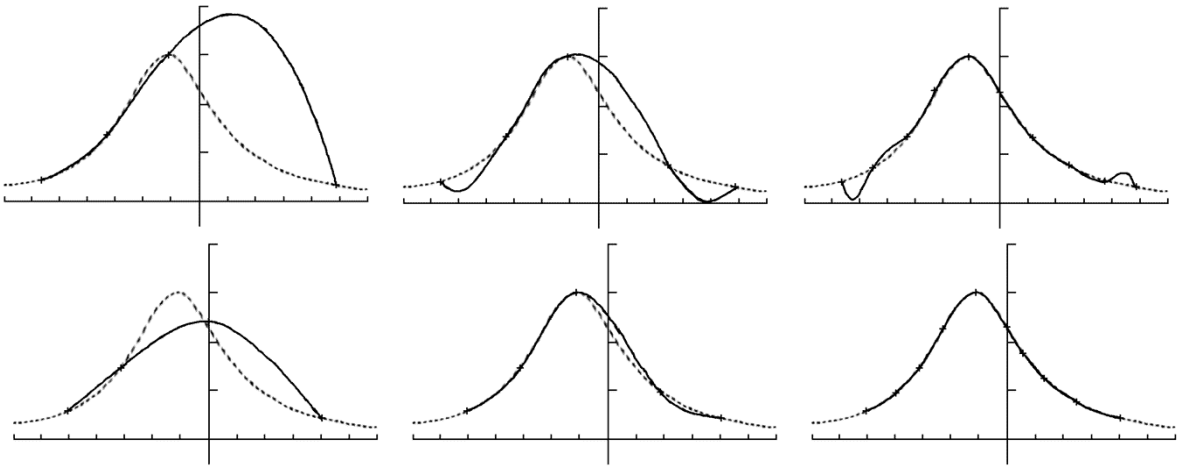


**Figura 3.** Las gráficas de la parte superior corresponden (de izquierda a derecha) a la interpolación de Lagrange con 4, 5 y 10 puntos. Observar cómo con 10 puntos, el fenómeno de *Runge* se aprecia claramente. Las gráficas de la parte inferior corresponden a una interpolación con splines cúbicas naturales. La interpolación conseguida con 5 puntos es muy buena. Con 10 puntos (derecha-abajo), la interpolación es perfecta.

## 6. Condiciones de continuidad paramétrica.

Para asegurar una transición suave en la curva de un intervalo al siguiente, impondremos una serie de condiciones de continuidad en los puntos de conexión.

Establecemos la continuidad al comparar las derivadas de secciones de curvas adyacentes en su frontera común.

- Continuidad de **orden cero** $C^0$; únicamente implica que las curvas se unen.
- Continuidad de **primer orden** $C^1$; las primeras derivadas (tangentes a la curva) son iguales en su punto de unión.
- Continuidad de **segundo orden** $C^2$; además de lo dicho para $C^1$, se tiene que cumplir que la variación de los vectores tangente según nos acercamos al punto de unión de ambas curvas por la derecha y por la izquierda es equivalente. Esto se logra forzando a que la segunda derivada sea igual en la frontera de ambas curvas. Así, se consigue una transición suave de una sección de curva a la siguiente.

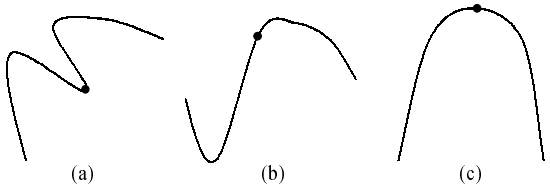



**Figura 4.** Construcción por piezas de una curva al unir dos segmentos curvos. En (a) la continuidad es de orden cero, (b) continuidad de primer orden y (c) de segundo orden.

## 7. Métodos de interpolación de spline cúbica.

Los polinomios cúbicos ofrecen una relación buena entre flexibilidad y velocidad de cálculo; comparando con polinomios de orden superior, las splines cúbicas requieren menos cálculos y memoria, a la vez que són más estables (recordemos el fenómeno de Runge que presentaba el método de Lagrange).

Dado un conjunto de puntos de control, las splines de interpolación cúbica se obtienen de ajustar los puntos de entrada con una curva que pase a través de todos los puntos de control. Supongamos que tenemos n+1 puntos de control que están definidos con las coordenadas:

$$P_k=(x_k,y_k,z_k) \qquad k=0,1,2,...,n$$

El polinomio cúbico que debe ajustarse entre cada par de puntos viene dado por las ecuaciones:

$$x(u) = a_x u^3 + b_x u^2 + c_x u + d_x$$
$$y(u) = a_y u^3 + b_y u^2 + c_y u + d_y$$
$$z(u) = a_z u^3 + b_z u^2 + c_z u + d_z$$

Recordemos que $0 \le u \le 1$. Únicamente nos falta determinar los valor de a,b,c,d (en cada eje de coordenadas). Esto se consigue estableciendo suficientes condiciones frontera en las «uniones» de las secciones de curva.

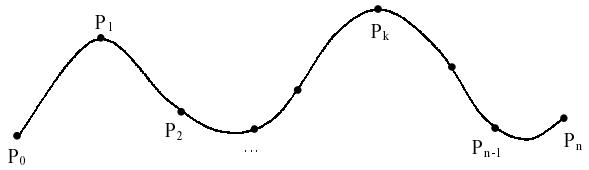



**Figura 5**. Interpolación por piezas de spline cúbica (n+1 puntos de control).

### 7.1. Splines cúbicas naturales.

Es una representación matemática de la spline del dibujo técnico original. Requerimos que dos secciones curvas adyacentes tengan tanto la primera como la segunda derivada igual en su frontera común; es decir, exigiremos continuidad $C^2$.

Si tenemos n+1 puntos de control, habrá n secciones de curva, con 4n coeficientes a determinar (según vimos en la ecuación del apartado anterior). En los n-1 puntos interiores ya tenemos las 4 condiciones de frontera; las dos secciones curvas en cualquier lado de un punto de control debe tener tanto la primera como la segunda derivada iguales en ese punto de control. Así tenemos (4n-4) ecuaciones. Nos faltan pues, 4 ecuaciones más para poder determinar el valor de los 4n coeficientes.

Obtenemos dos nuevas ecuaciones de $P_0$ y $P_n$; los puntos de inicio y fin de la curva. Para definir las 2 condiciones que nos faltan hay

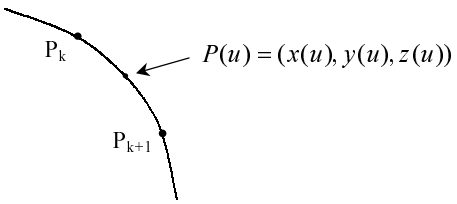
varias alternativas. La tomada por las splines cúbicas naturales es considerar 0 las segundas derivadas $P''_0$ y $P''_n$.

Otros métodos consisten en agregar 2 puntos de control «simulados» al principio y final de la curva $P_{-1}$ y $P_{n+1}$. De esta forma, todos los puntos son interiores.

Desventaja de splines cúbicas naturales: no permiten control local de la curva. Es decir, si se altera la posición de cualquier punto de control, afecta a la curva entera (teniendo que rehacer cálculos). Esta desventaja se aprecia claramente en el programa de ejemplo de la *sección 11*. Si modificamos la posición de cualquier punto de control, se puede ver un cambio (aunque sea leve).

## 7.2. Splines de Hermite.

Toman su nombre del matemático francés Charles Hermite. Su característica principal es que poseen una tangente específica en cada punto de control. Permiten control local.




La expresión general de las splines de Hermite es:

$$x(u) = x_k(2u^3 - 3u^2 + 1) + x_{k+1}(-2u^3 + 3u^2) + \\ + D_k(u^3 - 2u^2 + u) + D_{k+1}(u^3 - u^2)$$

Análogamente se definiría para y,z.

En la expresión anterior:
$x_k$ es el valor de x en el punto $P_k$
$x_{k+1}$ es el valor de x en el punto $P_{k+1}$
$D_k$ es el valor de la primera derivada en $P_k$
$D_{k+1}$ es el valor de la primera derivada en $P_{k+1}$

Las splines de Hermite pueden ser útiles para algunas aplicaciones donde no sea difícil aproximar o dar valores a las pendientes de la curva.

Sin embargo, generalmente es más útil generar valores para las pendientes de forma automática, sin requerir la entrada por parte del usuario.

Estos cálculos de pendientes se suelen hacer en función de las posiciones de los puntos de control.

## 7.3. Splines cardinales.

Al igual que las de Hermite, tienen tangentes fijas en la frontera de cada sección de curva. Sin embargo, no la tenemos que dar nosotros (se calcula con base en las coordenadas de los dos puntos de control adyacentes).

Para especificar cada sección de curva, es necesario 4 puntos de control consecutivos. Los 2 puntos centrales son los extremos de la sección y los otros dos puntos sirven para calcular la pendiente de los extremos.

$$P'(k) = \frac{(1-t)}{2}(P_{k+1} - P_{k-1})$$

$$P'(k+1) = \frac{(1-t)}{2}(P_{k+2} - P_k)$$

Por tanto, las pendientes en los puntos de control $P_k$ y $P_{k+1}$ son proporcionales a las cuerdas $P_{k-1}P_{k+1}$ y $P_kP_{k+2}$. t es el «parámetro de tensión». Cuando t=0, se denominan «curvas splines de *Overhauser*».

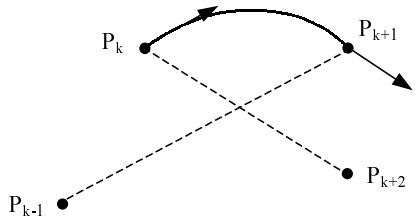



**Figura 6**. Los vectores de tangente en los extremos de la sección son proporcionales a las cuerdas $P_{k-1}P_{k+1}$ , $P_kP_{k+2}$ (líneas punteadas)

Dependiendo del valor de t, la curva estará más o menos tensa. Si t>0, la curva será más tensa.

La ecuación general de las splines cardinales es de la forma:

$$P(u) = P_{k+1}(-su^3 + 2su^2 - su) + \\ + P_k\left[(2-s)u^3 + (s-3)u^2 + 1\right] + \\ + P_{k+1}\left[(s-2)u^3 + (3-2s)u^2 + su\right] + \\ + P_{k+2}(su^3 - su^2) \quad siendo \;\; s = (1-t)/2$$

## 8. B-Spline.

La B-spline (usadas en CAD por Riesenfeld en 1973) generalmente tiene una naturaleza no global. Cada vértice del polígono generador de la curva está asociado a una función. De esta manera, cada vértice tiene influencia sobre la forma de la curva en un intervalo limitado, en el cual la función asociada es distinta de cero.

Las ecuación principal que define a las B-Spline es:

$$P(t) = \sum_{i=0}^{n} P_i N_{i,j}(t)$$

En esta fórmula, los $P_i$ son los n+1 vértices del polígono y las $N_{i,j}(t)$ son las funciones mezcla que se definen como:

$$N_{i,j}(t) = \begin{cases} 1 \quad si \quad x_i \le t \le x_{i+1} \\ 0 \; en \; los \; demás \; casos \end{cases}$$

$$N_{i,j}(t) = \frac{(t - x_i) N_{i,j-1}(t)}{x_{i+j-1} - x_j} + \frac{(x_{i+j} - t) N_{i+1,j-1}(t)}{x_{i+j} - x_{j+1}}$$

Siendo j el orden de la curva, que es siempre un entero que cumple la siguiente desigualdad:

$$2 \le j \le n+1$$

Los valores de $x_i$ son elementos de un vector nudo. Un vector nudo es una secuencia de enteros positivos $x_i$ tales que $x_i \le x_{i+1}$. Por ejemplo, [012345] y [001233] son dos vectores nudo.

El orden j de la curva está reflejado en el vector nudo, el cual se puede calcular de la siguiente manera:

$$x_i = 0 \qquad si \; i < j$$
$$x_i = i - j + 1 \qquad si \; j \le i \le n$$
$$x_i = n - j + 2 \qquad si \; i > n$$

Siendo n = (nº vértices polígono - 1)

El parámetro t puede variar en el intervalo [0,tmax], donde tmax es el valor máximo de los elementos del vector nudo. Por ejemplo, el vector nudo [00123455] indica que el parámetro t puede variar de 0 a 5. Si hubiera algún vértice repetido (múltiple), el valor de tmax viene dado por la relación tmax=n-j+2.

| n | j | vector nudo | tmax |
|---|---|---|---|
| 3 | 2 | 001233 | 3 |
| | 3 | 0001222 | 2 |
| | 4 | 00001111 | 1 |
| 6 | 2 | 001234566 | 6 |
| | 3 | 0001234555 | 5 |
| | 4 | 00001234444 | 4 |
| | 5 | 000001233333 | 3 |
| | 6 | 0000001222222 | 2 |
| | 7 | 00000001111111 | 1 |

**Tabla 1.** Ejemplos de cálculo de vector nudo.

## 8.1. Propiedades de las B-Splines.

1. Una B-Spline pasa por el primer y último punto de control.

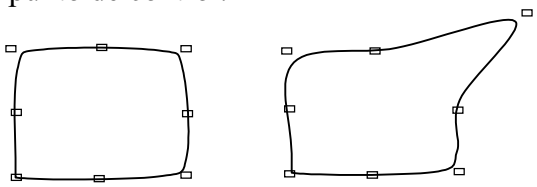


**Figura 7**. Control local de las B-Splines.

2. El control de la curva es local. Cada vértice afecta a la forma de la curva en un intervalo dado (*ver Figura 7*).
3. Es posible cambiar el orden de la curva sin cambiar el número de vértices del polígono. Las splines de orden 4 son las más utilizadas en las aplicaciones de diseño.
4. Si el orden de la curva es igual al número de vértices del polígono, la B-Spline recibe el nombre de curva de Bézier.
5. Las B-Splines permiten crear curvas con «esquinas». Para ello, bastará con introducir vértices múltiples.
6. Una B-Spline de orden 3 siempre es tangente a la parte media de los lados del polígono (*Figura 8*).

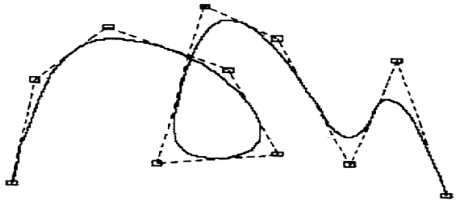


**Figura 8**. Ejemplo de B-Spline de orden 3.

## 9. Conceptos generales de superficies.

En el apartado de representación paramétrica ya se dijo que las superficies se creaban variando los parámetros u y v (entre 0 y 1) en la ecuación genérica:

$$P(u,v) = (x(u,v), y(u,v), z(u,v))$$

Esto nos genera una cuadrícula tridimensional que coloquialmente se denomina parche (del inglés *patch*). Las más variadas superficies se modelan utilizando redes de parches.

Al igual que las curvas complejas que hemos estudiado se generaban con segmentos de curvas más sencillas (las splines cúbicas con segmentos de polinomios cúbicos, por ejemplo), las superficies se crean con redes de parches.

Para obtener superficies con redes de parches, seguiremos la siguiente ecuación:

$$Q(u,v) = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} p_{ij} T(u,v)$$

$p_{ij}$ es el conjunto de puntos de control (puntos de unión entre parches que forman la red). Con T(u,v) denotamos una colección de polinomios linearmente independientes que toman valor exponencial respecto de u y de v ($u^i$, $v^j$ respectivamente).

n es el número de puntos de control en cada dirección (x, y). En el ejemplo de la figura 10, n valdría 4 en ambos sentidos (la red tiene 4x4 puntos de control). De esta forma, ambos sumatorios estarían definidos de 0 a 3.

La representación mediante parches tiene importantes ventajas, la principal es que resulta exacta y económica. El coste de almacenamiento de los puntos de control es mucho menor que si tuviéramos que almacenar todas las intersecciones de la malla resultantes. Además, utilizando parches, siempre podremos calcular con mayor precisión (o menor, según se necesite) la forma de la superficie resultante. Esta forma de trabajar se utiliza en todos los programas CAD.

Existen otras ventajas como la variación suave. En la figura 10 hay un claro ejemplo de edición de mallas por puntos de control.

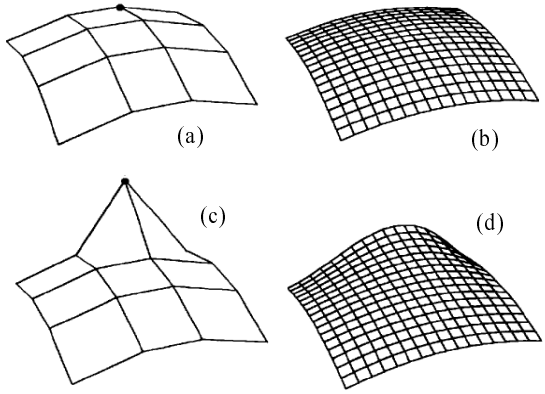



**Figura 10.** Podemos observar el efecto de «tirar» de un punto de control de una red de parches. En (a) la red de parches (formada por 16 puntos de control), da lugar a la malla resultante de aplicar el algoritmo de Béizer. Al desplazar el punto de control seleccionado (c), la malla (d) muestra la modificación suavemente

No necesitaremos cambiar uno a uno todos los puntos de intersección en la malla; moviendo el punto de control de la red de parches, la malla resultante variará de forma suave. Esta técnica se utiliza en programas punteros de diseño tridimensional como ClothReyes (el plug-in de 3DStudioMax creado por Rem Infográfica).

La mayoría de los programas de diseño tratan la generación de superficies como redes de parches, excepto las que tengan que ser totalmente planas (que se generan como un único parche con la fórmula inicial).

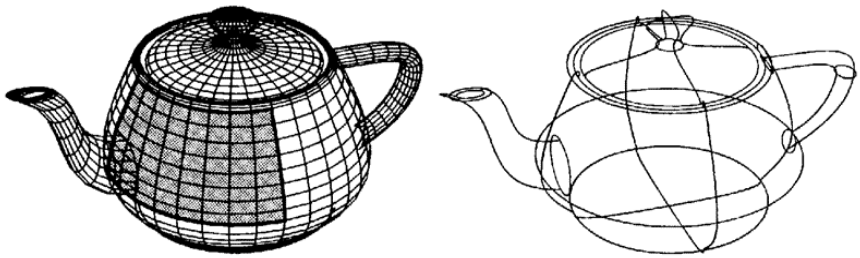


**Figura 9**. La típica tetera de todos los programas de diseño 3D. Está diseñada con 32 parches de Béizer. Un parche está sombreado en la imagen de la izquierda. En la derecha tenemos un modelo de alambre de los puntos de control (realizado con curvas de Béizer).

## 10. Implementación de splines cúbicas naturales.

Como ya hemos visto en otros apartados, la representación *paramétrica* de cada segmento que forman las splines cúbicas es del modo:

$$Y_i(u) = a_i + b_i u + c_i u^2 + d_i u^3 \quad (Ec.1.1)$$

De forma análoga se definiría la ecuación para X y Z (si fuera una curva en tres dimensiones). Como también se comentó, u varía desde 0 a 1 en cada segmento de la curva.

Como se ve en la *Fig.11*, y acorde con la fórmula descrita anteriormente, el segmento i-ésimo va desde el punto de control $V_i$ hasta el $V_{i+1}$. En la ecuación 1.1, $Y_i(u)$ representa y(u) a lo largo del segmento i-ésimo. De forma similar podríamos definir $X_i(u)$ y $Z_i(u)$.

Como u varía desde 0 hasta 1 a lo largo de cada segmento de curva, entonces $Y_i(u)$ nos dará el punto de control i-ésimo ($y_i$) cuando u=0, e igualmente, el punto de control yi+1 cuando u=1. Por tanto, $X_i(u) = x_i$ cuando u=0 y $X_i(u) = x_{i+1}$ cuando u=1.

Las cuatro incógnitas de la ecuación 1.1, que estarán presentes en cada segmento, se determinarán «emparejando» los puntos de control, forzando la continuidad de las primera y segunda derivadas en los puntos interiores y aplicando ciertas condiciones (que veremos más adelante) para los puntos extremos de la curva; primer y último punto.

A lo largo de cada segmento (tomaremos el segmento de curva i-ésimo), con el valor de los puntos extremos, obtendremos las siguientes ecuaciones:

$$Y_i(0) = y_i = a_i \quad (Ec.1.2)$$

$$Y_i(1) = y_{i+1} = a_i + b_i + c_i + d_i \quad (Ec.1.3)$$

Recordemos que el número de puntos de control es n, y el número de segmentos es m (m = n-1). Las 2 ecuaciones anteriores las obtendremos de todos los puntos de control interiores; por lo que tendremos 2m ecuaciones para resolver las 4m incógnitas que se nos plantean inicialmente.

Forzando a que las primeras y segundas derivadas sean iguales (respectivamente) en los extremos de cada intervalo, obtendremos otras 2m-2 ecuaciones.

Nos faltan pues, 2 ecuaciones para resolver el sistema. Como ya hemos comentado, existen varias opciones para definirlas. Las splines cúbicas naturales toman su segunda derivada valor cero en los puntos inicial y final. Es decir;

$$Y_0''(0) = Y_{m-1}''(1) = 0$$

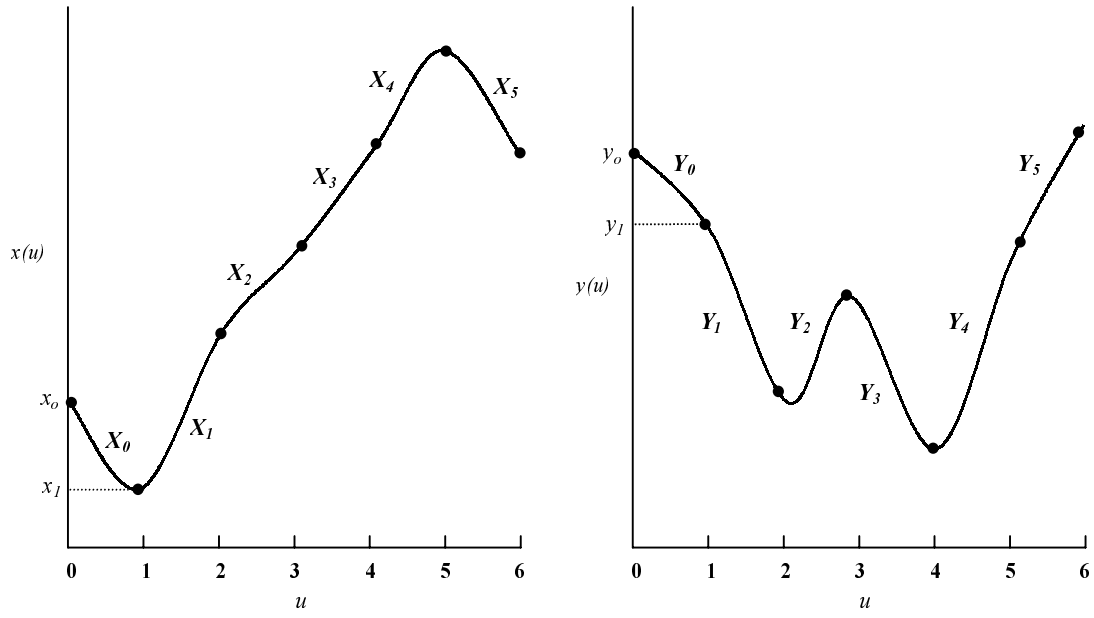



**Figura 11.** Notación usada para nombrar segmentos en una spline cúbica de 2 dimensiones.

Para resolver las incógnitas que tenemos, resulta útil y cómodo, introducir nuevas constantes $D_i$ que representarán el valor de la primera derivada en el punto de control i. El valor de la primera derivada es:

$$Y_i'(u) = b_i + 2c_i u + 3d_i u^2$$

La primera derivada en los extremos del segmento i, nos da las siguientes ecuaciones:

$$Y_i'(0) = D_i = b_i \quad (Ec.1.4)$$

$$Y_i'(1) = D_{i+1} = b_i + 2c_i + 3d_i \quad (Ec.1.5)$$

La ecuación 1.4 y 1.5 nos darán realmente 2m ecuaciones (serán 2 para cada intervalo que definen los puntos interiores). Multiplicando la ecuación 1.3 por 3 y restándole 1.5 tendremos que...

$$\begin{array}{r} 3y_{i+1} = 3a_i + 3b_i + 3c_i + 3d_i \\ D_{i+1} = b_i + 2c_i + 3d_i \quad - \\ \hline 3y_{i+1} - D_{i+1} = 3a_i + 2b_i + c_i \end{array}$$

Recordando las ecuaciones 1.2 que nos daba el valor de $a_i$, y la 1.4 el valor de $b_i$ respecto de $D_i$; sustituyendo:

$$c_i = 3y_{i+1} - 3y_i - 2D_i - D_{i+1}$$

$$c_i = 3(y_{i+1} - y_i) - D_{i+1} - 2D_i \quad (Ec.1.6)$$

Hemos obtenido el valor de $c_i$ . De forma similar (multiplicando Ec.1.3 por 2 y restándole Ec.1.5) obtendremos el valor de $d_i$:

$$d_i = 2(y_i - y_{i+1}) + D_i + D_{i+1} \quad (Ec.1.7)$$

De esta forma; hemos obtenido los valores de las incógnitas a, b, c, d para cada intervalo respecto de y, D. Las expresiones son:

$$a_i = y_i \quad (Ec.1.2)$$

$$b_i = D_i \quad (Ec.1.4)$$

$$c_i = 3(y_{i+1} - y_i) - D_{i+1} - 2D_i \quad (Ec.1.6)$$

$$d_i = 2(y_i - y_{i+1}) + D_i + D_{i+1} \quad (Ec.1.7)$$

Calculemos el valor de la segunda derivada en el punto i:

$$y_i''(u) = 2c_i + 6d_i u$$

Como habíamos dicho que, por las condiciones de spline cúbica natural, se tiene que cumplir que $Y_0''(0) = Y_{m-1}''(1)$, obtenemos:

$$2c_i = 2c_{i-1} + 6d_{i-1}$$

$$c_i = c_{i-1} + 3d_{i-1}$$

Sustituyendo en la ecuación obtenida los valores de $c_i$ y $d_i$ (Ec. 1.6 y 1.7); sumando ambas y simplificando, obtenemos:

$$\begin{array}{r} c_i = 3(y_{i+1} - y_i) - D_{i+1} - 2D_i \\ 3d_i = 6(y_i - y_{i+1}) + 3D_i + 3D_{i+1} \\ \hline \end{array}$$

$$c_i + 3d_i = 3(y_{i+1} - y_i) + 6(y_i - y_{i+1}) + D_i + 2D_{i+1}$$

$$c_i + 3d_i = -3y_{i+1} + 3y_i + D_i + 2D_{i+1}$$

$$c_i + 3d_i = 3(y_i - y_{i+1}) + D_i + 2D_{i+1}$$

$$\underbrace{c_{i-1} + 3d_{i-1}}_{c_i} = 3(y_{i-1} - y_i) + D_{i-1} + 2D_i$$

Usamos de nuevo la ecuación 1.6, cambiando el valor de ci por esa expresión.

$$3(y_{i+1} - y_i) - 2D_i - D_{i+1} = 3(y_{i-1} - y_i) + D_{i-1} + 2D_i$$

$$-D_{i-1} - 4D_i - D_{i+1} = 3(y_{i-1} - y_i) - 3(y_{i+1} - y_i)$$

$$-D_{i-1} - 4D_i - D_{i+1} = 3y_{i-1} - 3y_i - 3y_{i+1} + 3y_i$$

Simplificando y sacando factor común:

$$D_{i-1} + 4D_i + D_{i+1} = 3y_{i+1} - 3y_{i-1}$$

$$D_{i-1} + 4D_i + D_{i+1} = 3(y_{i+1} - y_{i-1}) \quad (Ec.1.8)$$

Por una de las condiciones de spline natural, sabemos que $Y_0''(0) = 2c_0 = 0$, por lo que, sustituyendo en la Ec.1.6 tenemos que:

$$2D_0 + D_1 = 3(y_1 - y_0) \quad (Ec.1.9)$$

y como la condición también dice que

$$Y_{m-1}''(1) = 2c_{m-1} + 6d_{m-1} = 0$$

Obtenemos (operando en Ec.1.6 y Ec.1.7; multiplicando la primera por 2 y la segunda por 6):

$$D_{m-1} + 2D_m = 3(y_m - y_{m-1}) \quad (Ec.1.10)$$

Las ecuaciones (1.8), (1.9), (1.10), que corresponden a las primeras derivadas $D_i$, se pueden expresar en forma matricial de la siguiente manera:

$$\begin{bmatrix} 2 & 1 & & & & & \\ 1 & 4 & 1 & & & & \\ & 1 & 4 & 1 & & & \\ & & 1 & \dots & & & \\ & & & 1 & 4 & 1 & \\ & & & & 1 & 4 & 1 \\ & & & & & 1 & 2 \end{bmatrix} \cdot \begin{bmatrix} D_0 \\ D_1 \\ D_2 \\ \dots \\ D_{m-2} \\ D_{m-1} \\ D_m \end{bmatrix} = \begin{bmatrix} 3(y_1 - y_0) \\ 3(y_2 - y_1) \\ 3(y_3 - y_2) \\ \dots \\ 3(y_{m-2} - y_{m-3}) \\ 3(y_{m-1} - y_{m-2}) \\ 3(y_m - y_{m-1}) \end{bmatrix}$$

## 10.1. Optimización del método de resolución:

La matriz mxm anterior es una matriz tridiagonal; los elementos que no están indicados implícitamente son ceros. Este tipo de matrices son muy cómodas para calcular el valor de la expresión por medio de un computador. Sin embargo, puede optimizarse todavía más si logramos hacer una de las 3 líneas diagonales que hemos obtenido también cero.

El «truco» consiste en restar a cada fila la inmediatamente superior y multiplicarla luego por $\omega_i$. De esta forma, hemos transformado el sistema anterior, obteniendo:

$$\begin{bmatrix} 1 & \omega_0 & & & & & \\ & 1 & \omega_1 & & & & \\ & & 1 & & & & \\ & & & \dots & & & \\ & & & & 1 & \omega_{m-2} & \\ & & & & & 1 & \omega_{m-1} \\ & & & & & & 1 \end{bmatrix} \cdot \begin{bmatrix} D_0 \\ D_1 \\ D_2 \\ \dots \\ D_{m-2} \\ D_{m-1} \\ D_m \end{bmatrix} = \begin{bmatrix} \delta_0 \\ \delta_1 \\ \delta_2 \\ \dots \\ \delta_{m-2} \\ \delta_{m-1} \\ \delta_m \end{bmatrix}$$

así, la matriz que contiene las $\omega_i$ únicamente tiene dos diagonales que no son ceros (acelerando los cálculos un 33%). Los valores de $\omega$ y $\delta$ se calculan mediante las expresiones:

$$\omega_0 = \frac{1}{2}$$

$$\omega_i = \frac{1}{(4 - \omega_{i-1})} \qquad \forall i \in [1, m-1]$$

$$\omega_m = \frac{1}{(2 - \omega_{m-1})}$$

$$\delta_0 = 3(y_1 - y_0)\omega_0$$

$$\delta_i = \left[3(y_{i+1} - y_{i-1}) \delta_{i-1}\right]\omega_i \qquad \forall i \in [1, m-1]$$

$$\delta_m = \left[3(y_m - y_{m-1}) \delta_{m-1}\right]\omega_m$$

Los casos particulares de cálculo son el primer y último punto de control. El resto, serán calculados de forma general por la expresión de término i.

Resolviendo la ecuación matricial «a mano», nos damos cuenta que la derivada primera en el último punto de control se puede resolver directamente calculando el valor de $\delta$. Será lo que se utilice en la implementación del algoritmo; primero resolveremos el término m, y volviendo atrás hacia el primero aplicaremos la ecuación general de término i. Es decir, tendremos que:

$$D_m = \delta_m$$

$$D_i = \delta_i - \omega_i D_{i+1} \qquad \forall i \in [m-1, 1]$$

## 10.2. Algunas notas sobre el código fuente:

El algoritmo de interpolación de puntos por spline cúbica natural está en la función SplineCubicaNatural del módulo splines.c.

La implementación resulta de transformar directamente las ecuaciones vistas anteriormente a código. Bastará con seguir las últimas optimizaciones para comprender el código fuente de dicha función. El resto del programa inicializa el modo de vídeo VGA y mira el usuario ha pulsado alguna tecla para mover los puntos de control de la spline (se cambia de un punto a otro con las teclas + y -. Para mover los puntos de control se utilizan las teclas l,j,k,i,h,f,g,t -ver fuentes de main-).

Se utilizó el entorno de desarrollo Borland C 3.1, y las librerías gráfigas BGI. No os olvidéis copiar el fichero EGAVGA.BGI en el directorio donde se encuentre el ejecutable. Si tecleáis el código fuente, antes de compilar el proyecto, debéis activar la casilla de «Graphics Library» en el menú options/linker/libraries (ver figura 12)

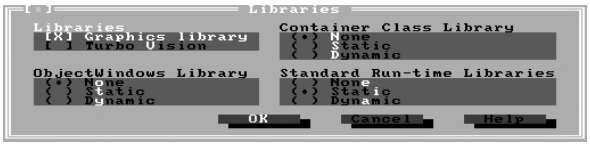



**Figura 12**. Ventana options/linker/libraries de BC3.1

```
/*****************************************************************
 *  GRAFICOS.C                          Ultima version: 19/Mar/00
 *****************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <graphics.h>

int modo_grafico (void)
{
   /* Pedimos autodeteccion del driver ... */
   int gdriver = DETECT, gmode, errorcode;
   /* Inicializa el modo grafico */
   initgraph(&gdriver, &gmode, "");
   /* Tomamos el resultado de la inicializacion */
   errorcode = graphresult();

   if (errorcode != grOk) {  /* Ocurrio un error! */
      printf("\n-Error inicializando modo grafico!\n");
      printf("Fallo en: %s\n", grapherrormsg(errorcode));
      return(1);   /* Devolvemos un error */
   }
   return(0);
}

void modo_texto(void)
{
   closegraph();
}

/*****************************************************************
 *  GRAFICOS.H                          Ultima version: 19/Mar/00
 *****************************************************************/
int modo_grafico (void);
void modo_texto(void);


/*****************************************************************
 *  SPLINES.C                          Ultima version: 24/Mar/00
 *****************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <graphics.h>
#include "graficos.h"

#define MAX_PTOS 10   /* Numero maximo de puntos de control */
#define NUM_SEG 20    /* Cuanto mayor sea, mejor se dibuja  */

void dibujaPuntos (int n, double x[], double y[])
{
   int i;
   for (i=0; i<n; i++) circle ((int) x[i], (int) y[i], 2);
}
```

```
void SplineCubicaNatural (int n, double x[], double y[])
{
   int i, j, m, xp, yp;
   double ax[MAX_PTOS], bx[MAX_PTOS], cx[MAX_PTOS], dx[MAX_PTOS];
   double ay[MAX_PTOS], by[MAX_PTOS], cy[MAX_PTOS], dy[MAX_PTOS];
   double der[MAX_PTOS], gam[MAX_PTOS], ome[MAX_PTOS];
   double t, dt;

   m = n-1;       /* m es el numero de intervalos que tendremos */

   /* Calculamos el valor de gamma (sera el mismo en X y en Y) */
   gam[0] = .5;
   for (i=1; i<m; i++) gam[i] = 1./(4.-gam[i-1]);
   gam[m] = 1./(2.-gam[m-1]);

   /* Calculamos el valor de omega para abcisas */
   ome[0] = 3.*(x[1]-x[0])*gam[0];
   for (i=1; i<m; i++) ome[i] = (3.*(x[i+1]-x[i-1])-ome[i-1])*gam[i];
   ome[m] = (3.*(x[m]-x[m-1])-ome[m-1])*gam[m];

   /* Valor de la primera derivada en los puntos (eje X) */
   der[m]=ome[m];
   for (i=m-1; i>=0; i=i-1) der[i] = ome[i]-gam[i]*der[i+1];

   /* Sustituimos los valores de gamma, omega y la primera derivada
      para calcular los coeficientes a, b, c y d */
   for (i=0; i<m; i++) {
      ax[i] = x[i];
      bx[i] = der[i];
      cx[i] = 3.*(x[i+1]-x[i])-2.*der[i]-der[i+1];
      dx[i] = 2.*(x[i]-x[i+1])+der[i]+der[i+1];
   }

   /* Calculamos omega para el eje de ordenadas */
   ome[0] = 3.*(y[1]-y[0])*gam[0];
   for (i=1; i<m; i++) ome[i] = (3.*(y[i+1]-y[i-1])-ome[i-1])*gam[i];
   ome[m] = (3.*(y[m]-y[m-1])-ome[m-1])*gam[m];

   /* Hallamos el valor de la primera derivada... */
   der[m]=ome[m];
   for (i=m-1; i>=0; i=i-1) der[i] = ome[i]-gam[i]*der[i+1];

   /* Valor de los coeficientes a, b, c y d en el eje Y */
   for (i=0; i<m; i++) {
      ay[i] = y[i];
      by[i] = der[i];
      cy[i] = 3.*(y[i+1]-y[i])-2.*der[i]-der[i+1];
      dy[i] = 2.*(y[i]-y[i+1])+der[i]+der[i+1];
   }

   /* En esta parte, se dibujara la curva por segmentos de lineas
      rectas; si NUM_SEG es un valor alto, la grafica se dibujara
      con mayor precision. */

   dt = 1./(double) NUM_SEG;
   moveto((int) x[0], (int) y[0]);
   for (i=0; i<m; i++) {
      for (j=1, t=dt; j<NUM_SEG; j++, t+=dt) {
         xp = (int) (ax[i]+bx[i]*t+cx[i]*t*t+dx[i]*t*t*t);
         yp = (int) (ay[i]+by[i]*t+cy[i]*t*t+dy[i]*t*t*t);
         lineto (xp, yp);
      }
   }
}
```

```
int main (void)
{
   double cx[] = {80.5, 120.4, 150.3, 210.0, 352.8, 450.23, 583.33, 564.0};
   double cy[] = {20.0, 148.3, 189.32, 275.0, 77.2, 182.81, 315.72, 82.23};
   int npuntos = 8, actual =0, tecla;

   if (modo_grafico() == 0) {  /* Inicializacion correcta. */
      while (tecla != 's') {
         switch (tecla) {
            case '+' : actual == npuntos-1 ? actual = 0 : actual++; break;
            case '-' : actual == 0 ? actual = npuntos-1 : actual--; break;
            case 'h' : cx[actual]+=2; break;  /* Derecha precision */
            case 'f' : cx[actual]-=2; break;  /* Izquierda precision */
            case 'g' : cy[actual]+=2; break;  /* Abajo precision */
            case 't' : cy[actual]-=2; break;  /* Arriba precision */
            case 'l' : cx[actual]+=5; break;  /* Derecha Rapida */
            case 'j' : cx[actual]-=5; break;  /* Izquierda Rapida */
            case 'k' : cy[actual]+=5; break;  /* Abajo Rapida */
            case 'i' : cy[actual]-=5;         /* Arriba Rapida */
            default : ;
         }
         dibujaPuntos (npuntos, cx, cy);
         floodfill( (int) cx[actual], (int) cy[actual], EGA_WHITE);
         SplineCubicaNatural(npuntos, cx, cy);
         tecla = getch();
        cleardevice();     /* Limpiamos la pantalla */
      }
      modo_texto();
   }
   return(0);
}
```

## 11. Bibliografía


- Watt A., "*Avanced Animation and Rendering Techniques - Theory and Practice*", Ed. Addison - Wesley, 1992 (ISBN 0-201-54412-1)
- Farin G., "*Curves and Surfaces for CAGD. A practical guide (Third Edition)*", Ed. Academic Press, 1993 (ISBN 0-12-249052-5)
- AAVV, "Numerical Reciples in C: The art of Scientific Computing", Ed. Cambridge University Press, 1992 (ISBN 0-521-43108-5)
- Olfe, D.B. "*Computer Graphics for Design. From Algorithms to Auto-CAD*", Ed. Prentice Hall, 1995 (ISBN 0-13-159583-0)
- Dony, R, "*Eliminación de partes ocultas. Aproximación de curvas por el método de Beizer y las B-Splines*", Ed. Masson, 1988.
- Burden, R.L. "*Análisis Numérico*", Ed. International Thomson, 1998.