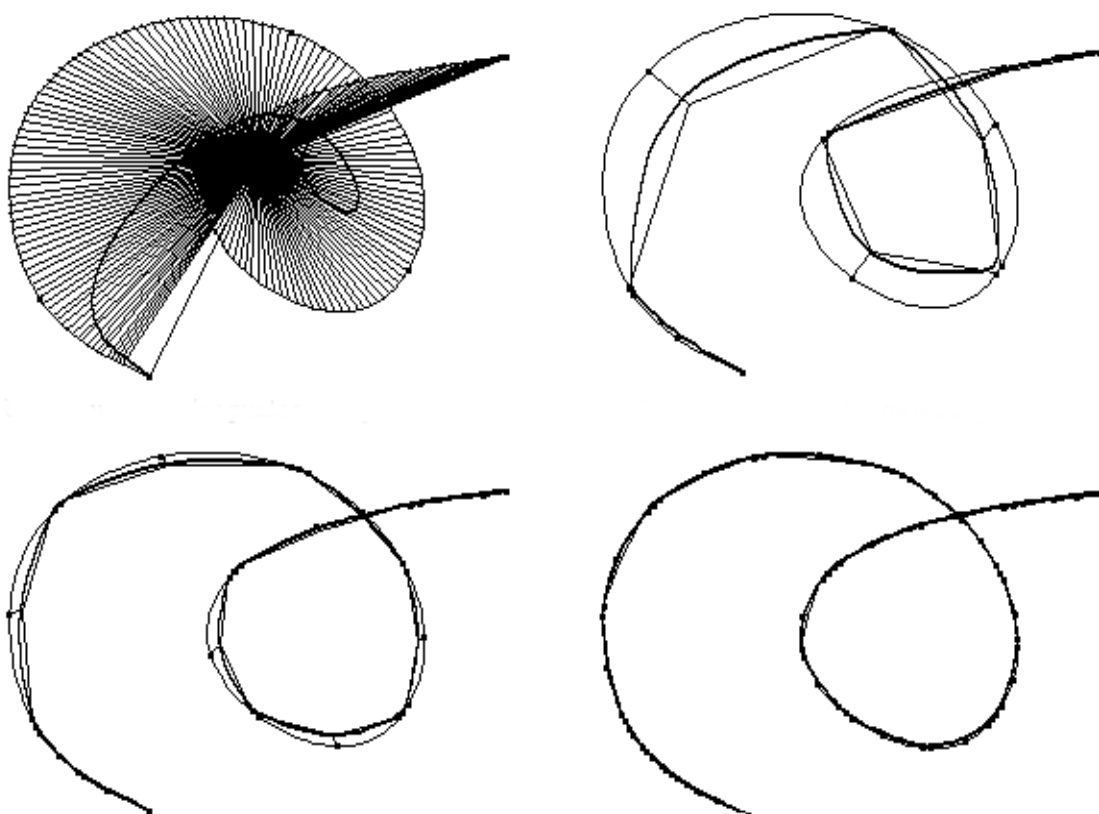


# 4

## Aproximación e Interpolación de Curvas



## 4.1 Motivaciones

Hasta ahora hemos visto cómo elaborar gráficos a partir de los comandos gráficos más comunes en un lenguaje de programación (el punto y el segmento de línea recta). Estas primitivas son suficientes en el sentido de que cualquier otra construcción geométrica puede ser convenientemente aproximada con puntos y segmentos, hasta el punto de ser indistinguible para una resolución gráfica dada. Muy pronto se comprendió en la Computación Gráfica que las técnicas de representación de objetos están muy limitadas en las posibilidades geométricas de las entidades gráficas que se pueden modelar y graficar.

Esencialmente, los modelos basados en la representación de *puntos* destacados, los cuales son unidos entre sí por aristas, tienen la ventaja de que todas las operaciones involucradas en la graficación, excepto el clipping, se realizan por medio de la transformación de dichos puntos. Los mismos son posteriormente unidos entre sí por medio de algoritmos eficientes de discretización de segmentos de recta. Abandonar esta filosofía de trabajo implica un aumento de varios órdenes de magnitud en el costo de procesamiento.

Podemos describir un objeto determinado –una cuádrica, por ejemplo– por medio de un sistema de ecuaciones. Esto implicaría que debemos muestrear dicho sistema de ecuaciones una cantidad suficiente de veces para obtener un resultado gráfico adecuado, procesando luego cada muestra por la misma tubería de transformaciones. Por otra parte, si dicho objeto ocupa una porción pequeña o muy grande de la pantalla, este esquema de representación permite relacionar la cantidad necesaria de muestras para producir un resultado adecuado con el tamaño o porción que el objeto ocupa en la pantalla.

Sin embargo, mantenernos dentro del modelo de puntos y vértices puede tener grandes desventajas a la hora de representar objetos con geometrías más variadas. Por ejemplo, para representar objetos cuya descripción ecuacional se desconoce, se puede utilizar un conjunto determinado de puntos que conformen un “cuadriculado” del mismo. Dichos datos pueden provenir de un *scanner*, de muestras, o de simulaciones procedimentales. Esto presupone de por sí un compromiso complejidad-calidad, es decir, representaciones más fieles implican un mayor costo en tiempo y memoria.

En esta representación “poligonal”, podemos elegir un buen compromiso para garantizar una apariencia adecuada del objeto graficado en circunstancias normales. Pero al representarse el modelo a muy grandes escalas, su naturaleza poligonal se volvería evidente, pudiendo inclusive “desaparecer” el objeto si en su proyección en la pantalla

no aparece ningún vértice o arista. Y por el contrario, si la escala es muy pequeña, el objeto ocupa una parte reducida de la pantalla pero para ser graficado requiere procesar la misma cantidad de puntos y aristas.

Pero no es ésta la desventaja más importante de una representación de objetos por medio de polígonos. A comienzos de la década del 60 ya era posible comandar por computadora las maquinarias necesarias para producir piezas en madera, plástico o acero, en lo que se denomina Computer Aided Manufacturing (CAM) o manufactura asistida por computadora. Dichas piezas pueden ser luego utilizadas en el desarrollo industrial, como por ejemplo la matricería o estampado de carrocerías de automóviles. Para ello es necesario representar la forma de dichas piezas de modo que la ejecución del programa produzca el resultado adecuado. Con la representación poligonal podemos llegar eventualmente a obtener una estructura de datos adecuada, pero, como puede suceder, si se requieren cambios o modificaciones de último momento, entonces es necesario acometer la trabajosa tarea de editar punto por punto en la base de datos que representa los polígonos.

Los sistemas comerciales se hicieron sensibles a estas dificultades. Los utilitarios de dibujo y diseño gráfico comenzaron a incluir la posibilidad de trabajar con círculos, arcos y cónicas, mientras que sistemas más avanzados de dibujo técnico proveen procedimientos para aproximar y ajustar curvas planas, y para combinar perfiles planos con direcciones de extrusión para elaborar modelos de objetos tridimensionales más complejos. Muy pronto se llegó a la conclusión que esta forma de trabajar es esencialmente incorrecta, trabajosa y sujeta a errores. Por dicha razón se comenzaron a estudiar los fundamentos de los métodos que estudiaremos en este Capítulo, y que constituyen esencialmente lo que se conoce como Computer Aided Design (CAD), Computer Aided Geometric Design (CAGD) o diseño (geométrico) asistido por computadora.

El pleno poder para modelar objetos tridimensionales complejos a partir de descripciones geométricas precisas solo se obtuvo a partir del desarrollo en el CAD de las técnicas de aproximación de curvas y superficies paramétricas a partir de *puntos de control*. En la actualidad el CAD se independizó del CAM y constituye una disciplina aparte, con sus propias motivaciones, e intereses que van más allá de la manufactura industrial, aunque esta última siempre requiere una fase previa de diseño.

Los métodos de aproximación e interpolación de curvas que veremos a continuación, entonces, están pensados en función de facilitar la tarea de diseño, la cual es muchas veces una tarea iterativa de prueba y error. En algunas aplicaciones el diseño tiene un uso *analítico*, es decir, a partir de un determinado modelo real se obtiene una representación característica manipulable y más económica. Otro tipo de uso es el  *sintético*, en el cual se parte de un conjunto de especificaciones geométricas o analíticas

que van determinando la forma final del objeto. Por lo tanto la creación y modificación de modelos geométricos debe ser sencilla y predecible. Pero al mismo tiempo debe ser eficiente en su cómputo y por lo tanto compatible con el modelo de procesamiento visto en los Capítulos anteriores.

El origen de estos sistemas, que permiten modelar objetos con total libertad, puede rastrearse a los alrededores de la década del 60. Los desarrollos más importantes se iniciaron en la industria automotriz francesa, donde Pierre de Casteljau en Citroën, y Pierre Bézier en Renault elaboraron el fundamento teórico de los primeros sistemas de aproximación de curvas que se sobreponían exitosamente a los problemas técnicos y geométricos de los métodos matemáticos de interpolación de funciones basados en los polinomios de Lagrange. Esencialmente ambos trabajos coinciden, aunque fueron independientemente desarrollados. Como Bézier fue el único que publicó sus resultados, se llevó todo el crédito y la fama.

Muy pronto se produjo una convergencia con los métodos de la teoría de aproximación de funciones, especialmente con las aproximaciones polinomiales a trozos o *splines*. Ya en 1975 comenzaron a existir foros de discusión y Conferencias especializadas, y en la década del 80 el CAD se independizó como disciplina, con sus propias metodologías, publicaciones y grupos de interés. Pese al gran desarrollo ocurrido desde entonces, las curvas de Bézier-de Casteljau continúan siendo la base, tanto porque se basan en las bases funcionales numérica y analíticamente más estables conocidas, como por el hecho demoledor de que todo otro método puede ponerse en términos de casos particulares de estas curvas (salvo casos excepcionales, que normalmente no son aceptados como “buenos” métodos).

Los métodos paramétricos de aproximación de curvas logran flexibilidad de representación y eficiencia en el cómputo, al utilizar una formulación basada en *puntos de control*. Dichos puntos de control son localizaciones en el plano o el espacio que gobiernan la forma final de la curva o superficie. La implementación de un método consiste en encontrar funciones polinomiales de grado bajo que aproximan o interpolan a dichos puntos de control con características analíticas o geométricas específicas. Dichas funciones pueden ser luego evaluadas para obtener una cantidad suficiente de puntos, los cuales son procesados como en el caso poligonal.

#### 4.1.1 Especificaciones y requisitos

Las curvas que buscamos deben tener ciertas propiedades geométricas que es necesario especificar con claridad, del mismo modo que lo hicimos en el Capítulo 2 con los

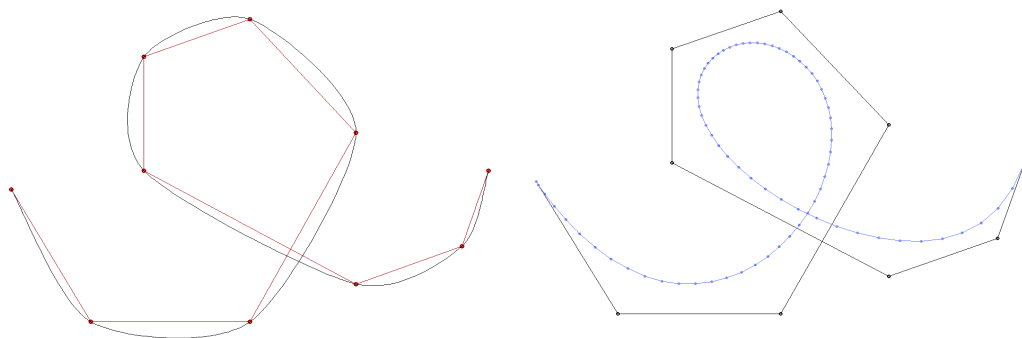


Figura 4.1 Interpolación vs. aproximación.

métodos de discretización. Como ya mencionáramos, la descripción de estas entidades se realiza por medio de puntos de control. La secuencia de puntos de control, unidos por segmentos de rectas, se denomina *grafo de control*. Podemos mencionar, entre otras, las siguientes especificaciones en el caso de curvas, aunque no es difícil extrapolar los requisitos para superficies.

**Interpolación o aproximación:** Dado un conjunto de puntos de control, es posible que la curva resultante pase por todos ellos (ver Figura 4.1(a)). En dicho caso el método que encuentra la curva a partir de los puntos de control se denomina *método de interpolación*. Puede suceder, en cambio, que la curva normalmente no pase por los puntos, excepto tal vez por los puntos extremos (ver Figura 4.1(b)). Entonces estamos frente a un *método de aproximación*.

**Invariancia afín:** Esta propiedad es indispensable si buscamos mantenernos dentro del “modelo de tubería”, con su eficiencia computacional asociada. Básicamente, se espera que sea indistinto aplicar una transformación afín al grafo de control y luego aproximar (o interpolar) una curva sobre el mismo, o aproximar una curva sobre el grafo original y luego transformarla. Dicho de otra manera, las operaciones de aproximar (o interpolar) y de transformar *conmutan*. Esta propiedad permite garantizar no solo que la apariencia de una curva aproximante no varía si se la traslada, rota, etc., sino además que es correcto encontrar un conjunto de puntos sobre la curva y aplicarle el mismo procesamiento que a las demás entidades de la escena.

**Entidades multivaluadas:** Si las curvas aproximantes son de la forma  $y = f(x)$  entonces están limitadas a no poder representar entidades multivaluadas (ver Figura 4.2). Esta limitación es intolerable, dado que la mayor parte de las curvas

o superficies tienen características geométricas de esta naturaleza. Además, la representación debe ser necesariamente independiente del sistema de coordenadas.

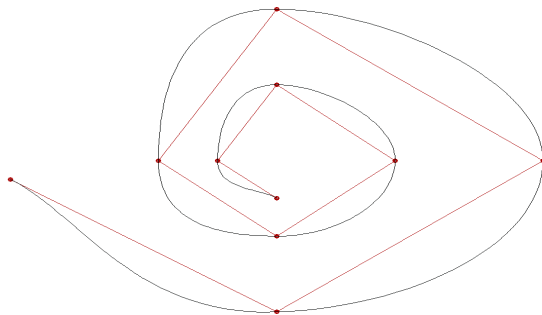


Figura 4.2 Entidades multivaluadas.

**Disminución de variaciones:** El grafo de control “sugiere” la forma general de la curva, pero localmente ésta podría tener variaciones u oscilaciones (ver Figura 4.3(a)). Esto es indeseable en general. Se desea que toda variación u oscilación esté ya presente en el grafo de control, y que, por el contrario, las mismas sean suavizadas o atenuadas por la curva resultante (ver Figura 4.3(b)).

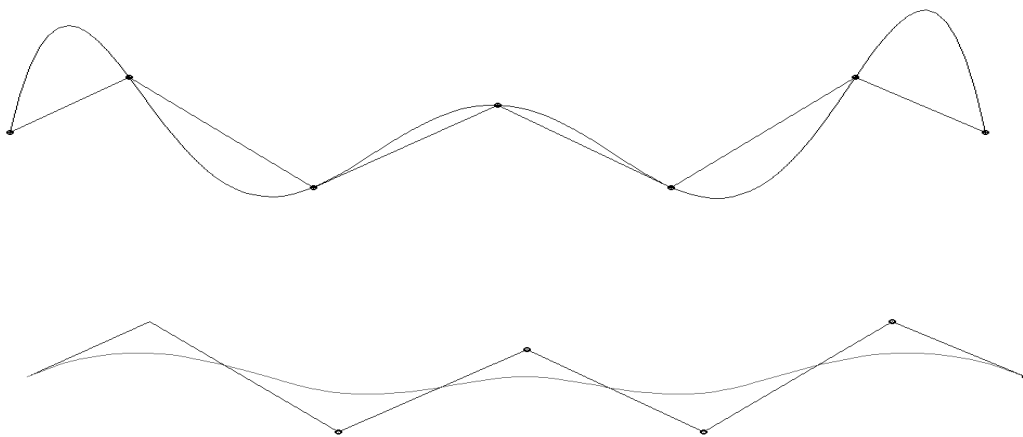


Figura 4.3 Amplificación o disminución de variaciones.

**Orden de continuidad:** Dada una curva, sea  $k$  el máximo orden de derivación de la curva tal que la función resultante sea continua. En dicho caso diremos que la

curva es  $C^k$  continua, o que tiene un orden de continuidad  $k$ . Es indispensable poder conocer matemáticamente el orden de continuidad resultante de una curva aproximante.

**Control local o global:** En algunos métodos la curva aproximante experimenta un cambio localizado si se modifica un único punto de control, mientras que en otros, toda la curva es modificada. Los primeros métodos tienen, entonces, la propiedad de *control local* y los segundos de *control global*. Ninguna de las dos propiedades es por sí misma mejor que la otra, sino que depende del tipo de aplicación.

#### 4.1.2 La representación paramétrica

La clave para poder satisfacer los requisitos planteados más arriba consiste en suponer que la curva resultante de aproximar o interpolar el grafo de control es un conjunto de funciones paramétricas, donde el parámetro  $u$  varía dentro de un intervalo cerrado (usualmente  $u \in [0, 1]$ ). Por ejemplo, para una curva en el espacio podemos encontrar tres funciones  $f_x(u)$ ,  $f_y(u)$ , y  $f_z(u)$  que gobiernen la posición de un punto sobre la curva para un valor dado de  $u$ .

$$C(u) = \begin{cases} x(u) = f_x(u) \\ y(u) = f_y(u) \\ z(u) = f_z(u) \end{cases}$$

En principio no hay nada que limite la forma que pueden asumir estas funciones. Sin embargo, desde el punto de vista de la Computación Gráfica, es importante que la evaluación de las mismas sea estable, económica y robusta. Otro de los puntos importantes, entonces, es que dichas funciones normalmente son polinomios de grado relativamente bajo.

$$C(u) = \begin{cases} f_x(u) = c_x^n u^n + c_x^{n-1} u^{n-1} + \dots + c_x^1 u + c_x^0 \\ f_y(u) = c_y^n u^n + c_y^{n-1} u^{n-1} + \dots + c_y^1 u + c_y^0 \\ f_z(u) = c_z^n u^n + c_z^{n-1} u^{n-1} + \dots + c_z^1 u + c_z^0 \end{cases}$$

El grado y los coeficientes de dichos polinomios se extraen de la ubicación de los puntos de control. Por lo tanto, los métodos de interpolación y aproximación tienen como objetivo encontrar las funciones  $f_x(u)$ ,  $f_y(u)$ , y  $f_z(u)$  a partir de los datos geométricos de dichos puntos.

## 4.2 Interpolación de Curvas

Los métodos de interpolación de curvas fueron por primera vez estudiados en el análisis matemático y se utilizan en la actualidad en el cálculo numérico. Su aplicación en la Computación Gráfica, sin embargo, es muy limitada dado que en los últimos 20 años se desarrollaron técnicas más específicas para dicho ámbito. De todas maneras, las ideas básicas y terminología utilizadas en los modelos de aproximación de curvas se apoyan en los modelos de interpolación. Por dicha razón es que dedicaremos una Sección para describir brevemente los métodos de Lagrange y de Hermite para interpolar puntos.

### 4.2.1 Interpolación de Curvas de Lagrange

La determinación del interpolante de Lagrange es uno de los métodos polinomiales más directos. Sea una secuencia de  $n + 1$  números reales  $x_0 \leq x_1 \leq \dots \leq x_n$ , denominados *nudos* y un segundo conjunto de  $n + 1$  números  $y_0, y_1, \dots, y_n$ , denominados *valores*, de modo que  $p_i = (x_i, y_i)$  es un punto de control. Una secuencia de puntos de control es llamada *grafo de control*, y debe ser tal que la secuencia de nudos (valores de la coordenada  $x$  sea creciente. Entonces buscamos una función polinomial  $f(x)$  de grado  $n$  tal que satisfaga el problema de interpolación del grafo de control

$$\forall i \in [0..n], f(x_i) = y_i.$$

El interpolante de Lagrange se define como

$$f(x) = \sum_{i=0}^n L_i(x) y_i,$$

donde

$$L_i = \frac{(x - x_0), (x - x_1), \dots, (x - x_{i-1}), (x - x_{i+1}), \dots, (x - x_n)}{(x_i - x_0), (x_i - x_1), \dots, (x_i - x_{i-1}), (x_i - x_{i+1}), \dots, (x_i - x_n)}.$$

Por ejemplo, si  $n = 1$ , tenemos dos nudos  $x_0$  y  $x_1$ , y sus valores asociados, encontrando que

$$f(x) = \frac{x - x_1}{x_0 - x_1} y_0 - \frac{x - x_0}{x_1 - x_0} y_1,$$

expresión que es idéntica a

$$f(x) = y_0 + (y_1 - y_0) \frac{x - x_0}{x_1 - x_0}.$$

```

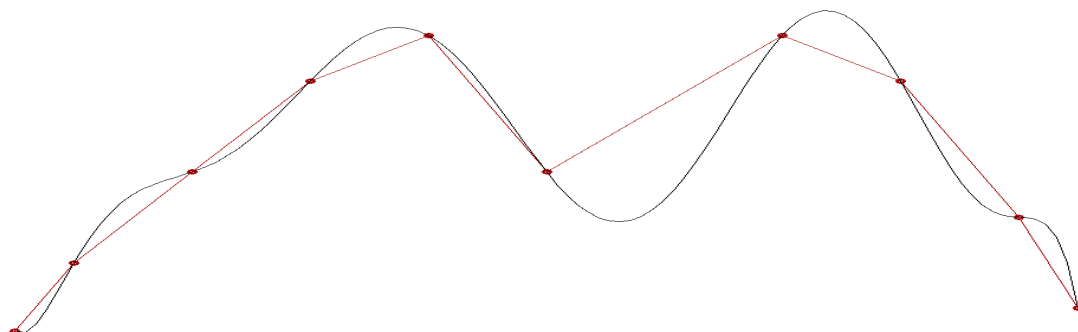
const pts_ ctrol= (cantidad de puntos de control);

type  punto = record x,y,z,w:real
                end;
grafo_contr =  array [1..n] of punto;
...
procedure lagrange(var g:grafo_contr);
var x,y,incrx,num,den:real;
    i,j,k:integer;
    p:punto;
begin;
    incrx:=(g[pts_ctrol].x-g[1].x)/100;
    for i:=0 to 100 do begin
        x:=g[1].x+i*incrx;          y:=0;
        p.z:=1;                      p.w:=1;
        for j:=1 to pts_ctrol do begin
            num:=1;                  den:=1;
            for k:=1 to pts_ctrol do
                if (k<>j) then begin
                    num:=num*(x-g[k].x);
                    den:=den*(g[j].x-g[k].x);
                end;
            y:= y+g[j].y*num/den;
        end;
        p.x:=x;                      p.y:=y;
        graf_linea(p);
    end;
end;

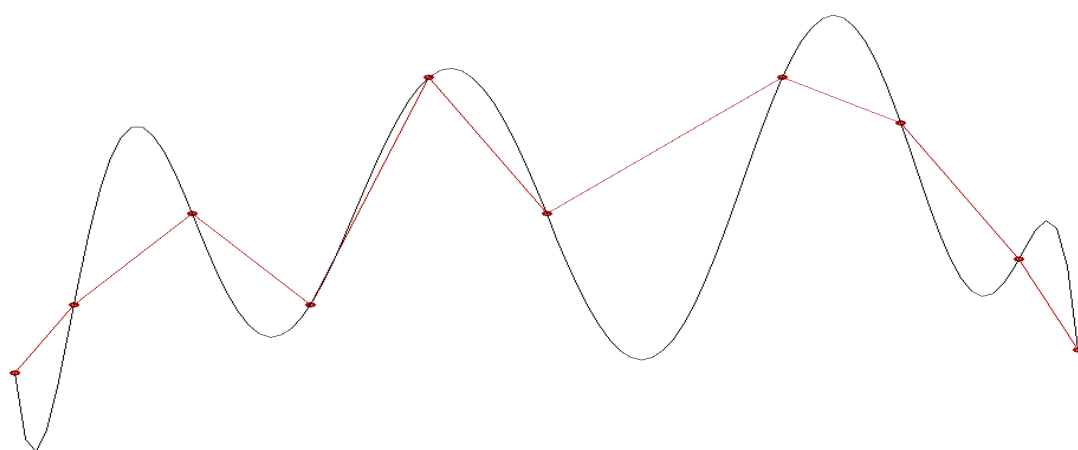
```

**Figura 4.4** Procedimiento que implementa la interpolación de Lagrange.

Es decir, para un par de puntos de control, el interpolante de Lagrange coincide con el segmento de recta que une a dichos puntos. Es posible demostrar que el interpolante de Lagrange siempre existe y es único, para un  $n$  finito, y que es el único polinomio de grado  $n$  que satisface el problema de interpolar  $f(x_i) = y_i$  para todos los puntos de control. El procedimiento de la Figura 4.4 muestra una implementación del método, el cual es utilizado en la Figura 4.5 con un grafo de control dado.



**Figura 4.5** Interpolación de una secuencia de puntos de control por medio del interpolante de Lagrange.



**Figura 4.6** Efecto de mover un punto de control al grafo en el método de Lagrange.

Es importante notar que si se modifica el valor de un nudo cualquiera, se modifica la expresión completa del polinomio. Es decir, este método tiene control global. Al mismo tiempo, dado que el método no utiliza una representación paramétrica, entonces no permite interpolar curvas multivaluadas.

La desventaja más considerable del método de Lagrange es que si aumentamos la cantidad de puntos (tal vez para mejorar la calidad de la entrada al procedimiento), entonces aumenta el grado de los polinomios. Esto en sí no sería más que un inconveniente computacional, pero en la práctica, a partir de  $n = 5$ , comienzan a ocurrir oscilaciones indeseadas en el interpolante entre los puntos de control, las cuales se vuelven de mayor amplitud cuando  $n$  aumenta o cuando se altera la posición de un punto de control (ver Figuras 4.5 y 4.6). Es decir, al modificar o agregar un punto de control para mejorar la interpolación, en realidad lo que ocurre es que aumentan las oscilaciones.

#### 4.2.2 Interpolación de Curvas de Hermite

Una solución posible para el problema de las oscilaciones es la propuesta por Hermite. En Hermite, la interpolación se realiza entre pares sucesivos de puntos de control, de modo que no solo los valores son interpolados en los nudos, sino también un cierto número de derivadas. Por ejemplo, para interpolar una curva plana  $C_i(u)$  que pase por  $p_i = (x_i, y_i)$  y por  $p_{i+1} = (x_{i+1}, y_{i+1})$ , con una determinada derivada  $\dot{p}_i$  en el primer punto y con una derivada  $\dot{p}_{i+1}$  en el segundo, necesitamos expresar a las funciones  $f_x(u)$  y  $f_y(u)$  como polinomios cúbicos en  $u$ . Esto es así dado que tenemos, para cada uno de ellos, cuatro “datos” (las posiciones y derivadas al comienzo y al final de la curva), y por lo tanto podemos encontrar cuatro “incógnitas” (los coeficientes del polinomio).

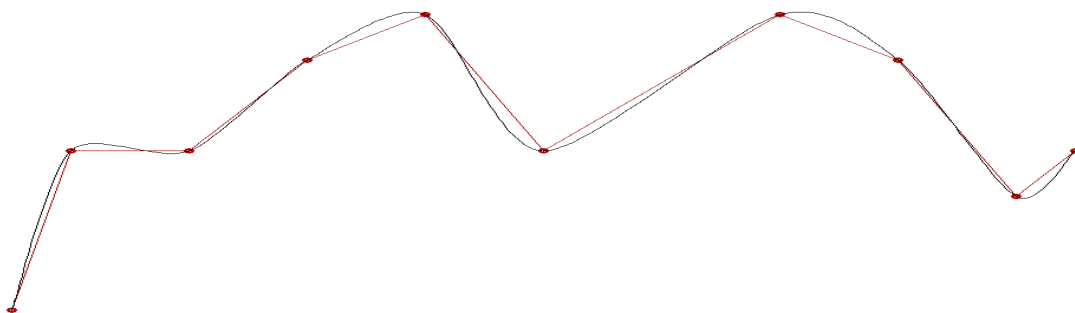


Figura 4.7 Interpolación de Hermite de un grafo de control.

Es decir, a partir de los datos de posición  $(x_i, y_i)$ ,  $(x_{i+1}, y_{i+1})$ , y de derivada  $(\dot{x}_i, \dot{y}_i)$ ,  $(\dot{x}_{i+1}, \dot{y}_{i+1})$ , existen dos únicos polinomios  $f_x(u)$  y  $f_y(u)$  tales que el interpolante de Hermite

$$C(u) = \begin{cases} f_x(u) = c_x^3 u^3 + c_x^2 u^2 + c_x^1 u + c_x^0 \\ f_y(u) = c_y^3 u^3 + c_y^2 u^2 + c_y^1 u + c_y^0 \end{cases}$$

es un segmento de curva plana que pasa por los puntos con las derivadas establecidas.

```

procedure hermite(var p1,p2:punto; var d1,d2:deri);
var u,x,y,cy0,cy1,cy2,cy3,cx0,cx1,cx2,cx3:real;
    p:punto;
    i:integer;
begin
    p.z:=1;                                p.w:=1;
    cx3:=2*p1.x-2*p2.x+d1.x+d2.x;        cy3:=2*p1.y-2*p2.y+d1.y+d2.y;
    cx2:=-3*p1.x+3*p2.x-2*d1.x-d2.x;      cy2:=-3*p1.y+3*p2.y-2*d1.y-d2.y;
    cx1:=d1.x;                             cy1:=d1.y;
    cx0:=p1.x;                             cy0:=p1.y;
    p.x:=cx0;                              p.y:=cy0;
    graf_punto(p);
    for i:=1 to 30 do begin
        u:=i/30;
        p.x:=cx0+u*(cx1+u*(cx2+u*cx3)); p.y:=cy0+u*(cy1+u*(cy2+u*cy3));
        graf_linea(p);
    end;
end;
```

**Figura 4.8** Procedimiento que implementa la interpolación de Hermite.

Si bien los datos de posición surgen de una manera relativamente directa del problema de diseño, no sucede lo mismo con los datos de derivada. Si queremos encontrar el modelo de un objeto cualquiera, es posible estimar la posición de un punto dado y de la pendiente en el mismo, pero la magnitud de la derivada puede ser difícil de estimar. Por lo tanto, trabajar con estimadores de derivadas puede ser una tarea un tanto artesanal.

Para encontrar los coeficientes de los polinomios debemos primero asignar un dominio para el parámetro  $u$ . Normalmente se utiliza  $u \in [0, 1]$ , con lo cual por ejemplo  $f_x(0) = x_i$ ,  $f_x(1) = x_{i+1}$ , y también  $\dot{f}_x(0) = \dot{x}_i$ , etc., y también podemos expresar por ejemplo  $C_i(0) = p_i$  y  $\dot{C}_i(1) = \dot{p}_{i+1}$ . Además,  $C_i(1) = C_{i+1}(0)$ , etc. Esta asigna-

ción de dominio para el parámetro es arbitraria, pudiéndose encontrar una formulación equivalente para cualquier otra asignación no trivial.

Podemos ahora expresar a los polinomios de Hermite como producto escalar de un vector (fila) de funciones de  $u$  por un vector (columna) de coeficientes:

$$f_x(u) = c_x^3 u^3 + c_x^2 u^2 + c_x^1 u + c_x^0 = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} \begin{bmatrix} c_x^3 \\ c_x^2 \\ c_x^1 \\ c_x^0 \end{bmatrix}. \quad (4.1)$$

Si bien esta manipulación algebraica es relativamente trivial, sus implicaciones son importantes. Por ejemplo, nos permite ver a un polinomio como un elemento o “punto” en un espacio de funciones. En este caso la base del espacio funcional es

$$\begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix}$$

y el vector de coeficientes determina “las coordenadas” de  $f_x(u)$  dentro de dicho espacio. Esta base funcional es denominada la *base polinomial*, en este caso para funciones de orden cúbico, pero pueden existir otras bases. De hecho, en las Secciones siguientes utilizaremos otras familias de bases funcionales más adecuadas que la base polinomial para la interpolación y aproximación de curvas.

Es posible expresar a las derivadas de la curva de una manera similar:

$$\dot{f}_x(u) = 3c_x^3 u^2 + 2c_x^2 u + c_x^1 + 0 = \begin{bmatrix} 3u^2 & 2u & 1 & 0 \end{bmatrix} \begin{bmatrix} c_x^3 \\ c_x^2 \\ c_x^1 \\ c_x^0 \end{bmatrix}.$$

Podemos ahora expresar todas nuestras restricciones de una manera homogénea:

$$\begin{aligned} f_x(0) &= c_x^3 0^3 + c_x^2 0^2 + c_x^1 0 + c_x^0 = \begin{bmatrix} 0^3 & 0^2 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_x^3 \\ c_x^2 \\ c_x^1 \\ c_x^0 \end{bmatrix} \\ f_x(1) &= \begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} c_x^3 \\ c_x^2 \\ c_x^1 \\ c_x^0 \end{bmatrix} \\ \dot{f}_x(0) &= \begin{bmatrix} 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} c_x^3 \\ c_x^2 \\ c_x^1 \\ c_x^0 \end{bmatrix} \end{aligned}$$

$$\dot{f}_x(1) = \begin{bmatrix} 3 & 2 & 1 & 0 \end{bmatrix} \begin{bmatrix} c_x^3 \\ c_x^2 \\ c_x^1 \\ c_x^0 \end{bmatrix}.$$

Por fin, podemos representar todas las restricciones en un único sistema de ecuaciones:

$$\begin{bmatrix} f_x(0) \\ f_x(1) \\ \dot{f}_x(0) \\ \dot{f}_x(1) \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix} \begin{bmatrix} c_x^3 \\ c_x^2 \\ c_x^1 \\ c_x^0 \end{bmatrix}.$$

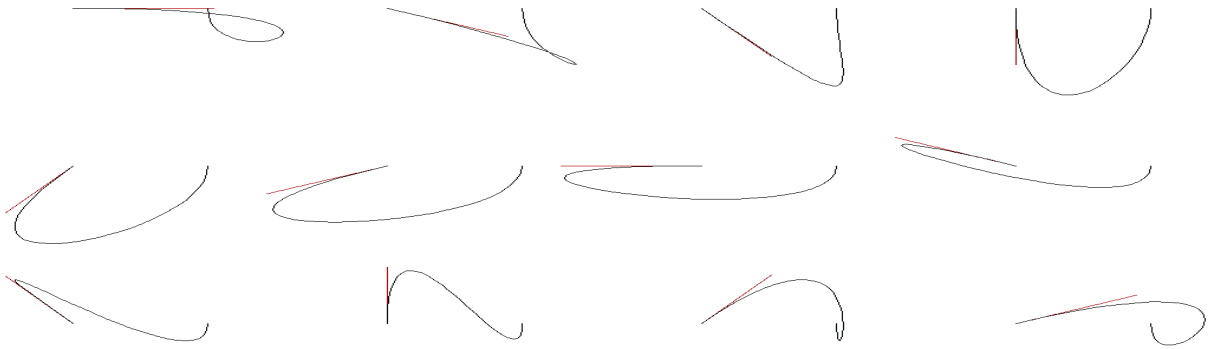
Por lo tanto, para encontrar el arreglo de coeficientes debemos encontrar la matriz inversa, denominada matriz de Hermite  $M_H$ :

$$M_H = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix}^{-1} = \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}.$$

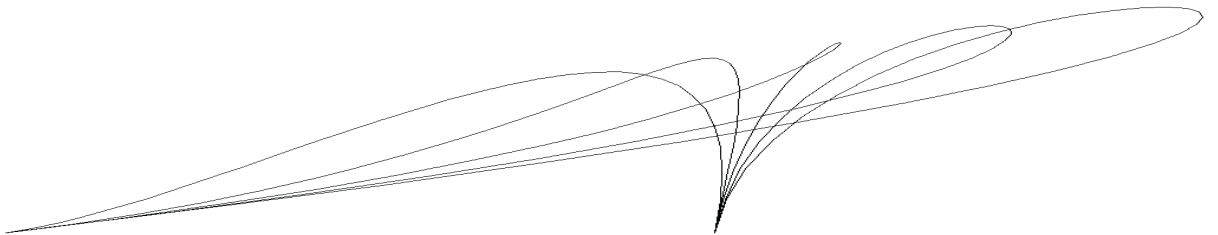
Nuestro problema de interpolación se transformó en un producto de un vector por una matriz. Lo más importante a tener en cuenta con este método es que  $M_H$  es la misma matriz para los polinomios  $f_x$  y  $f_y$  de cada segmento  $C_i$  de la curva. Con los dos polinomios de cada segmento de curva, entonces, se elige una secuencia (probablemente uniforme) de valores de  $u$  entre cero y uno. La evaluación de los polinomios en cada uno de dichos valores de  $u$  produce una secuencia de puntos, los cuales, unidos por una poligonal, se aproximan a la curva de Hermite con un grado de precisión arbitrario. Por lo tanto, un procedimiento que recibe 2 puntos de control y sus derivadas y los interpola con un segmento polinomial cúbico puede seguirse del procedimiento mostrado en la Figura 4.8.

Dicho procedimiento debe ser llamado  $n - 1$  veces para un grafo de control de  $n$  puntos. La modificación de cualquiera de los cuatro datos involucrados en los coeficientes de un polinomio modifica a todo el polinomio. Podemos ver en la Figura 4.9 la interpolación de Hermite entre dos puntos, variando la dirección de la derivada en el primero de ellos, y en la Figura 4.10 el efecto de modificar la magnitud de la misma. En este caso se modifican los dos segmentos de curva que concurren a dicho punto. Por lo tanto, el método de Hermite tiene control local. También podemos destacar que los datos de las derivadas son parámetros de diseño que se pueden modificar localmente para alterar la forma definitiva de la curva (ver Figura 4.11).

Es importante observar que la segunda derivada de los segmentos de curva no están restringidos, es decir, normalmente podemos esperar que  $\ddot{C}_i(1) \neq \ddot{C}_{i+1}(0)$ , es decir,

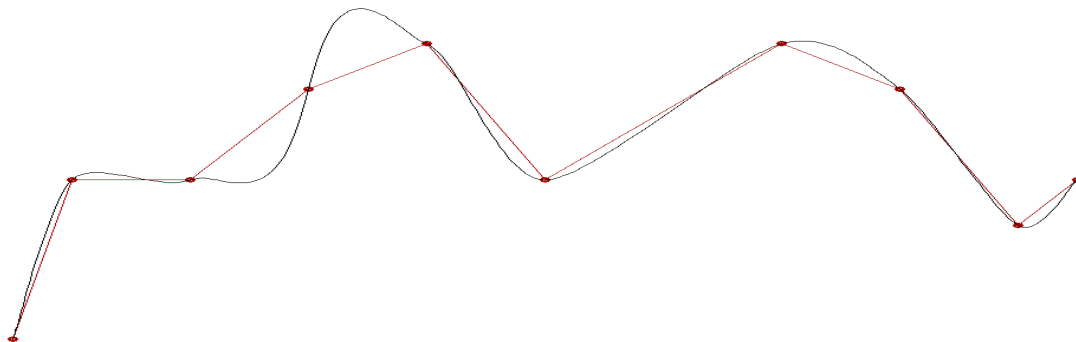


**Figura 4.9** *Modificación de la dirección de la derivada en el primer punto de control.*



**Figura 4.10** *Modificación de la magnitud de la derivada en el primer punto de control.*

existen discontinuidades de curvatura en los nudos. Una forma de evitar este problema es agregar condiciones de igualdad en la segunda derivada, y realizar el mismo desarrollo pero para un vector de coeficientes de un polinomio de quinto grado. Sin embargo, esto complica aún más la tarea artesanal de estimar la entrada al algoritmo, dado que hay que agregar estimadores de curvatura en los nudos. Dicha complicación, sumada a la relativa complejidad resultante de trabajar con polinomios de quinto grado, hacen que



**Figura 4.11** Interpolación de Hermite del mismo grafo de control que la Figura 4.7 modificando las restricciones de derivadas en el cuarto punto de control.

este método resulte inadecuado en la práctica, prefiriéndose los métodos que veremos en las siguientes Secciones de este Capítulo.

### 4.3 Aproximación de Curvas I: de Casteljau, Bernstein y Bézier

Los modelos de interpolación de curvas que vimos en la Sección anterior cumplen con el requisito de encontrar una curva que pase por una secuencia de puntos de control obedeciendo ciertas restricciones geométricas. Los resultados, sin embargo, son insatisfactorios desde el punto de vista del CAD y del modelado de primitivas gráficas en general. El modelo para construir curvas que veremos en esta Sección, denominadas *curvas de Bézier* es el primer método de aproximación desarrollado exclusivamente para los objetivos de la Computación Gráfica, principalmente en las compañías fabricantes de autos en Francia, durante los comienzos de la década del 70. De alguna manera, las curvas de Bézier comparten con los polinomios de Lagrange el hecho de que todos los puntos de control participan de la forma final de las mismas, es decir, tienen control global. Sin embargo, como veremos, la contribución de cada uno de los puntos determina una suma convexa, por lo que la curva no puede tener las oscilaciones indeseadas de Lagrange. Por el contrario, las curvas de Bézier suelen ser excesivamente suaves, y se amoldan al grafo de control de una manera perezosa.

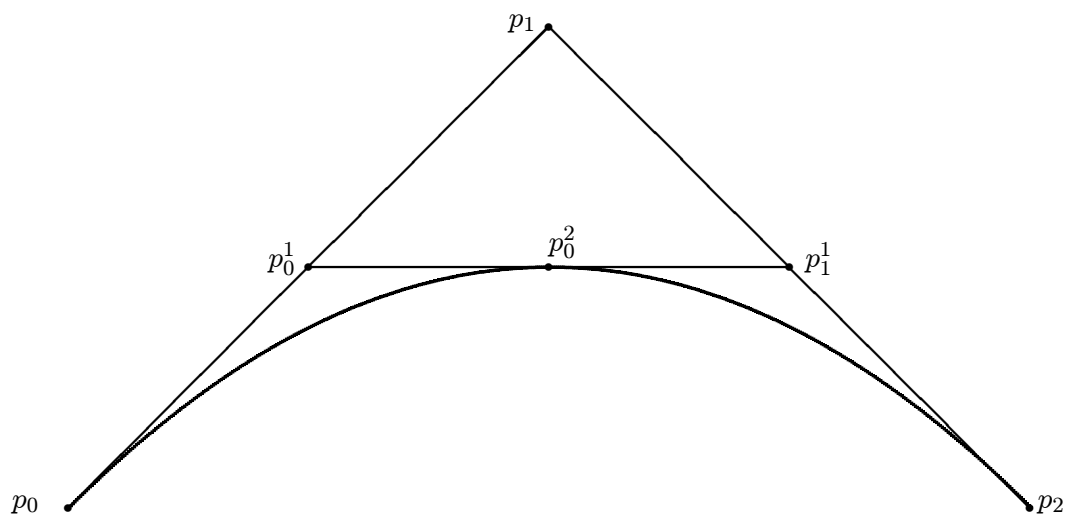


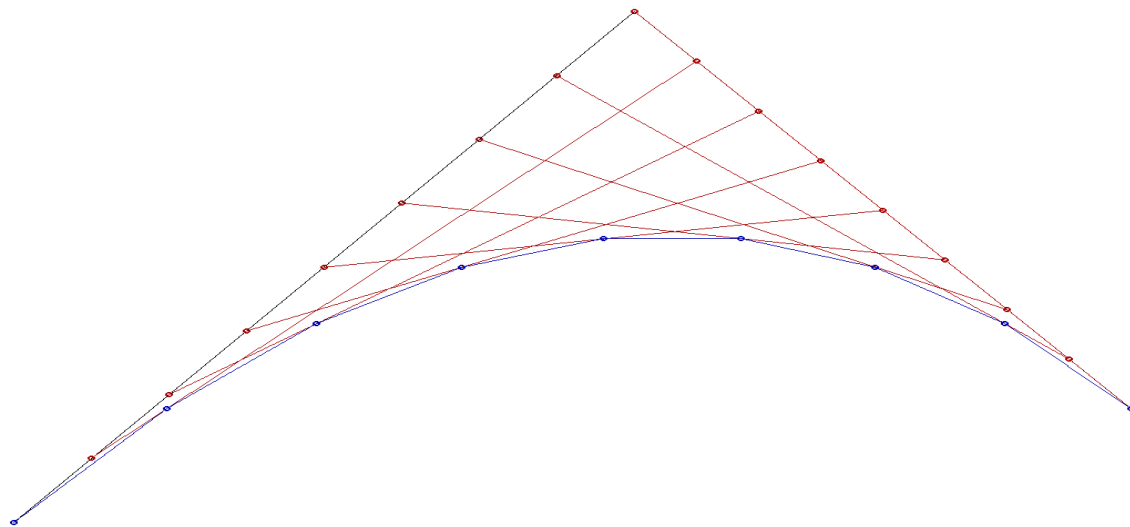
Figura 4.12 Construcción de parábolas.

El estudio de las curvas de Bézier es fundamental en este tema, porque provee los elementos geométricos y analíticos necesarios para comprender todos los demás temas en CAD. Esto es así dado que, como veremos a lo largo de este capítulo, todos los métodos de aproximación e interpolación (excepto el de Lagrange, porque el polinomio interpolante no está geoméricamente acotado) pueden plantearse como un conjunto de casos particulares de aproximaciones de Bézier.

#### 4.3.1 Construcción de parábolas

Una de las técnicas más antiguas para el trazado de parábolas en el dibujo técnico es la siguiente. Sean  $p_0, p_1, p_2 \in E^2$  (o  $E^3$ ) y un parámetro  $u \in [0, 1]$  (ver Figura 4.12). Entonces podemos mapear<sup>1</sup> distintos valores de  $u$  en  $[0, 1]$  a los segmentos  $\overline{p_0, p_1}$  y  $\overline{p_1, p_2}$ , obteniendo, respectivamente, los nuevos puntos  $p_0^1(u)$  y  $p_1^1(u)$ . Por último, mapeamos los mismos valores de  $u$  en el segmento  $\overline{p_0^1(u), p_1^1(u)}$ , obteniendo un punto  $p_0^2(u)$  que pertenece a la parábola.

<sup>1</sup>En todos los casos por *mapear* nos referimos a aplicar un mapeo afín.



**Figura 4.13** Aproximación de una parábola con interpolaciones lineales.

Utilizando la expresión paramétrica para representar un punto a lo largo de un segmento de recta, podemos encontrar expresiones para los puntos

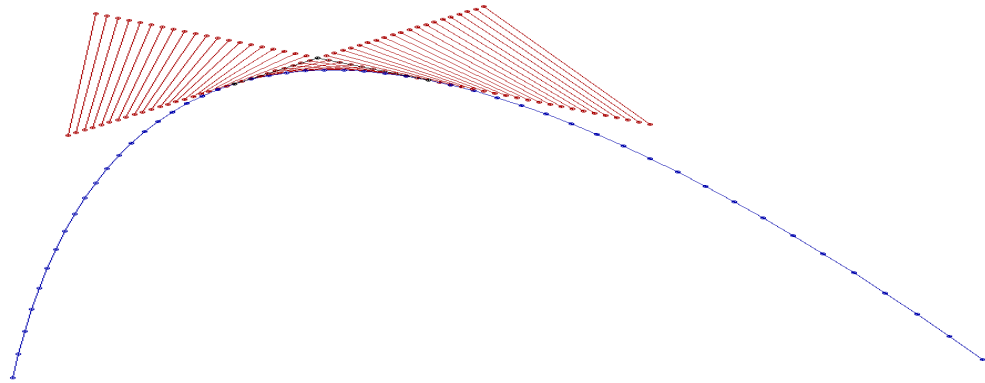
$$\begin{aligned} p_0^1(u) &= (1-u)p_0 + up_1 \\ p_1^1(u) &= (1-u)p_1 + up_2 \\ p_0^2(u) &= (1-u)p_0^1(u) + up_1^1(u), \end{aligned}$$

donde  $p_0^2(u)$  es la expresión paramétrica de una parábola tangente a los segmentos  $\overline{p_0, p_1}$  y  $\overline{p_1, p_2}$  en  $p_0$  y  $p_2$ , respectivamente. Si  $u \in [0, 1]$ , entonces la curva es el segmento de dicha parábola que va exactamente desde  $p_0$  hasta  $p_2$  (ver Figura 4.13).

Es importante tener en cuenta que el algoritmo sigue siendo correcto fuera del intervalo  $u \in [0, 1]$ . En dicho caso, se obtienen las partes correspondientes de la parábola por *extrapolación* lineal de los puntos de control (ver Figura 4.14). La demostración de que dicha curva es en efecto una parábola es un tanto indirecta y apela a la intuición geométrica del lector. Reemplazando las dos primeras ecuaciones anteriores en la tercera obtenemos

$$p_0^2(u) = (1-u)^2 p_0 + 2u(1-u)p_1 + u^2 p_2,$$

lo cual es una expresión de segundo grado, es decir, una cónica. Como es fácil ver, si los tres puntos de control no son colineales, entonces la curva no es un caso degenerado de cónica, por lo que puede ser una parábola, una elipse o una hipérbola. Pero, analizando



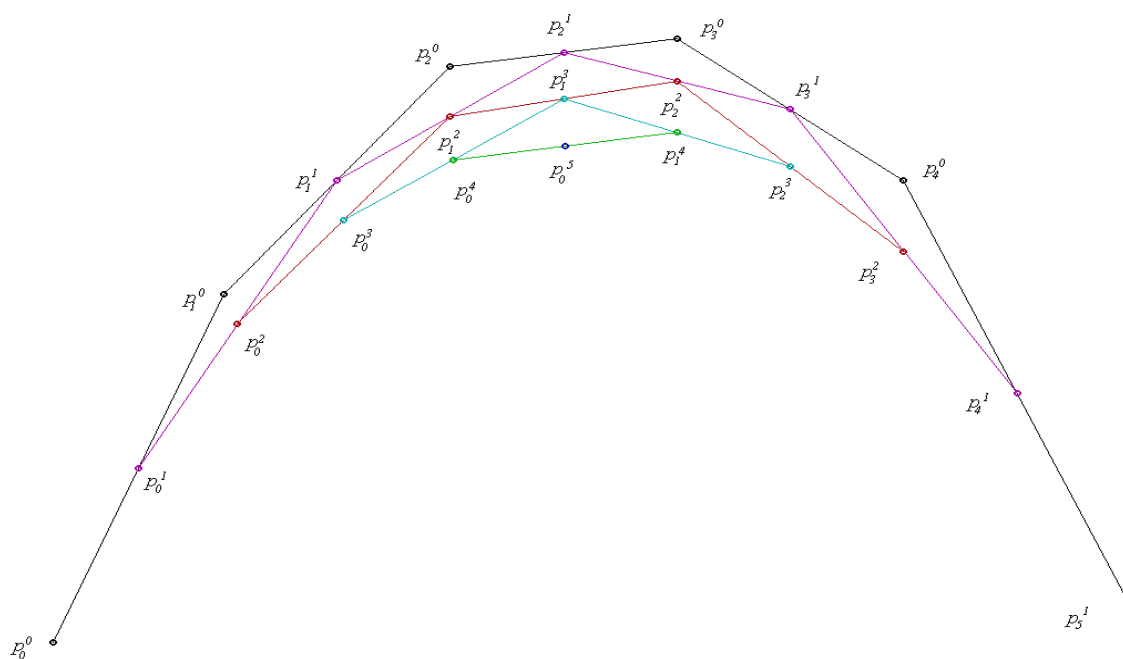
**Figura 4.14** Aproximación de una parábola fuera del intervalo  $u \in [0, 1]$ .

la dirección de su derivada, vemos que si  $u$  tiende a infinito, la dirección de la derivada tiende a  $p_0 - 2p_1 + p_2$ , y si  $u$  tiende a menos infinito, la dirección de la derivada tiende a  $-(p_0 - 2p_1 + p_2)$ . Por lo tanto, la curva tiene un solo eje, lo que excluye a las elipses (que no tienen asíntotas porque no se van a  $\infty$ ) y a las hipérbolas (que tienen dos asíntotas diferentes).

Es importante notar sin embargo que si  $u \in [0, 1]$ , entonces el segmento de curva es una suma convexa de los tres puntos que la determinan. Además, como todos los pasos involucrados en la construcción de la parábola son combinaciones lineales, una transformación afín aplicada a los puntos de control determina una nueva parábola que coincide con el resultado de aplicar la misma transformación afín a la parábola obtenida con los puntos sin transformar. Es decir, el procedimiento de construir una parábola es invariante frente a transformaciones afines.

### 4.3.2 El algoritmo de de Casteljau

Consideraremos ahora una generalización de la construcción de la parábola. Sea una secuencia de  $n + 1$  puntos o grafo de control  $p_0, \dots, p_n \in E^2$  (o  $E^3$ ) y un parámetro  $u \in [0, 1]$ . Podemos mapear el valor de  $u$  en  $[0, 1]$  a los segmentos  $\overline{p_i, p_{i+1}}$  obteniendo nuevos puntos  $p_i^1(u)$ ,  $i = 0..n - 1$ , donde  $p_i^1$  denota al  $i$ -ésimo punto de la secuencia de puntos de la primer interpolación. Luego mapeamos el valor de  $u$  en  $[0, 1]$  a los segmentos  $\overline{p_i^1(u), p_{i+1}^1(u)}$  obteniendo nuevos puntos  $p_i^2(u)$ ,  $i = 0..n - 2$ , donde  $p_i^2$  denota al  $i$ -ésimo punto de la secuencia de puntos de la segunda interpolación. En general,



**Figura 4.15** Aproximación de una secuencia de puntos de control por interpolaciones lineales sucesivas (algoritmo de de Casteljau).

mapeamos el valor de  $u$  en el segmento  $\overline{p_i^k(u), p_{i+1}^k(u)}$ , obteniendo un punto  $p_i^{k+1}(u)$ ,  $k = 1..n, i = 0..n - k$  (ver Figura 4.15). La recurrencia termina en el punto  $p_0^n(u)$ , el cual es un punto que pertenece a la *curva de Bézier*  $C(u)$  que *aproxima* al grafo de control inicial.

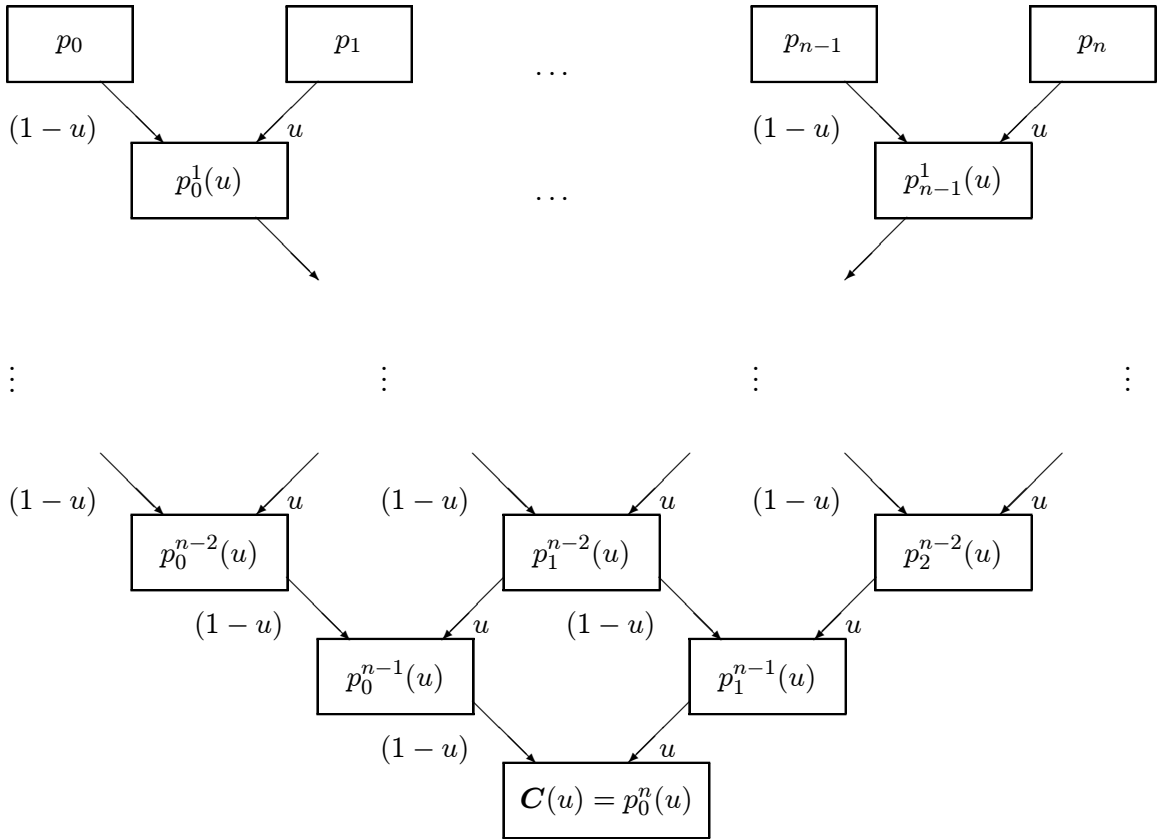


Figura 4.16 Secuencia de interpolaciones en el algoritmo de de Casteljau.

Si llamamos  $p_i^0(u) = p_i$  a los puntos de control iniciales (interpolaciones de orden cero), entonces el algoritmo define a la curva  $C(u) = p_0^n(u)$  computando la recurrencia

$$p_i^k(u) = (1 - u)p_i^{k-1}(u) + up_{i+1}^{k-1}(u) \quad (4.2)$$

con  $k = 1..n$ , e  $i = 0..(n - k)$  (ver Figura 4.16). Podemos ver el resultado de aplicar el algoritmo de de Casteljau sobre el mismo grafo de control pero con distintos valores de  $u$  entre 0 y 1 en la Figura 4.18.

```

procedure de_casteljau(u:real;g:grafo_control;var p:punto);
...
begin
  for k:=1 to n do begin
    for l:=0 to n-k do begin
      g[l].x:=(1-u)*g[l].x+u*g[l+1].x;
      g[l].y:=(1-u)*g[l].y+u*g[l+1].y;
    end;
  end;
  p.x:=g[0].x;          p.y:=g[0].y;
end;

procedure Bezier(var g:grafo_control);
...
begin
  p.x:= g[0].x;          p.y:=g[0].y;
  for k:=1 to paso do begin
    u:=k/paso;
    de_casteljau(u,g,p);
    graf_linea(p);
  end;
end;

```

**Figura 4.17** Procedimiento que implementa el algoritmo de de Casteljau.

Las curvas de Bézier se obtienen computando la recurrencia de de Casteljau para una cantidad suficiente de valores de  $u$  entre 0 y 1. Normalmente se asigna un paso fraccionario, se computa el valor de la curva incrementando dicho paso, y se unen dichos valores con una poligonal (ver Figura 4.17).

Es posible demostrar geométricamente que la curva de Bézier así definida tiene todas las propiedades requeridas. Es invariante frente a transformaciones afines, dado que está definida exclusivamente en base a interpolaciones lineales. Tiene la propiedad de *control global*, dado que (excepto en los extremos), todos los puntos de control ejercen influencia sobre su forma. Tiene la propiedad de *armazón convexo*, dado que cada punto de la curva se obtiene como combinaciones afines convexas de los puntos de control. La curva comienza en  $p_0$  con tangente  $\overline{p_1, p_0}$  y termina en  $p_n$  con tangente  $\overline{p_{n-1}, p_n}$  (ver Figura 4.19).

Es posible hacer coincidir los puntos de control extremos para obtener una curva cerrada. Es más, si  $\overline{p_1, p_0}$  es colineal con  $\overline{p_{n-1}, p_n}$ , entonces la curva es continuamente

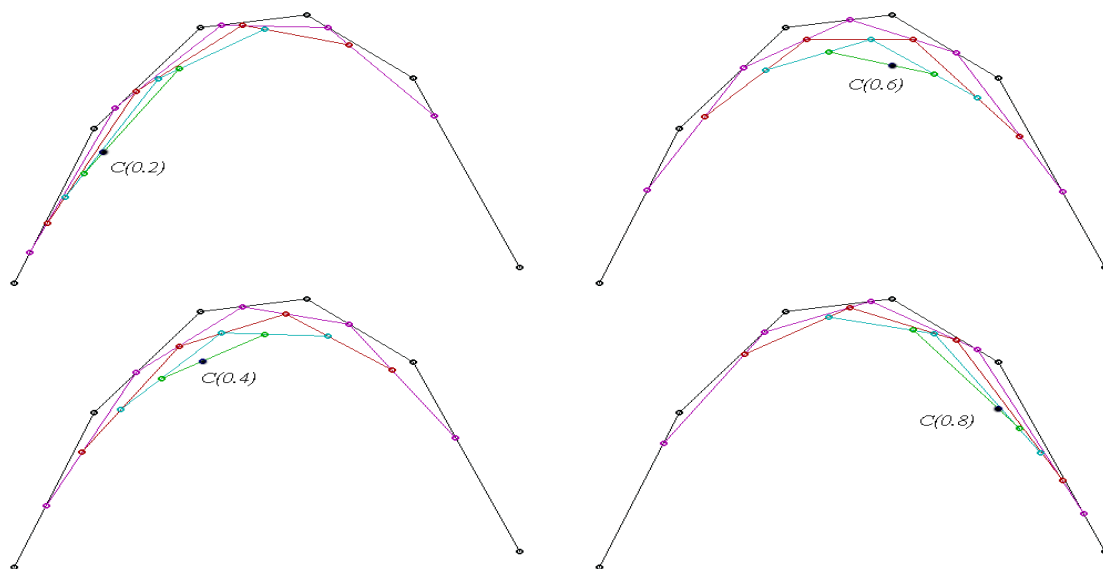


Figura 4.18 Algoritmo de de Casteljau con distintos valores del parámetro  $u$ .

derivable en dicha unión. También se muestra en dicha figura que es posible repetir consecutivamente la posición de un punto para que éste ejerza una mayor influencia en la forma de la curva (ver Figura 4.20).

### 4.3.3 La base de Bernstein

Uno de los inconvenientes del algoritmo de de Casteljau es que debe realizar  $n^2$  interpolaciones, muchas de las cuales son redundantes, dado que lo importante es el resultado final y no los resultados intermedios de la computación. Es posible encontrar una formulación alternativa de menor costo computacional. La idea es expresar a la curva  $C(u)$  como una combinación afín convexa de los puntos de control:

$$C(u) = [B_0^n(u), B_1^n(u), \dots, B_n^n(u)] \begin{bmatrix} p_0 \\ p_1 \\ \dots \\ p_n \end{bmatrix}, \quad (4.3)$$

donde el vector  $[B_0^n(u), B_1^n(u), \dots, B_n^n(u)]$  es un conjunto de funciones que debe conformar una *base funcional* para los polinomios de grado  $n$ , diferente a la base polinomial de Hermite vista en la Sección anterior.

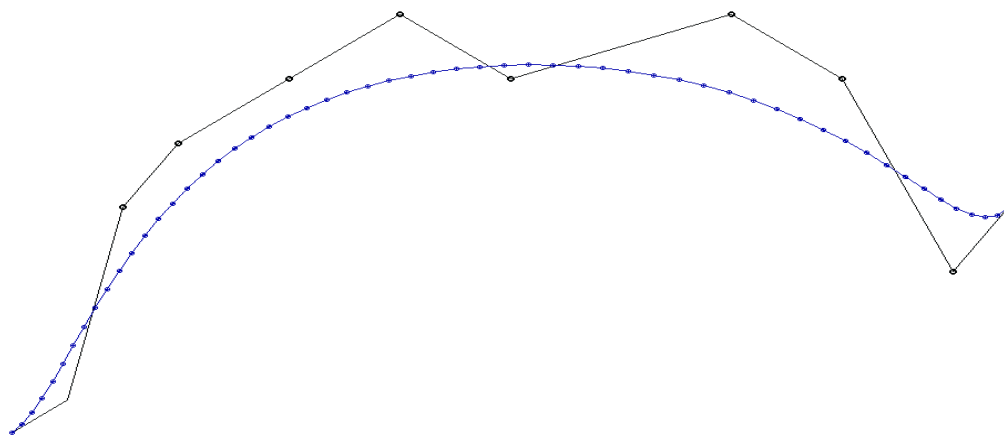


Figura 4.19 Un ejemplo de curvas de Bézier.

Otra forma de interpretar esta formulación es como una sumatoria:

$$C(u) = \sum_{i=0}^n B_i^n(u) p_i,$$

donde la función  $B_i^n(u)$  es la *función de mezcla* asociada al punto de control  $p_i$ . Luego de examinar la Figura 4.16, podemos ver que la contribución de cada punto a  $C(u)$  depende de productos de  $u$  y  $(1 - u)$ . Por ejemplo,  $p_0$  tiene una contribución de la forma  $(1 - u)^n$ ,  $p_1$  influye con  $n(1 - u)^{n-1}u$ , etc.

Bézier propuso utilizar dichas funciones como base para la curva, siendo que las mismas habían ya sido estudiadas con el nombre de los polinomios de Bernstein. De esa manera, la forma de Bézier de una curva utiliza la base de Bernstein:

$$B_i^n(u) = \frac{n!}{i!(n-i)!} u^i (1-u)^{n-i}. \quad (4.4)$$

La serie de números  $\frac{n!}{i!(n-i)!}$  es la conocida como “triángulo de Pascal”, y es muy sencilla de calcular. Es posible comprobar que la formulación de la curva  $C(u)$  expresada en la Ecuación 4.2, con  $C(u) = p_0^n(u)$ , coincide con la Ecuación 4.3 a través de un análisis cuidadoso de la Figura 4.21, donde se puede ver por ejemplo el gráfico de la

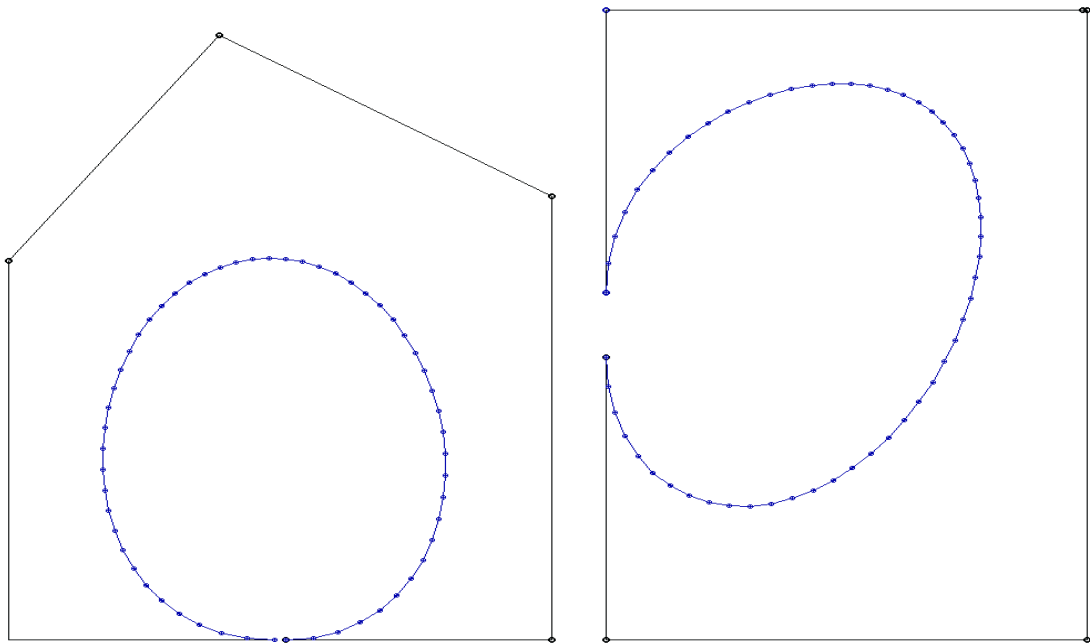


Figura 4.20 Curvas de Bézier cerradas y con puntos repetidos.

base de polinomios de Bernstein cúbicos. Dejamos como ejercicio para el lector que constate geoméricamente, en la Figura 4.19, que puntos interpolados sucesivamente con el algoritmo de de Casteljaú para distintos valores de  $u$  son idénticos a los obtenidos por medio de la Ecuación 4.3.

La base de Bernstein posee varias propiedades:

- Produce siempre combinaciones afines de puntos. Esto es así porque en la Ec. 4.4 podemos ver que  $\sum_{i=0}^n B_i^n(u) = [(1-u) + u]^n = 1$  (es decir, la suma de todas las bases de orden  $n$  coincide con el desarrollo del binomio de Newton correspondiente).
- $B_i^n(u) \geq 0$  si  $0 \leq u \leq 1$ , es decir, cada punto de la curva es una combinación convexa de todos los puntos de control.
- $B_i^n(u) = (1-u)B_i^{n-1}(u) + uB_{i+1}^{n-1}(u)$  (es recursiva).
- Es muy económica de computar.

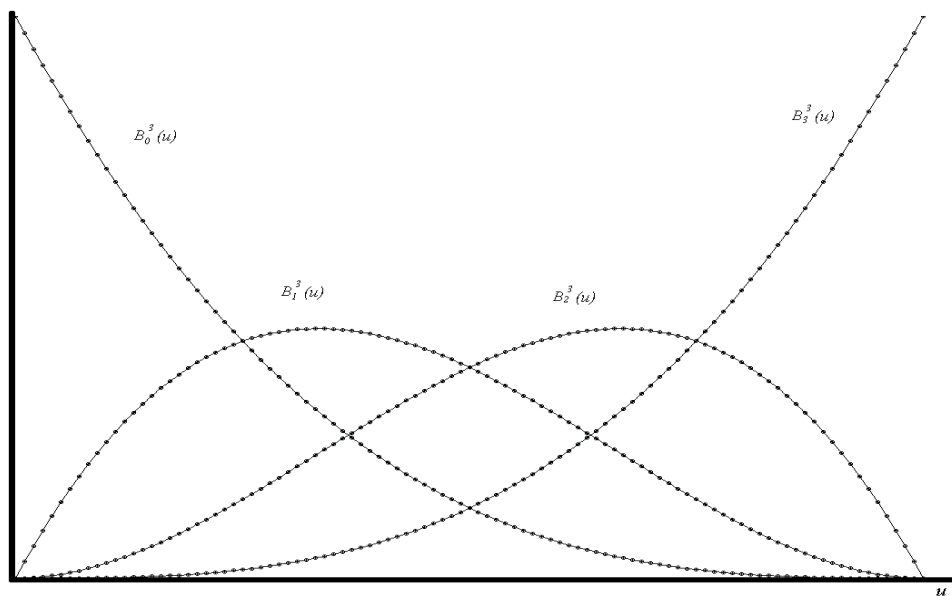


Figura 4.21 Polinomios de Bernstein cúbicos.

#### 4.4 Aproximación de Curvas II: B-Splines

Las curvas de Bézier son de grado muy alto para grafos de control complejos. Además tienden a suavizar excesivamente la geometría del grafo de control, es decir, se vuelven “insensibles” a pequeños cambios locales. Este efecto se puede constatar al considerar la escasa importancia que tiene un punto de control intermedio en una curva de grado 20 (ver Figura 4.22). Además, en determinadas circunstancias es simplemente preferible no tener control global. Por dicha razón, otra de las alternativas usuales en Computación Gráfica para aproximar curvas es utilizar Splines. El objetivo de estas curvas es obtener control local y bajo grado.

Los Splines deben su nombre a las herramientas curvilíneas utilizadas por los dibujantes para el trazado de figuras curvas. En matemática se utiliza el término para referirse a funciones definidas como uniones de segmentos polinomiales. Por lo tanto, el método de Hermite visto hace un par de Secciones es implícitamente un Spline cúbico. Los Splines más usados en Computación Gráfica, sin embargo, utilizan formulaciones semejantes a la de las curvas de Bézier vista en la Sección anterior. Es decir, son Splines determinados por bases funcionales, y por lo tanto se los denomina B-Splines.

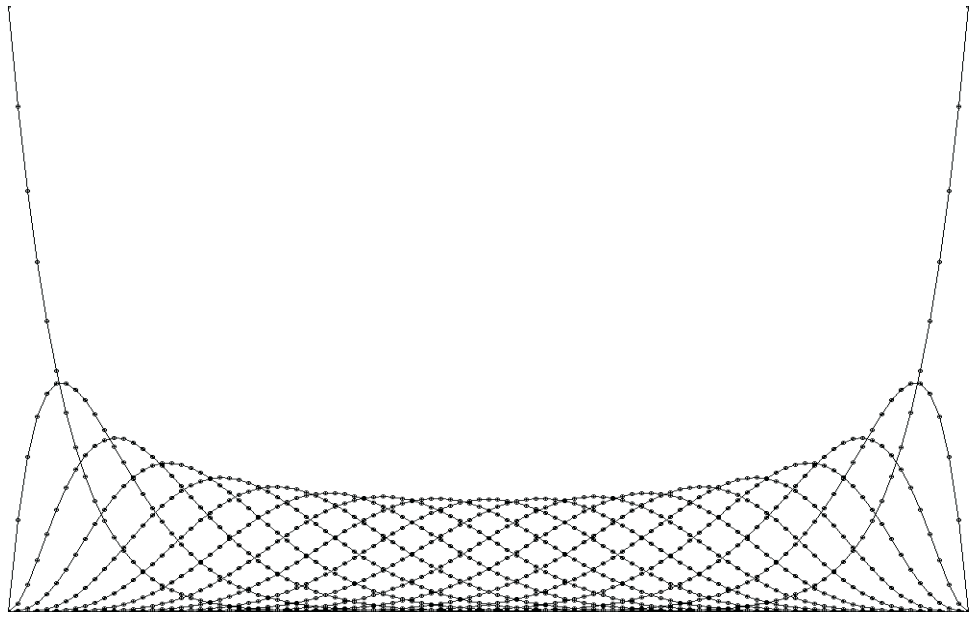
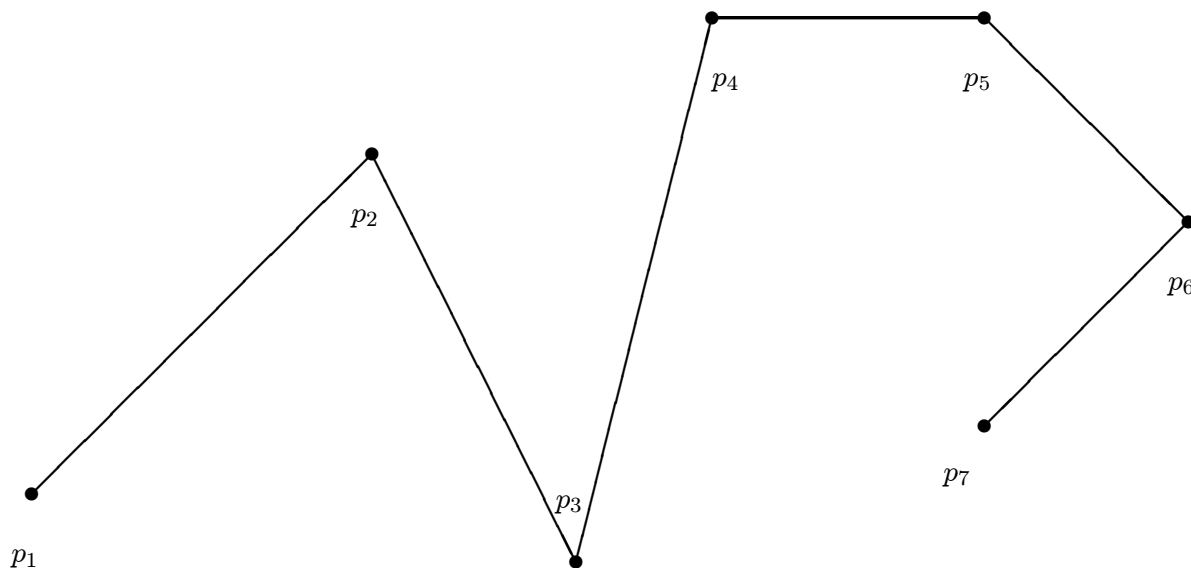


Figura 4.22 Polinomios de Bernstein de orden 20.

La diferencia entre Bézier y B-Splines es que las bases de esta última formulación tienen *soporte local*, es decir, son no nulas solamente para un subconjunto de la secuencia de puntos de control. La curva, entonces, es una combinación afín convexa de un subconjunto localmente determinado de los puntos de control. El grado de los polinomios en un B-Spline está determinado por la cantidad de puntos que localmente afectan a la curva, y determina el orden de continuidad de la misma, es decir, la cantidad de veces que es continuamente diferenciable en las uniones. El grado usual de una curva B-Spline es  $n = 3$ , lo que determina que cada segmento queda definido por  $n + 1$  puntos de control sucesivos, y que en las uniones el orden de continuidad es  $n - 1$ .

#### 4.4.1 Parámetro local

Una forma de entender la mecánica en la definición de una curva B-Spline es partir del caso trivial en el cual la “curva” aproximante es de primer grado. Podemos pensar que la poligonal es la “curva” resultante de unir los puntos de control con segmentos de polinomios de grado  $n = 1$ , definidos entre  $n + 1 = 2$  puntos sucesivos, y con un orden de continuidad  $n - 1 = 0$  en las uniones (ver Figura 4.23).



**Figura 4.23** Una “curva” Spline, de primer grado.

Cada segmento está definido como

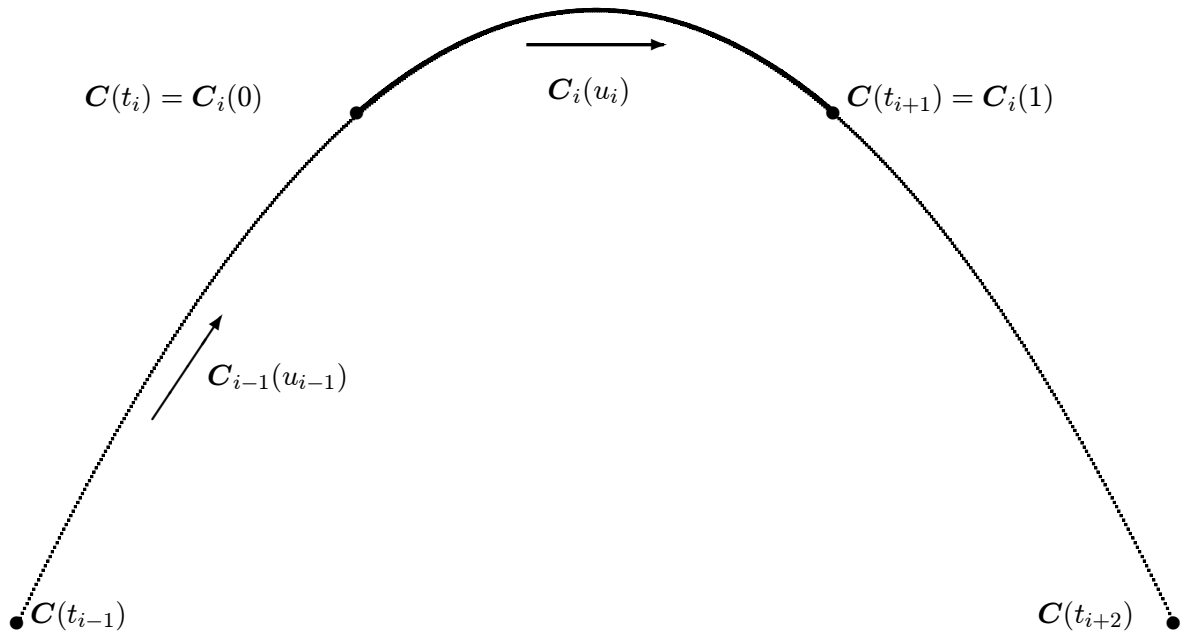
$$C_i(u) = (1 - u)p_i + up_{i+1},$$

donde  $0 \leq u \leq 1$ . Podríamos definir la curva como la unión de todos los segmentos:

$$C = \bigcup_{i=0}^{k-1} C_i.$$

El problema es que todos los segmentos están definidos para un mismo parámetro  $u$ . Por lo tanto, es necesario utilizar algún mecanismo que permita describir a la curva con un único parámetro global, el cual está relacionado con los parámetros locales en cada segmento polinomial.

Observemos que los segmentos polinomiales se unen entre sí en ciertos puntos definidos para formar la curva. Dichos puntos, por analogía con Lagrange, se denominan *nudos*. El parámetro global  $u$  asume el valor  $t_i$  al pasar por cada uno de dichos nudos. La secuencia  $t_0, t_1, \dots, t_k$  de valores del parámetro global asociados a cada uno de los nudos es no decreciente. Se define entonces un *parámetro local*  $u_i$  para cada segmento polinomial de curva  $C_i$ .  $u_i$  varía entre 0 y 1 mientras  $u$  varía entre  $t_i$  y  $t_{i+1}$ . El punto que ocupa la curva cuando el valor del parámetro global es el valor del nudo  $t_i$  tiene que coincidir con el comienzo del  $i$ -ésimo segmento de curva, es decir,  $C(t_i) = C_i(0)$ , y también con el final del  $i - 1$ -ésimo segmento, es decir,  $C(t_i) = C_{i-1}(1)$  (ver Figura 4.24).



**Figura 4.24** Relación entre segmentos de curva, parámetros locales y parámetro global.

El cálculo del valor del parámetro local en función del valor del parámetro global y de los valores en los nudos se efectúa por interpolación lineal:

$$u_i = \frac{u - t_i}{t_{i+1} - t_i}.$$

De esa manera, podemos pasar de la representación global de la curva  $C(u)$  a las representaciones locales (es decir, definidas en cada uno de los segmentos polinomiales). Por ejemplo, si la secuencia de nudos son los enteros no negativos, entonces dado el un valor  $u$  del parámetro global, el segmento polinomial “activo” es  $C_i$ , donde  $i$  es la parte entera de  $u$ , y el valor del parámetro local  $u_i$  es la parte fraccionaria de  $u$ , es decir  $u - i$ .

#### 4.4.2 La base de Splines

Buscaremos ahora una formulación en términos de una base funcional para nuestro ejemplo de polinomial a trozos, y luego extenderemos dicha formulación para grados superiores. Cada  $C_i$  es una combinación convexa de los puntos

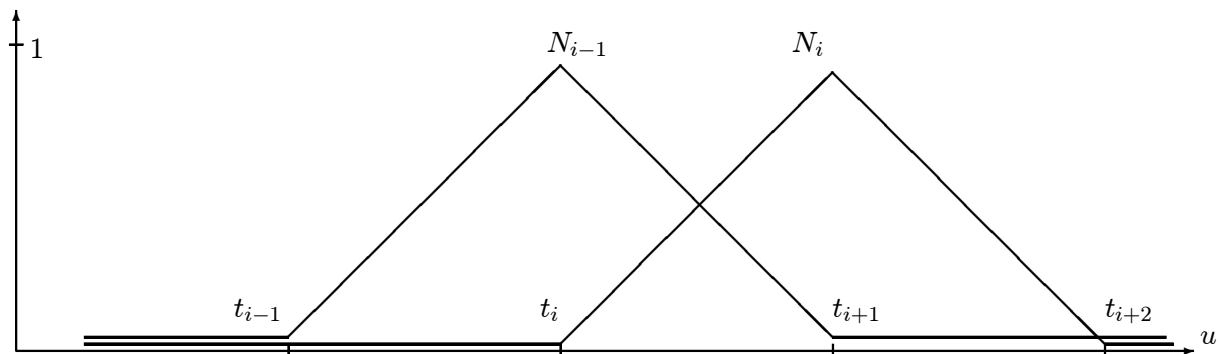


Figura 4.25 Las bases B-Splines de primer grado.

$p_i$  y  $p_{i+1}$ :

$$C_i(u_i) = (1 - u_i)p_i + u_i p_{i+1}.$$

Recordando la relación entre parámetro global y parámetros locales podemos reescribir la combinación convexa como:

$$C_i(u) = \frac{t_{i+1} - u}{t_{i+1} - t_i} p_i + \frac{u - t_i}{t_{i+1} - t_i} p_{i+1},$$

con  $t_i \leq u < t_{i+1}$ .

El punto  $p_i$  contribuye al segmento de curva  $C_{i-1}(u)$  con un factor

$$\frac{u - t_{i-1}}{t_i - t_{i-1}},$$

y al segmento de curva  $C_i(u)$  con un factor

$$\frac{t_{i+1} - u}{t_{i+1} - t_i}.$$

El factor que aporta  $p_i$  puede pensarse como una función  $N_i(u)$ , la cual es, en efecto, una base funcional para funciones lineales con soporte local, es decir, distinta de cero solo en un subconjunto conexo del dominio. La definición de  $N_i(u)$  es entonces la de una función lineal a trozos (ver Figura 4.25):

$$N_i(u) = \begin{cases} \frac{u - t_i}{t_{i+1} - t_i} & \text{si } t_i \leq u < t_{i+1}, \\ \frac{t_{i+2} - u}{t_{i+2} - t_{i+1}} & \text{si } t_{i+1} \leq u < t_{i+2}, \\ 0 & \text{en todo otro caso.} \end{cases} \quad (4.5)$$

Entonces nuestra “curva” queda expresada como

$$C(u) = \sum_{i=0}^n N_i(u)p_i.$$

Lo importante de la formulación en Splines es que la base de funciones está compuesta por una *única* función, cuyo soporte local es trasladado (ver Figura 4.25).

Para curvas de orden  $k > 1$ , la idea general es encontrar una familia de bases  $N_i^k(u)$  con soporte local, respetando la formulación de la curva como producto de una base funcional por un conjunto de puntos. La derivación de la familia  $N_i^k(u)$  es relativamente compleja. Puede hacerse por integración sucesiva, lo cual lleva a una expresión recursiva, bastante similar a la encontrada en la Sección anterior para de Casteljau-Bézier:

$$N_i^k(u) = \frac{u - t_i}{t_{i+k-1} - t_i} N_i^{k-1}(u) + \frac{t_{i+k} - u}{t_{i+k} - t_{i+1}} N_{i+1}^{k-1}(u), \quad (4.6)$$

con

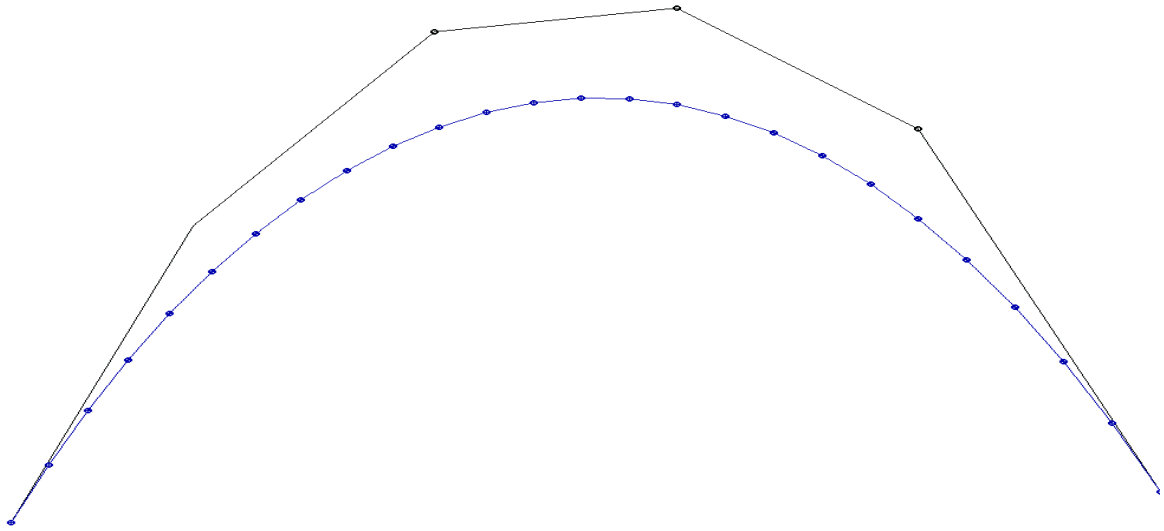
$$N_i^1(u) = \begin{cases} 1 & \text{si } t_i \leq u < t_{i+1}, \\ 0 & \text{en todo otro caso.} \end{cases}$$

$N_i^2(u)$  es idéntica a la función  $N_i(u)$  vista en la ecuación 4.5. Un ejemplo de curva B-Spline de grado quinto se puede observar en la Figura 4.26.

Por el momento asumimos que la secuencia de nudos  $t_i$  es uniforme, en particular que  $t_i = i$ , aunque luego veremos los efectos de alterar esta uniformidad. Para obligar a la curva a pasar por el primer y último punto de control, se repiten los mismos la cantidad necesaria de veces (según el grado de la curva), por medio de asignar un valor  $t_i = 0$  cuando  $i < 0$ , y un valor  $t_i = n - \text{grado} + 1$  cuando  $i > n$ . También luego veremos los efectos de alterar estas restricciones.

Los siguientes procedimientos muestran las implementaciones necesarias para computar las curvas B-Splines a partir de un grafo de control (asumimos vigentes muchas de las declaraciones de tipos de los trozos de código ya mostrados). En la Figura 4.27 podemos encontrar las funciones necesarias para computar los nudos para una parametrización uniforme, y la base recursiva de B-Splines, mientras que en la Figura 4.28 encontramos el procedimiento que utiliza las bases y la secuencia de puntos de control para computar la curva.

Podemos ver en la Figura 4.29 las funciones base de Splines de grado cúbico y de grado 10 utilizadas para aproximar un grafo de 11 puntos de control (comparar con los polinomios de Bernstein). Observar que para un orden  $k$  dado, desde  $i = k - 1$  hasta  $i = n - k + 1$  la base funcional  $N_i^k(u)$ , para distintos valores de  $i$ , es la misma función



**Figura 4.26** Curva B-Splines de grado 5.

pero desplazada en  $u$ . El efecto del aumento del grado de la curva es producir una disminución de la influencia local de los puntos de control (ver Figura 4.30).

Es importante considerar el efecto que tiene aumentar el grado de una curva B-Spline con respecto a variaciones locales en las posiciones de los puntos de control. En la Figura 4.31 se puede observar de qué manera el aumento del grado de la curva disminuye la influencia de un desplazamiento local (comparar con el método de Bézier).

Al mismo tiempo, y al igual que en el método de Bézier, es posible obtener curvas periódicas. Esto se consigue por medio de una asignación periódica a los índices de los puntos de control asociados a los nudos. De esa manera, se varía  $i$  entre  $-k$  y  $n + k$ , utilizándose valores  $t_i = i$  para los nudos, y la asignación periódica  $p_i \bmod n$ . Por último, también pueden repetirse puntos de control, aunque dado el control local de las curvas B-Splines, el efecto puede ser muy pronunciado.

```

const pts_ctrol=10; {n=10, luego 11 puntos}
      pasos=20;      {cantidad de evaluaciones}
      grado=3;        {grado deseado de la curva}

function nudo(i:integer):real;
begin
  if i < grado then nudo:=0
  else if i > pts_ctrol then nudo:=pts_ctrol-grado+2
  else nudo:=i-grado+1
end;

function base(u:real; i,k:integer):double;
var n1,n2,d1,d2,c1,c2:double;
begin
  if k=1 then
    if (u<nudo(i+1)) and (nudo(i)<=u) then base:=1
    else base:=0;
  else begin
    n1:=(u-i)*base(u,i,k-1);
    n2:=(nudo(i+k)-u)*base(u,i+1,k-1);
    d1:=nudo(i+k-1)-nudo(i);
    d2:=nudo(i+k)-nudo(i+1);
    if d1=0 then c1:=0 else c1:=n1/d1;
    if d2=0 then c2:=0 else c2:=n2/d2;
    base:=c1+c2;
  end;
end;

```

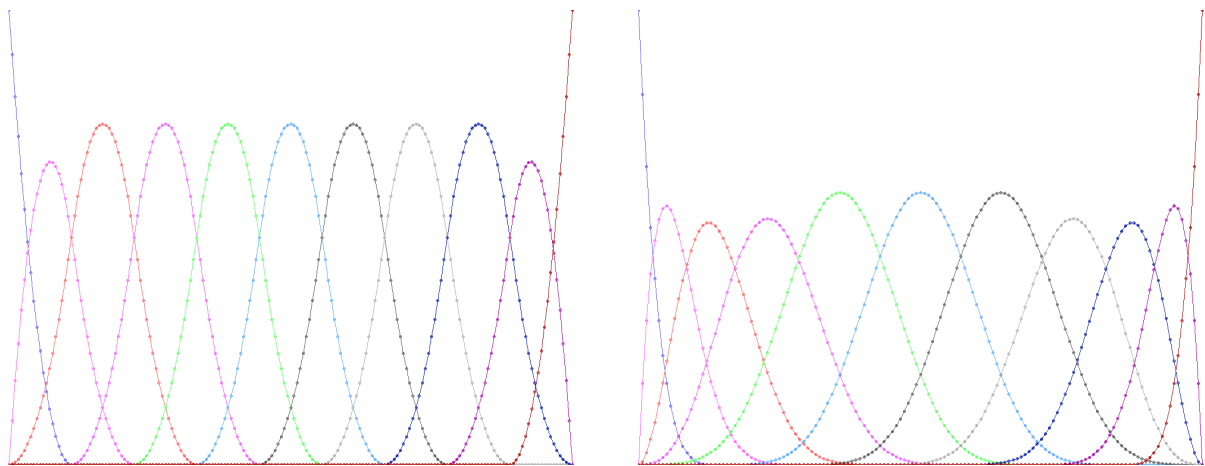
**Figura 4.27** Funciones para el cómputo de los valores de los nudos en la parametrización uniforme, y para el cómputo de la base de Splines recursiva.

```

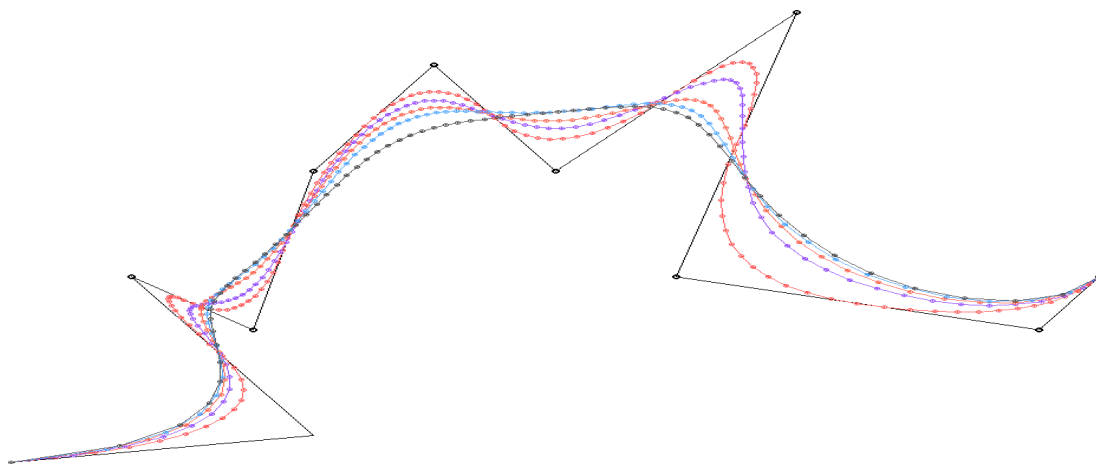
procedure bsplines(var g:grafo_contr);
var i,j:integer;
    u,v:real;
    p:punto;
begin
  for i:=0 to pasos*(pts_ctrol-grado+2) do begin
    u:=(i/pasos);
    p.x:=0; p.y:=0;
    for j:=0 to pts_ctrol do begin
      v:=base(u,j,grado);
      p.x:=p.x + g[j].x*v;
      p.y:=p.y + g[j].y*v;
    end;
    graf_linea(p);
  end;
end;

```

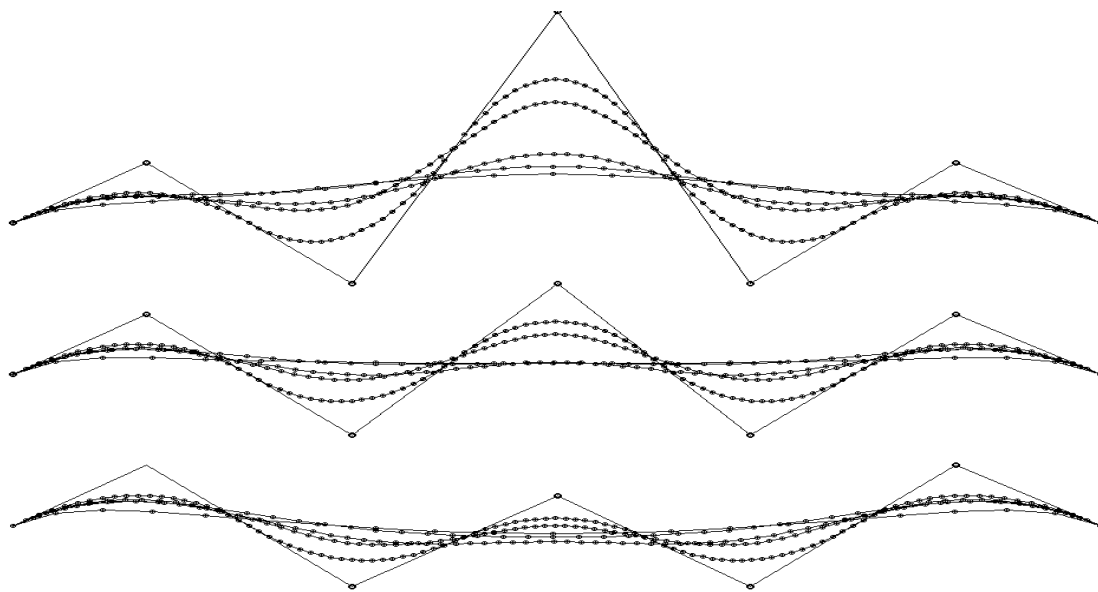
**Figura 4.28** Procedimiento que calcula la curva a partir de la base de Splines y de la secuencia de puntos de control.



**Figura 4.29** La base funcional de Splines de grados 3 y 5 para 11 puntos de control ( $n = 10$ ).



**Figura 4.30** Curvas B-Splines de grado 2, 3, 4 y 5 aproximando a un mismo grafo de control.



**Figura 4.31** Curvas B-Splines de grado 2, 3, 4 y 5 aproximando un grafo de control en el cual varía la posición de un punto.

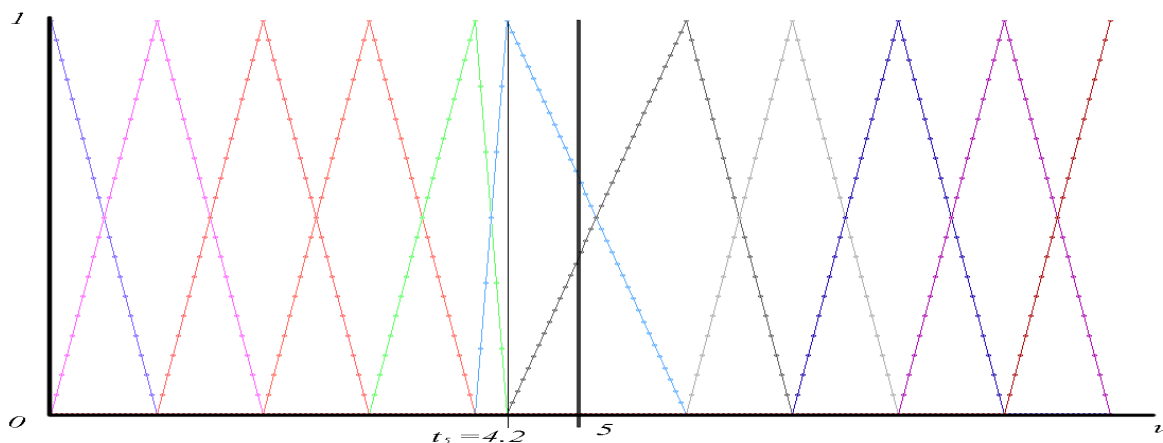


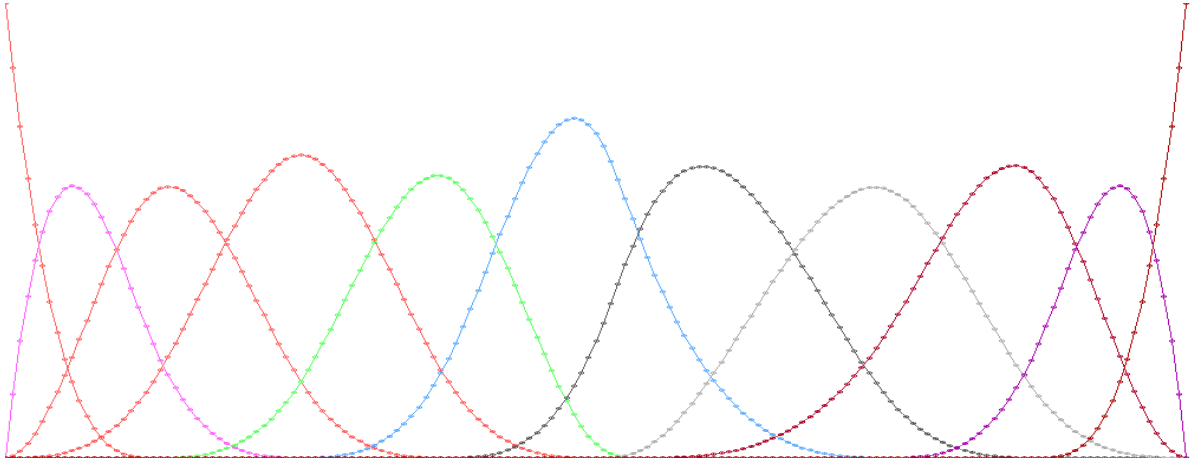
Figura 4.32 Bases de Spline de primer grado modificando el valor de un nudo.

#### 4.4.3 B-Splines no uniformes

¿Cuál es el resultado de alterar los valores en los nudos, respecto de la asignación uniforme  $t_i = i$ ? El efecto producido es el de alterar la velocidad paramétrica en la curva, es decir, cuánto se desplaza un punto  $p(u)$  respecto de un cambio en el valor de  $u$ . Supongamos tener una “curva” B-Spline de grado 1 con la parametrización uniforme mencionada más arriba. Las bases funcionales de esta “curva” son las funciones triangulares de la Figura 4.25. Modificar el valor  $t_i$  asociado al  $i$ -ésimo punto de control implica desplazar el valor donde ocurre el máximo de la  $i$ -ésima base funcional (ver Figura 4.32).

En este caso, el resultado de asignar a  $t_i$  un valor menor que  $i$  (pero mayor que  $i - 1$ , dado que la secuencia de nudos es no decreciente) es que la “curva” es recorrida más rápidamente que “lo normal” (el caso uniforme) con respecto al parámetro entre los puntos de control  $p_{i-1}$  y  $p_i$ , y más lentamente entre los puntos de control  $p_i$  y  $p_{i+1}$ . Pero el aspecto geométrico se mantiene constante. Sin embargo, cuando la curva es de un grado mayor, entonces es posible apreciar un efecto en las bases, dado que éstas provienen en definitiva de un promedio o integración ponderada de las bases lineales (ver Figura 4.33).

El resultado de disminuir o aumentar el valor de un nudo es similar a “acelerar” o “frenar” la curva cuando ésta pasa por el punto correspondiente a dicho nudo  $p(t_i)$ . Dicho efecto puede observarse en la Figura 4.34, donde se observa el resultado de

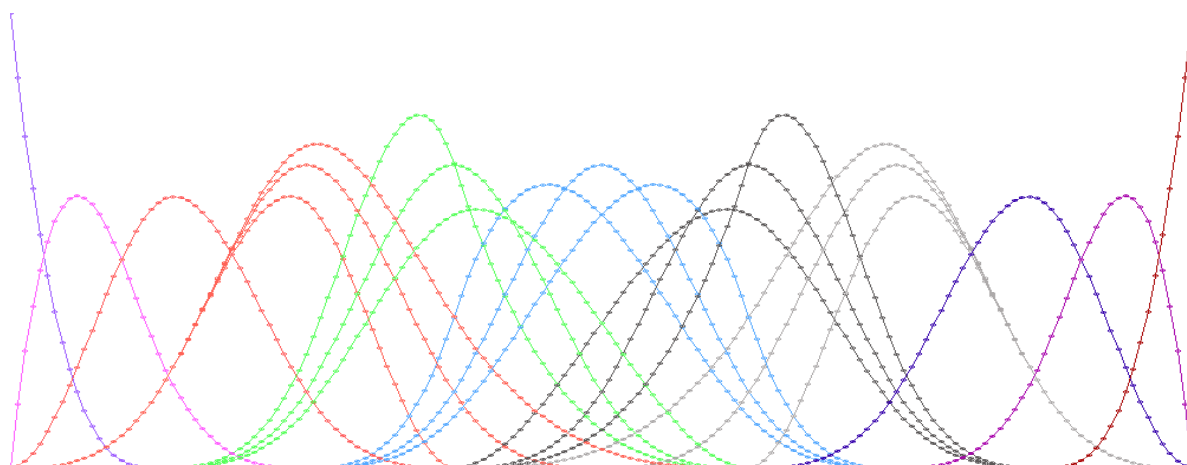


**Figura 4.33** Bases de Spline de tercer grado (con  $n = 10$ ) modificando el valor de un nudo.

aumentar o disminuir el valor de un nudo en la familia de bases. El resultado geométrico de dichas modificaciones puede observarse en la Figura 4.36, donde las curvas B-Splines no uniformes aproximan a un mismo grafo de control, pero modificando el valor del parámetro en el nudo asociado a un punto de control.

Está claro que pueden asignarse valores totalmente arbitrarios a los nudos, (inclusive a los que corresponden a la parte que antecede y sucede a los nudos asociados al grafo de control), siempre que se respete la condición de que la secuencia de nudos sea no decreciente. Una forma de implementar computacionalmente dicha secuencia es por medio de un arreglo auxiliar en vez de una función, como se muestra en la Figura 4.35.

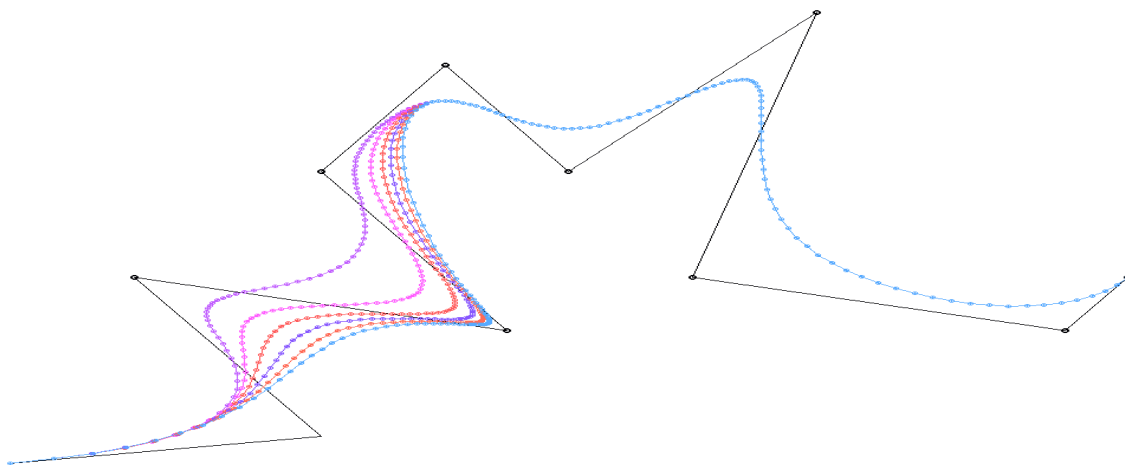
Recordamos que el valor de un nudo  $t_i$  es el valor que debe asumir el parámetro  $u$  cuando la curva pasa de un segmento polinomial al siguiente. Por lo tanto, la secuencia de nudos de una curva se denomina usualmente la *parametrización* de la curva, y permite obtener una gran variedad de resultados a partir de un mismo grafo de control. Sin embargo, no siempre es predecible o intuitivo el efecto que produce una determinada parametrización.



**Figura 4.34** Bases de Spline de tercer grado (con  $n = 10$ ) aumentando y disminuyendo el valor de un nudo.

```
...
var t: array [0..pts_ctrol-grado+2] of real;
...
function nudo(i:integer):real;
begin
  if i < grado then nudo:=t[0]
  else if i > pts_ctrol then nudo:=t[pts_ctrol-grado+2]
  else nudo:=t[i-grado+1]
end;
...
```

**Figura 4.35** Implementación de parametrizaciones no uniformes por medio de un arreglo de nudos.



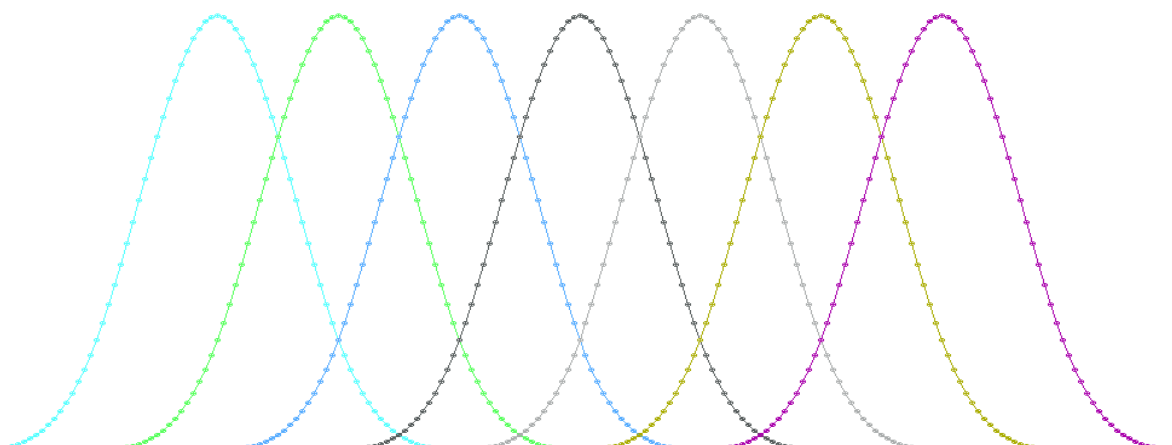
**Figura 4.36** Curva B-Spline de tercer grado disminuyendo y aumentando el valor de un nudo.

#### 4.4.4 B-Splines cúbicos uniformes

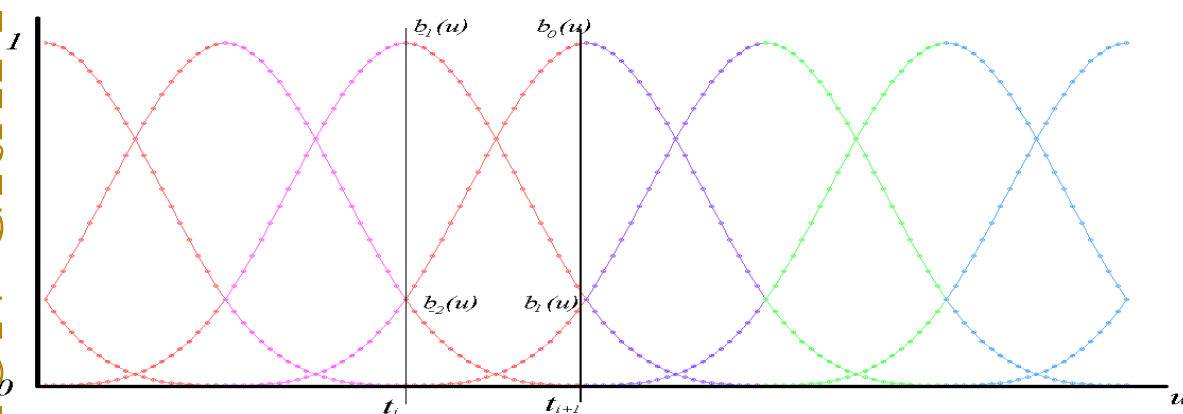
En la búsqueda de un método de aproximación geoméricamente versátil y computacionalmente económico, se plantean casos particulares para la ecuación 4.6 de modo tal que su expresión se simplifique. Específicamente se elige el menor grado que produzca resultados satisfactorios, junto con una parametrización uniforme que normalmente coincide con la secuencia de enteros no negativos. Lo usual es elegir  $k = 3$  de modo de tener orden de continuidad  $k - 1 = 2$ . Esto implica que cada segmento polinomial queda definido a partir de la ubicación de  $k + 1 = 4$  puntos de control. Este caso particular se conoce como B-Splines cúbicos uniformes, para los cuales derivaremos una formulación polinomial sencilla y un esquema computacional muy eficiente.

En este caso, las funciones de la base tienen que tener soporte entre  $k + 1 = 4$  nudos sucesivos. Es decir, cada punto ejerce su influencia en cuatro segmentos polinomiales. Supongamos que la función  $N_i^3(u)$  asociada a  $p_i$  tiene una forma como la mostrada en la Figura 4.37, y que la base asociado a todo otro punto de control es la misma, pero desplazada en  $u$ .

Entonces  $N_i^3(u_i)$  está compuesta por cuatro segmentos funcionales (o sub-bases), etiquetados  $b_1(u)$ ,  $b_0(u)$ ,  $b_{-1}(u)$ , y  $b_{-2}(u)$ , respectivamente, por razones que veremos



**Figura 4.37** La base funcional  $N_i^3(u)$  ignorando las primeras y últimas tres.



**Figura 4.38** La base funcional cúbica, y las cuatro sub-bases activas entre dos valores del parámetro global.

a continuación. En un intervalo entre dos nudos sucesivos, solo cuatro puntos influyen la posición de la curva. Es decir, cada segmento polinomial es una combinación convexa de cuatro puntos sucesivos, cada uno participando con una sub-base distinta (ver Figura 4.38).

Por dicha razón las sub-bases adoptan los curiosos nombres  $b_1(u)$ ,  $b_0(u)$ ,  $b_{-1}(u)$ ,  $b_{-2}(u)$ .  $b_1(u)$  es la sub-base activa asociada a  $p_{i+1}$ ,  $b_0(u)$  es la sub-base activa asociada a  $p_i$ ,  $b_{-1}(u)$  es la sub-base activa asociada a  $p_{i-1}$ , y  $b_{-2}(u)$  es la sub-base activa asociada a  $p_{i-2}$ . Es decir

$$\mathbf{C}_i(u_i) = \sum_{k=-2}^1 b_k(u_i) p_{i+k}.$$

Para encontrar las sub-bases podemos plantear las siguientes condiciones de continuidad.

$$\begin{array}{lll} b_1(0) = 0 & \dot{b}_1(0) = 0 & \ddot{b}_1(0) = 0 \\ b_1(1) = b_0(0) & \dot{b}_1(1) = \dot{b}_0(0) & \ddot{b}_1(1) = \ddot{b}_0(0) \\ b_0(1) = b_{-1}(0) & \dot{b}_0(1) = \dot{b}_{-1}(0) & \ddot{b}_0(1) = \ddot{b}_{-1}(0) \\ b_{-1}(1) = b_{-2}(0) & \dot{b}_{-1}(1) = \dot{b}_{-2}(0) & \ddot{b}_{-1}(1) = \ddot{b}_{-2}(0) \\ b_{-2}(1) = 0 & \dot{b}_{-2}(1) = 0 & \ddot{b}_{-2}(1) = 0 \end{array}$$

Es decir, tenemos cinco condiciones de continuidad en posición, en primera derivada y en segunda derivada. Cada sub-base es un polinomio cúbico, con cuatro incógnitas. Tenemos 16 incógnitas y 15 ecuaciones. La condición faltante es que

$$b_1(u) + b_0(u) + b_{-1}(u) + b_{-2}(u) = 1 \text{ para } 0 \leq u \leq 1$$

Resolviendo las ecuaciones obtenemos:

$$\begin{aligned} b_1(u) &= \frac{1}{6}u^3, \\ b_0(u) &= \frac{1}{6}(1 + 3u + 3u^2 - 3u^3), \\ b_{-1}(u) &= \frac{1}{6}(4 - 6u^2 + 3u^3), \\ b_{-2}(u) &= \frac{1}{6}(1 - 3u + 3u^2 - u^3). \end{aligned}$$

De esa manera, cada punto de control  $p_i$  tiene asociada una base compuesta por cuatro sub-bases no nulas, que determinan su influencia en la curva  $\mathbf{C}(u)$ . Dados  $n+1$  puntos de control  $p_0, p_1, \dots, p_n$ , la representación de la curva de B-Splines cúbica es:

$$\mathbf{C}(u) = \bigcup_{i=2}^{n-1} \mathbf{C}_i(u),$$

lo cual es equivalente a una expresión como sumatoria (al estilo de la formulación de Bézier):

$$\mathbf{C}(u) = \sum_{i=2}^{n-1} N_i^3(u) p_i.$$

```

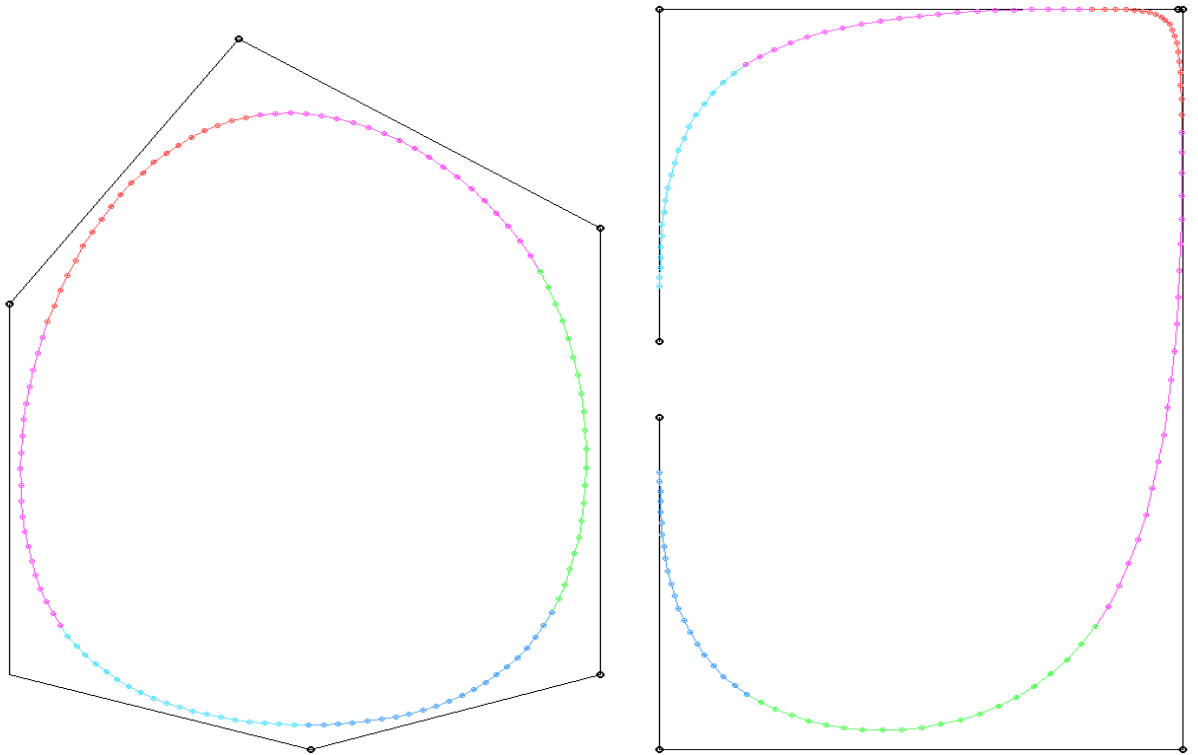
function base(u:real;i:integer):double;
begin
  case i of
    1 : base:=u*u*u/6;
    0 : base:=(1+u*3*(1+u*(1-u)))/6;
    -1 : base:=(4+u*u*3*(-2+u))/6;
    -2 : base:=(1+u*(-3+u*(3-u)))/6;
  end;
end;

procedure bsplines(var g:grafo_contr);
var i,j,k,index:integer;
    u,v:real;
    p:punto;
begin
  p.z:=1;    p.w:=1;
  for k:=1 to pasos do begin
    u:=k/pasos;
    p.x:=0;  p.y:=0;
    for i:=-2 to 1 do begin
      v:=base(u,i);
      index:=i+j;
      if index<0 then index:=0
        else if index>pts_ctrol then index:= pts_ctrol;
      p.x:=p.x+g[index].x*v;
      p.y:=p.y+g[index].y*v;
    end;
    graf_linea(p);
  end;
end;
end;

```

**Figura 4.39** Implementación de curvas Spline cúbicas uniformes por medio de sub-bases.

Por otra parte, teniendo en cuenta que las funciones base tienen soporte solamente en cuatro segmentos polinomiales, la expresión como sumatoria es ineficiente, siendo



**Figura 4.40** Curvas B-Splines de grado 3. La curva de la izquierda es periódica por medio de la asignación  $p_i \bmod n$ , mientras que la curva de la derecha tiene un punto de control repetido (levemente separado para que se aprecie su ocurrencia).

preferible expresarla como unión de los segmentos

$$C_i(u_i) = \sum_{k=-2}^1 b_k(u_i) p_{i+k}.$$

El trozo de código de la Figura 4.39 muestra una implementación de curvas B-Splines cúbicas uniformes. En la Figura 4.40 se observan curvas B-Spline cúbicas uniformes donde cada segmento polinomial fue computado con el algoritmo mostrado más arriba (y graficado en distinto color), tomando como ejemplo una curva cerrada utilizando asignación periódica, y un grafo de control con puntos de control repetidos.

Es posible observar que las curvas B-Spline no pasan por el primer y último punto de control. En efecto, dada una secuencia de puntos de control  $p_0, p_1, \dots, p_n$ , el segmento de curva

$$C_2(u) = b_{-2}(u)p_0 + b_{-1}(u)p_1 + b_0(u)p_2 + b_1(u)p_3$$

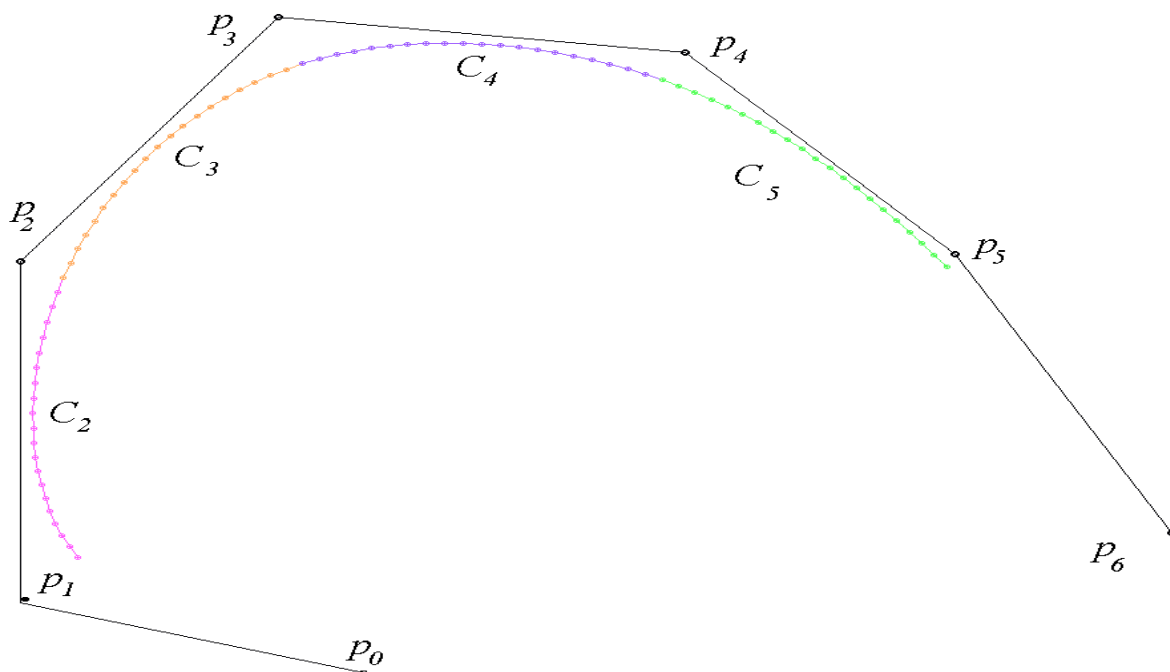


Figura 4.41 Los segmentos definidos de una curva B-Spline cúbica uniforme.

comienza en

$$C_2(0) = b_{-2}(0)p_0 + b_{-1}(0)p_1 + b_0(0)p_2 + b_1(0)p_3 = \frac{1}{6}p_0 + \frac{4}{6}p_1 + \frac{1}{6}p_2 + 0p_3.$$

El segmento  $C_1$  no está definido porque no tenemos punto de control  $p_{-1}$ . Lo mismo sucede con  $C_n(u)$ . En general, están definidos los segmentos  $C_2$  a  $C_{n-1}$  (ver Figura 4.41).

Para garantizar que la curva comience en  $p_0$  podemos *repetir* dicho punto. Repitiendo una sola vez, es decir, haciendo que la secuencia comience con  $p_{-1} = p_0$ , obtenemos la posibilidad de que  $C_1(u)$  esté definida, y

$$C_1(0) = b_{-2}(0)p_0 + b_{-1}(0)p_0 + b_0(0)p_1 + b_1(0)p_2 = \frac{1}{6}p_0 + \frac{4}{6}p_0 + \frac{1}{6}p_1 + 0p_2.$$

Es decir, la curva comienza en un punto que está a  $\frac{1}{6}$  de  $p_0$  y a  $\frac{5}{6}$  de  $p_1$ . La única forma de que la curva pase por  $p_0$  es repetirlo.

#### 4.4.5 Evaluación de B-Splines cúbicos

La representación de curvas por medio de B-Splines cúbicos uniformes quedó determinada por un esquema en el cual cada punto de la curva está determinado por la suma convexa de cuatro puntos de control, donde el coeficiente de cada punto es un polinomio cúbico. Por lo tanto, se requiere un esquema eficiente para evaluar funciones de la forma

$$p(u) = au^3 + bu^2 + cu + d,$$

con  $0 \leq u \leq 1$ .

Es posible encontrar dicha evaluación económica utilizando una metodología similar a la evaluación por diferencias finitas (DDA) como hicimos en el Capítulo 2 con rectas y circunferencias. Sea  $n + 1$  la cantidad de evaluaciones y sea el incremento en cada evaluación el valor

$$\delta = \frac{1}{n}.$$

La diferencia entre dos evaluaciones sucesivas

$$\Delta_1(i\delta) = p((i+1)\delta) - p(i\delta)$$

es una función de grado 2 en  $u$ . Lo mismo puede hacerse para diferencias de mayor orden:

$$\Delta_k(i\delta) = \Delta_{k-1}((i+1)\delta) - \Delta_{k-1}(i\delta), k > 1$$

hasta llegar a una función constante, en este caso

$$\Delta_3(i\delta) = \Delta.$$

De esa manera, conociendo  $\Delta$  y  $\Delta_2((i-1)\delta)$  podemos obtener  $\Delta_2(i\delta)$ , para todo  $i$  tal que  $0 \leq i \leq n$ . Igualmente, con  $\Delta_2(i\delta)$  y  $\Delta_1((i-1)\delta)$  obtenemos  $\Delta_1(i\delta)$ ,  $0 \leq i \leq n$ , y con  $\Delta_1(i\delta)$  y  $p((i-1)\delta)$  obtenemos  $p(i\delta)$ ,  $0 \leq i \leq n$ .

Por lo tanto, solamente es necesario conocer  $p$  y los distintos  $\Delta$  en  $u = 0$ . Un rápido análisis nos permite encontrar

$$\begin{aligned} p(0) &= d, \\ \Delta_1(0) &= p(\delta) - p(0) = a\delta^3 + b\delta^2 + c\delta, \\ \Delta_2(0) &= \Delta_1(\delta) - \Delta_1(0) = 6a\delta^3 + 2b\delta^2, \\ \Delta_3 &= \Delta_2(\delta) - \Delta_2(0) = 6a\delta^3. \end{aligned}$$

De esa manera, es posible computar un polinomio cúbico evaluado a intervalos regulares entre 0 y 1, utilizando solamente tres sumas (ver Figura 4.42).

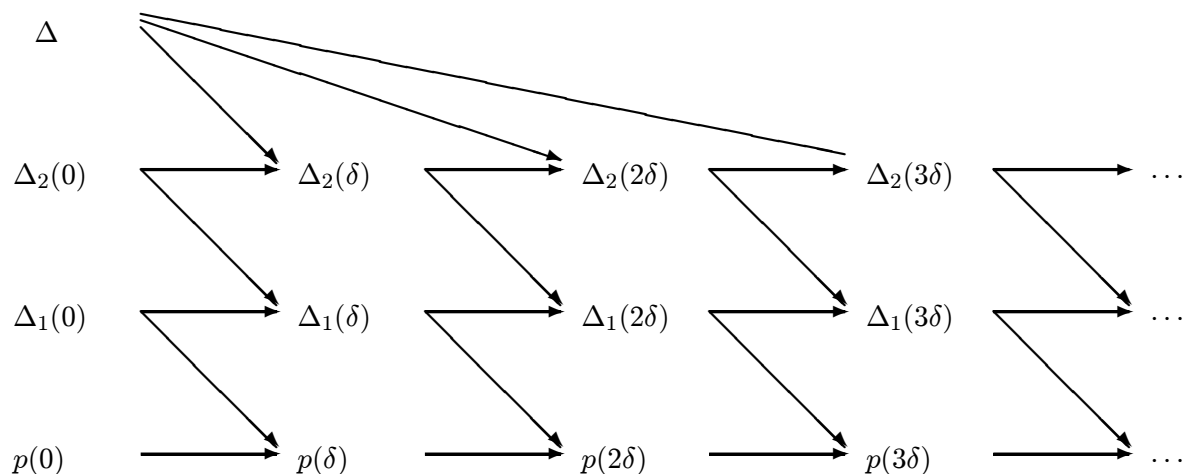


Figura 4.42 Evaluación de un polinomio cúbico por DDA.

## 4.5 Aproximación de Curvas III: Modelos Avanzados

Aunque no resulte aparente a primera vista, existen relaciones geométricas muy profundas entre los métodos de Hermite, Bézier y B-Splines. En particular, dado un grafo de control de  $n + 1$  puntos, los interpolantes de grado  $k$  de Hermite pueden transformarse en  $n - 1$  curvas de Bézier de grado  $k$  que se unen en los puntos de control con orden de continuidad  $k - 2$ , donde los  $k - 1$  puntos de control de cada curva de Bézier surgen de relaciones geométricas entre el grafo de control original y las restricciones de continuidad establecidas. Una situación similar ocurre con los B-Splines uniformes o no de grado  $k$ . Por lo tanto, dedicaremos la primera parte de esta Sección a desarrollar estos conceptos para el caso particular en el que  $k = 3$ .

El estudio de estas equivalencias geométricas es de gran importancia. Por una parte muestra en qué medida las diferencias entre los métodos estudiados son aparentes, pese a que cada uno fue desarrollado a partir de diferentes motivaciones. Esto le da al tema una notable maduración. Por otra parte, los métodos comparados dan origen a un “metamétodo” que permite definir y caracterizar abstractamente a cada caso particular, y posibilita la búsqueda de simplificaciones teóricas y algorítmicas.

### 4.5.1 Curvas de Hermite y B-Splines como curvas de Bézier

En la Sección 4.2 vimos que para interpolar por Hermite una curva cúbica  $C_i(u)$  que pase por  $p_i = (x_i, y_i)$  y por  $p_{i+1} = (x_{i+1}, y_{i+1})$ , con una determinada derivada  $\dot{p}_i$  en el primer punto y con una derivada  $\dot{p}_{i+1}$  en el segundo, necesitamos encontrar dos funciones  $f_x(u)$  y  $f_y(u)$  como polinomios cúbicos en  $u$ . Representando todas las restricciones en un único sistema de ecuaciones encontramos que:

$$\begin{bmatrix} f_x(0) \\ f_x(1) \\ \dot{f}_x(0) \\ \dot{f}_x(1) \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix} \begin{bmatrix} c_x^3 \\ c_x^2 \\ c_x^1 \\ c_x^0 \end{bmatrix}.$$

La solución de dicho sistema nos permite encontrar los coeficientes de los polinomios cúbicos buscados.

Si la curva de Hermite resultante entre  $p_i$  y  $p_{i+1}$  es una cúbica, y la base de Bernstein permite representar cualquier curva cúbica (dado que es, en efecto, una base), entonces debe existir una curva de Bézier cúbica idéntica al interpolante de Hermite. Por lo tanto, para describir dicha curva debemos encontrar los cuatro puntos que la definen. Por razones que quedarán claras en un momento, dichos puntos serán denotados como  $b_{3i}$ ,  $b_{3i+1}$ ,  $b_{3(i+1)-1}$  y  $b_{3(i+1)}$ .

Ahora bien, como quedó claro en la Sección 4.3, las curvas de Bézier comienzan y terminan en el primer y último punto de control, respectivamente. Por lo tanto

$$\begin{aligned} b_{3i} &= p_i, \\ b_{3(i+1)} &= p_{i+1}. \end{aligned}$$

Además, la curva de Bézier comienza siendo tangente en  $b_{3i}$  al segmento  $\overline{b_{3i}b_{3i+1}}$  y termina siendo tangente en  $b_{3(i+1)}$  al segmento  $\overline{b_{3(i+1)-1}b_{3(i+1)}}$ . Por lo tanto, dichos segmentos deben ser colineales a las derivadas  $\dot{p}_i$  y  $\dot{p}_{i+1}$ , respectivamente, que definen al interpolante de Hermite. Es decir

$$\begin{aligned} b_{3i-1} &= b_{3i} - \alpha \dot{p}_i, \\ b_{3i+1} &= b_{3i} + \alpha \dot{p}_i. \end{aligned}$$

Al asignar el mismo coeficiente, se determina que en los puntos  $b_{3i}$ , donde se unen dos curvas de Bézier (y por lo tanto denominado *punto de unión*), las dos curvas se unen con la misma derivada, y por lo tanto se unen en forma  $C^1$  continua (ver Figura 4.43).

Toda asignación de un coeficiente  $\alpha$  no trivial encuentra los demás puntos de Bézier (denominados *puntos interiores*), con los cuales se produce una interpolación cúbica  $C^1$

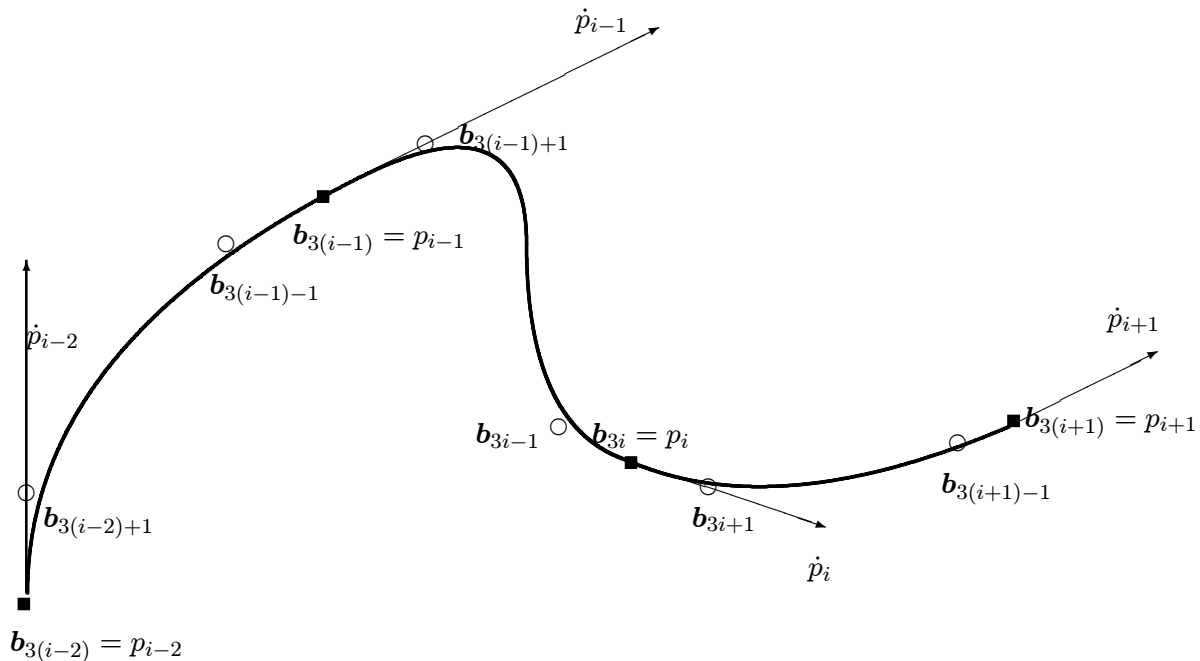


Figura 4.43 Interpolación de Hermite como curvas de Bézier.

continua, de la cual la interpolación de Hermite es un caso particular en la que  $\alpha = \frac{1}{3}$ . Otros coeficientes producen curvas interpolantes con propiedades diversas. Asimismo, es posible asignar un coeficiente distinto a cada punto de unión, lo cual equivale a modificar escalarmente la derivada en los puntos de control.

En los B-Splines cúbicos uniformes, las restricciones geométricas son mayores, dado que las curvas de Bézier subyacentes deben unirse con continuidad  $C^2$ . Esto se consigue haciendo que los puntos interiores  $b_{3i+1}$  y  $b_{3(i+1)-1}$  estén a un tercio y dos tercios (respectivamente) a lo largo del segmento  $\overline{p_i p_{i+1}}$ , y que los puntos de unión dependan de los interiores haciendo que  $b_{3i}$  esté a mitad de camino entre  $b_{3i-1}$  y  $b_{3i+1}$  y que  $b_{3(i+1)}$  esté a mitad de camino entre  $b_{3(i+1)-1}$  y  $b_{3(i+1)+1}$  (ver Figura 4.44). Esta construcción garantiza continuidad  $C^2$  dado que los segmentos  $\overline{b_{3(i-1)+1} b_{3i-1}}$  y  $\overline{b_{3i-1} b_{3i}}$ , y los segmentos  $\overline{b_{3i} b_{3i+1}}$  y  $\overline{b_{3i+1} b_{3(i+1)-1}}$  son proporcionales.

En los B-Splines cúbicos no uniformes, la equivalencia es similar. El único punto a tener en cuenta es que la parametrización es no uniforme, es decir, tenemos un

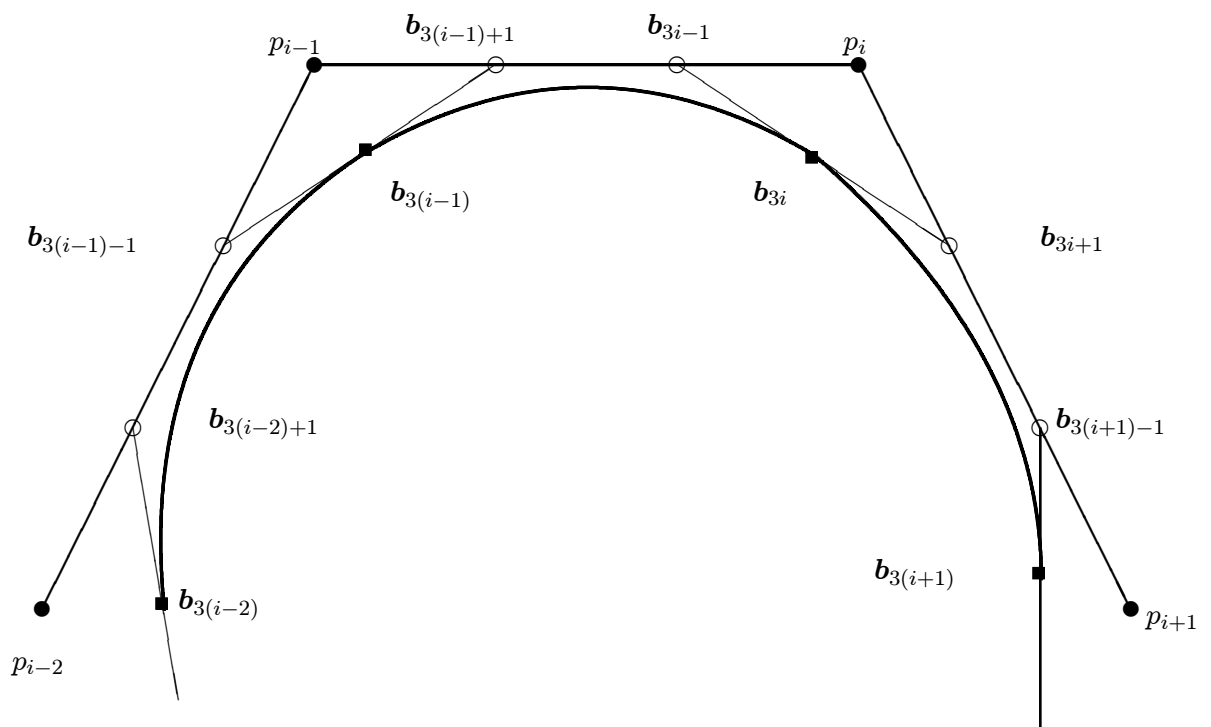


Figura 4.44 B-Splines cúbicos uniformes como curvas de Bézier.

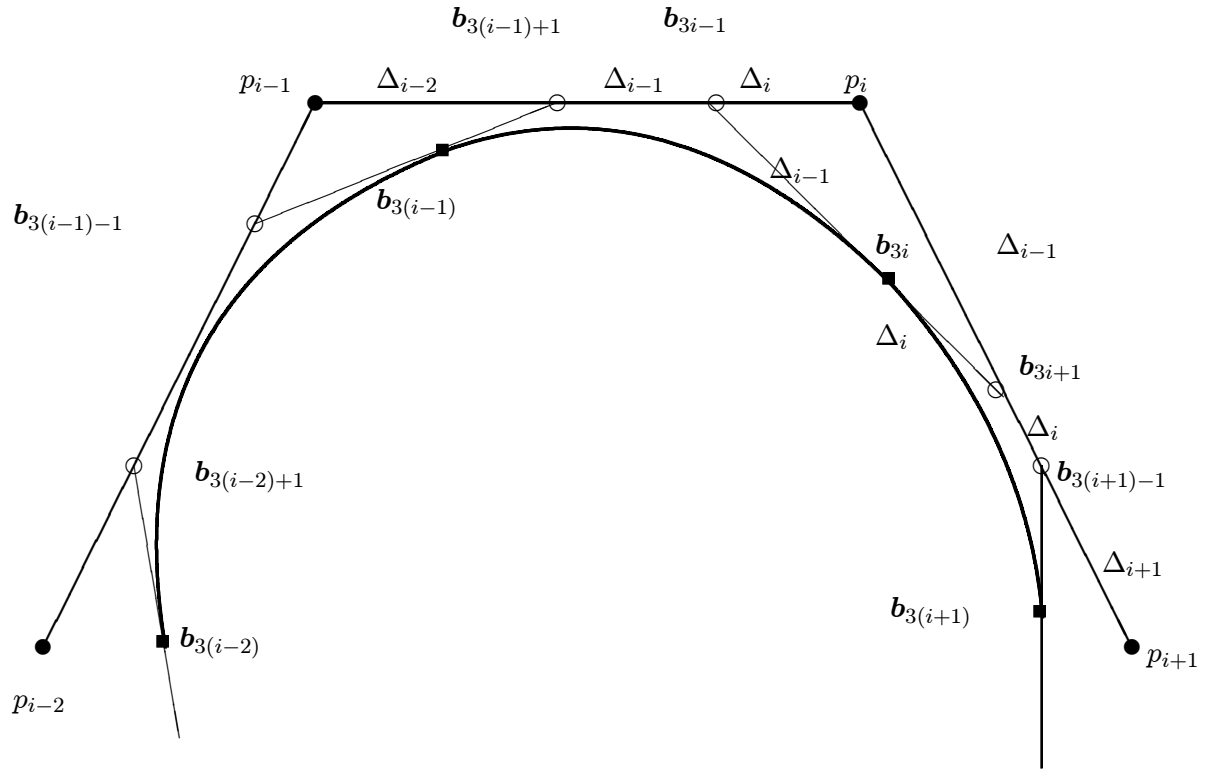


Figura 4.45 B-Splines cúbicos no uniformes como curvas de Bézier.

parámetro global  $u$  tal que la secuencia de valores  $t_0, t_1, \dots, t_k$  asociada a los nudos no cumple que  $\Delta_i = t_{i+1} - t_i = \Delta_j$  para  $i, j$  arbitrarios (ver Figura 4.45). Por lo tanto es necesario modificar el planteo geométrico anterior para tener en cuenta la parametrización no uniforme:

$$\begin{aligned}
 b_{3i-1} &= \frac{(\Delta_{i-2} + \Delta_{i-1})p_i + \Delta_i p_{i-1}}{\Delta_{i-2} + \Delta_{i-1} + \Delta_i}, \\
 b_{3i+1} &= \frac{(\Delta_{i-1} + \Delta_i)p_{i+1} + \Delta_{i+1} p_i}{\Delta_{i-1} + \Delta_i + \Delta_{i+1}}, \\
 b_{3i} &= \frac{\Delta_{i+1} b_{3i-1} + \Delta_i b_{3i+1}}{\Delta_{i+1} + \Delta_i}.
 \end{aligned} \tag{4.7}$$

### 4.5.2 Curvas de Bézier racionales

Si bien los resultados presentados en las dos Secciones anteriores solucionan satisfactoriamente el problema de aproximar interactivamente curvas de forma arbitraria a partir de puntos de control, existen todavía algunas limitaciones en los resultados obtenidos. Por ejemplo, no es posible encontrar un grafo de control razonable cuya aproximación produzca una circunferencia. Por dicha razón presentaremos brevemente algunos conceptos que permiten extender las posibilidades y dar mayor flexibilidad a las curvas de Bézier y B-Splines.

En muchas circunstancias de diseño es importante contar con algún dispositivo de representación que permita asignar distinta importancia a cada punto de control. Una forma de hacerlo fue (como vimos en la Sección 5.3) por medio de *repetir* puntos de control. Esta forma de trabajar es un tanto limitada, dado que la importancia de los puntos de control se representan con *enteros*, que determinan cuántas veces se repiten los mismos. Muchas veces, sin embargo, puede desearse un control más fino de la importancia. Además, el grado de una curva de Bézier con varios puntos múltiples se torna inmanejable.

Por dicha razón, se desarrollaron las curvas de Bézier *racionales* (en un momento veremos el por qué de este nombre), donde cada punto de control  $p_i$  tiene asignado un número real no negativo  $w_i$  que determina su *peso* en la curva. Los coeficientes de peso  $w_i$  son utilizados como parámetros de forma. Si se incrementa uno de ellos, una parte local de la curva es empujada hacia el punto  $p_i$  asociado, mientras que el resto de la curva se aleja del mismo. Este efecto es diferente al producido al modificar la posición del punto, dado que en dicho caso la curva se desplaza hacia una *dirección* uniforme (la dirección en que se desplazó el punto de control) en mayor o menor medida según sea la influencia del punto desplazado (ver Figura 4.46).

La descripción matemática de una curva de Bézier racional parte de tener en cuenta que la importancia de cada punto de control está en relación con su peso y con la sumatoria de todos los pesos de los puntos de control. Por lo tanto, la expresión para la curva de Bézier racional es

$$C(u) = \sum_{i=0}^n p_i \frac{w_i B_i^n(u)}{\sum_{i=0}^n w_i B_i^n(u)}.$$

Es fácil mostrar que la curva queda expresada en términos de las bases funcionales determinadas por cocientes de polinomios (razón por la cual se denominan racionales a estas curvas):

$$\frac{w_i B_i^n(u)}{\sum_{i=0}^n w_i B_i^n(u)},$$

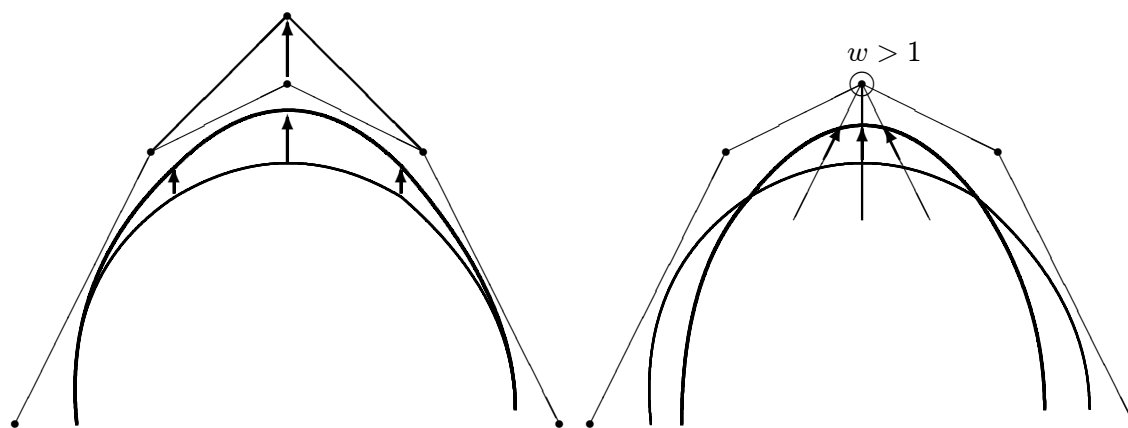
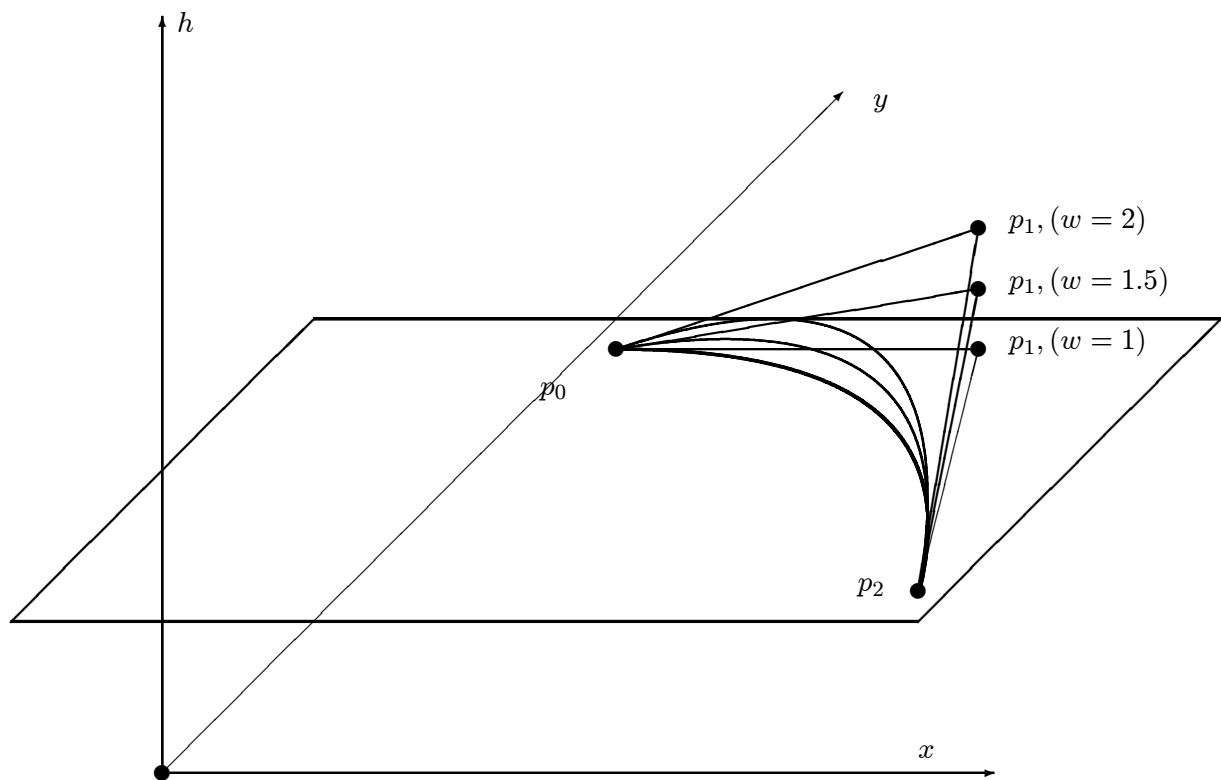


Figura 4.46 Modificar la posición vs. modificar el peso de un punto de control.

las cuales generalizan las bases de Bernstein para el caso racional. Estas bases son tales que su sumatoria es 1 en todos los casos. Por lo tanto generan combinaciones afines de puntos de control y por ello son invariantes frente a transformaciones afines. Si todos los pesos son no negativos, entonces la propiedad de armazón convexo sigue valiendo. También es posible ver que la curva comienza y termina en el primer y último punto de control. Sin embargo, estas curvas extienden enormemente las posibilidades de las curvas de Bézier no racionales, con un costo adicional bastante bajo.

Una forma de comprender los alcances de las curvas de Bézier racional, y de relacionar el tema con otros de la Computación Gráfica, proviene de interpretar el coeficiente  $w_i$  de peso de cada punto como su coordenada homogénea  $h_i$ . La razón de esta equivalencia es sencilla, dado que el pasaje del espacio homogéneo al espacio euclídeo se hace por medio de una proyección. Una curva 2DH (en la cual los puntos de control están en el espacio 2DH) es luego proyectada al plano  $h = 1$ .

Supongamos tener una parábola definida con tres puntos de control, y que incrementamos el valor de la coordenada homogénea del segundo punto. Las parábolas resultantes se inclinan según el plano que pasa por los tres puntos de control, pero la proyección de dichas parábolas sobre el plano  $h = 1$  es una colección de curvas que es empujada más y más hacia la proyección del segundo punto de control, a medida que el valor de su coordenada homogénea aumenta (ver Figura 4.47). Entonces el efecto de proyectar las parábolas con inclinación cada vez mayor es equivalente al de trazar curvas cada vez más tensionadas hacia la proyección del segundo punto de control.



**Figura 4.47** Interpretación proyectiva del peso de un punto de control en las curvas racionales.

```

procedure bezier_racional(var g:grafo_contr);
var i,j:integer;
    u,v:real;
    p:punto;
begin
  p:=g[0];
  graf_linea(p);
  for i:=1 to pasos do begin
    u:=i/pasos;
    p.x:=0; p.y:=0; p.w:=0;
    for j:=0 to pts_ctrol do begin
      v:=bernstein(u,j,pts_ctrol);
      p.x:=p.x + g[j].x*v*g[j].w;
      p.y:=p.y + g[j].y*v*g[j].w;
      p.w:=p.w + g[j].w*v;
    end;
    graf_linea(p);
  end;
end;

```

**Figura 4.48** Implementación proyectiva de las curvas de Bézier racionales.

Si las curvas resultantes son de segundo grado (porque las bases racionales siguen siendo de segundo grado) pero no son siempre parábolas (porque hay una única parábola definida entre tres puntos) ¿qué clase de curvas obtenemos cuando se modifica el peso del segundo punto de control? El lector atento notará que toda curva de segundo grado no degenerada es una cónica, y que por lo tanto las curvas resultantes deben ser arcos de elipse o de hipérbola. En particular, si el peso del segundo punto es mayor que uno, se puede mostrar (razonando a partir de la Figura 4.47) que la curva *se acerca* a dicho punto cuando  $u > 0$  y cuando  $u < 1$ , y por lo tanto en dicha circunstancia se acerca a los dos segmentos del grafo de control. Por lo tanto tiene dos asíntotas y luego es un segmento de hipérbola. Recíprocamente, si el peso en el segundo punto es menor que uno, entonces la curva tiende a acercarse a dicho punto cuando  $u < 0$  o cuando  $u > 1$ , de modo que en  $u = -\infty$  y en  $u = \infty$  la curva *se cierra*. Por lo tanto la curva no tiene asíntotas y es un arco de elipse.

De esa manera, la generalización de las curvas de Bézier no racionales a curvas racionales permite cubrir un importante grupo de problemas. Al mismo tiempo, ilustra de manera intuitiva las relaciones entre geometría proyectiva y analítica. Un algoritmo

para computar curvas de Bézier racionales se muestra en el trozo de código de la Figura 4.48, donde se utiliza el valor de la coordenada homogénea de los puntos del grafo de control como peso de dichos puntos (comparar con el algoritmo para curvas de Bézier por medio de las bases de Bernstein). Los resultados de modificar el peso en un punto de control se muestran en la Figura 4.49, donde es posible notar la versatilidad del resultado obtenido.

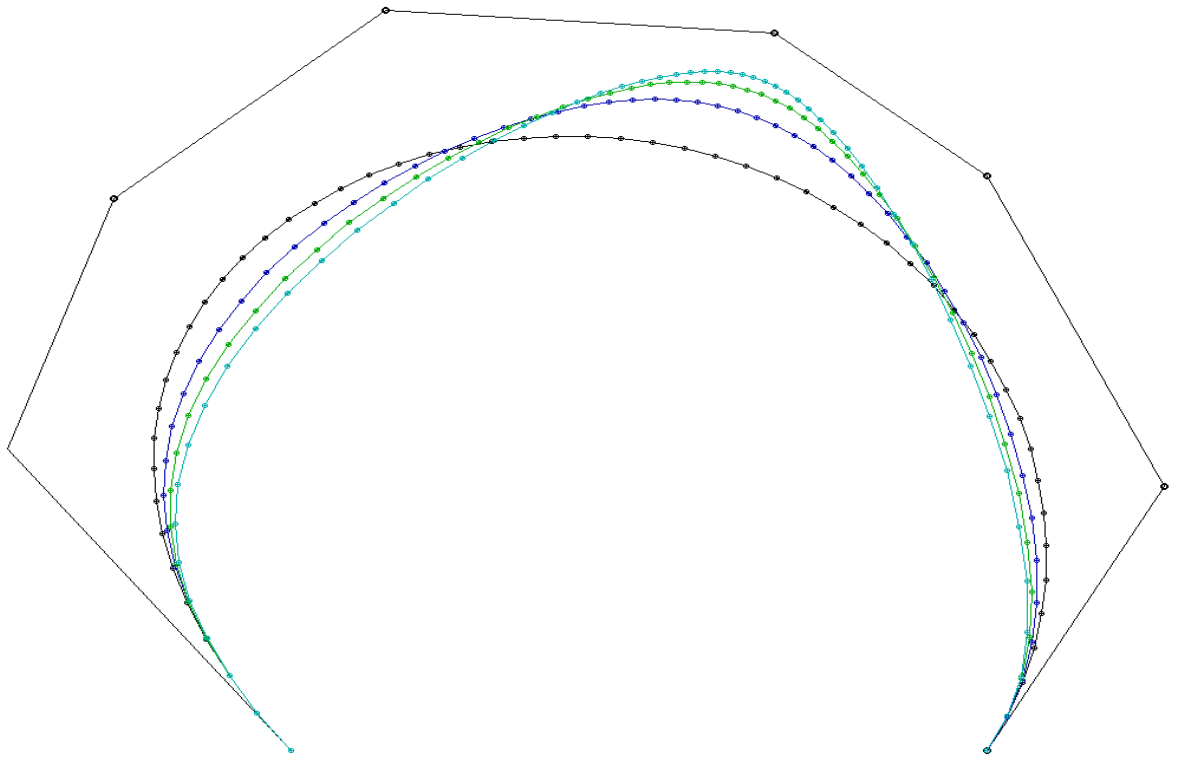


Figura 4.49 Curva de Bézier racional modificando el peso en un punto.

### 4.5.3 B-Splines cúbicos racionales no uniformes (NURBS)

Los B-Splines racionales no uniformes (NURBS) se han convertido en un estándar de descripción de curvas y superficies, tanto en Computación Gráfica como en CAD. La interpretación geométrica que realizamos para las curvas de Bézier racionales sigue siendo correcta en el caso de los B-Splines (uniformes o no) de cualquier grado. Por lo tanto, para obtener una curva B-Spline racional basta con modificar la coordenada homogénea de los puntos de control (originariamente 1) de modo de asignarles un peso mayor o menor.

De esa manera, la curva B-Spline racional (uniforme o no) de orden  $k$  par<sup>2</sup> que aproxima  $n + 1$  puntos queda expresada como

$$C(u) = \sum_{i=0}^{k+n-1} p_i \frac{w_i N_i^k(u)}{\sum_{i=0}^{k+n-1} w_i N_i^k(u)}.$$

De esa manera, las bases de Splines racionales son

$$\frac{w_i N_i^n(u)}{\sum_{i=0}^{k+n-1} w_i N_i^n(u)},$$

donde la base de Splines no racionales es la vista en la ecuación 4.6:

$$N_i^k(u) = \frac{u - t_i}{t_{i+k-1} - t_i} N_i^{k-1}(u) + \frac{t_{i+k} - u}{t_{i+k} - t_{i+1}} N_{i+1}^{k-1}(u).$$

Dicha ecuación se implementa computacionalmente combinando el algoritmo de B-Splines mostrado más arriba (ver Fig. 4.28) con la misma técnica de representar el peso con la coordenada homogénea de un punto (ver Figura 4.50).

El efecto de aumentar el peso de un punto en las bases se puede observar en la Figura 4.51, donde la parametrización es uniforme. La combinación de parametrizaciones no uniformes y pesos permite obtener una gran variedad de curvas NURBS a partir de un mismo grafo de control, aunque los resultados obtenidos no siempre reflejan intuitivamente el efecto de una parametrización con peso dada.

Al mismo tiempo, es posible representar una curva NURBS como compuesta por segmentos de curvas de Bézier racionales de grado  $k$ . Para ello es necesario encontrar, además, el peso de los puntos de Bézier de unión e interiores en función del peso de los puntos de control. En el caso cúbico, las ecuaciones 4.7 se transforman en

$$\begin{aligned} b_{3i-1} &= \frac{w_{i-1}\beta_i p_{i-1} + w_i(1-\beta_i)p_i}{w_{3i-1}}, \\ b_{3i+1} &= \frac{w_i(1-\alpha_{i+1})p_i + w_{i+1}\alpha_{i+1}p_{i+1}}{w_{3i+1}}, \end{aligned}$$

para los puntos interiores, cuyos pesos son

$$w_{3i-1} = w_{i-1}\beta_i + w_i(1-\beta_i),$$

$$w_{3i+1} = w_i(1-\alpha_i) + w_{i+1}\alpha_{i+1},$$

y las constantes auxiliares son

$$\alpha_i = \frac{\Delta_{i-2}}{\Delta_{i-2} + \Delta_{i-1} + \Delta_i},$$

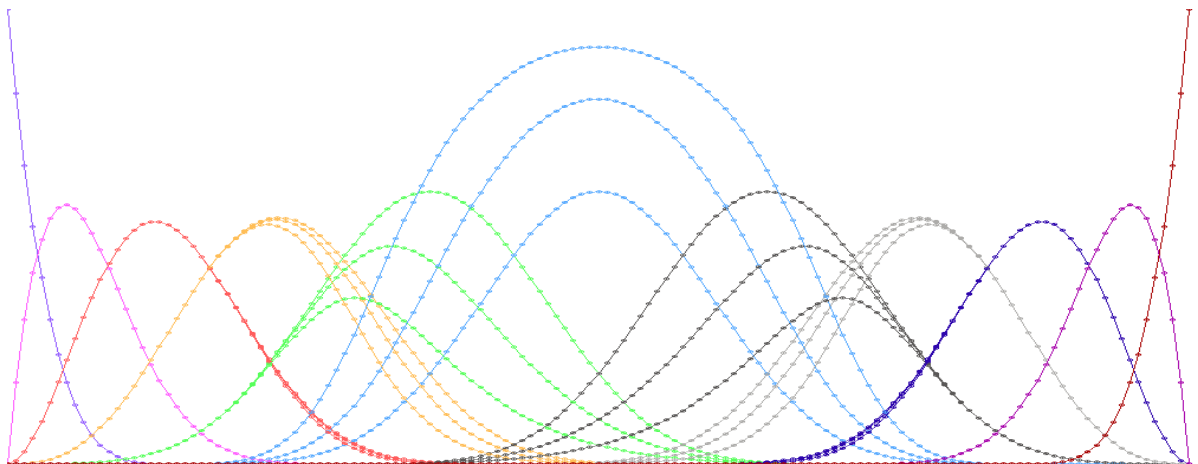
<sup>2</sup>Recordar que en estos casos, cada nudo de la curva está asociado a un punto de control.

```

procedure nurbs(var g: grafo_contr);
var i,j: integer;
    u,v: real;
    p: punto;
begin
  for i:=0 to pasos*(pts_ctrol-grado+2) do begin
    u:=(i/pasos);
    p.x:=0; p.y:=0; p.w:=0;
    for j:=0 to pts_ctrol do begin
      v:=base(u,j,grado)*g[j].w;
      p.x:=p.x + g[j].x*v;
      p.y:=p.y + g[j].y*v;
      p.w:=p.w + v;
    end;
    graf_linea(p);
  end;
end;

```

**Figura 4.50** Implementación de curvas NURBS como Splines no uniformes proyectivos.



**Figura 4.51** Bases NURBS modificando el peso en un punto.

$$\beta_i = \frac{\Delta_i}{\Delta_{i-2} + \Delta_{i-1} + \Delta_i}.$$

De manera similar se definen los puntos de Bézier de unión:

$$\mathbf{b}_{3i} = \frac{\gamma_i w_{3i-1} \mathbf{b}_{i-1} + (1 - \gamma_i) w_{3i+1} \mathbf{b}_{i+1}}{w_{3i}},$$

donde la constante auxiliar es

$$\gamma_i = \frac{\Delta_i}{\Delta_{i-1} + \Delta_i},$$

y el peso del punto de Bézier de unión es

$$w_{3i} = \gamma_i w_{3i-1} + (1 - \gamma_i) w_{3i+1}.$$

De esa forma, es posible computar los NURBS con la base de Bézier racional, la cual es menos compleja que la base recursiva de Splines no uniformes.

#### 4.5.4 $\beta$ -Splines

Es posible observar que el orden de continuidad paramétrica  $C^k$  (es decir,  $k$  veces continuamente derivable respecto del parámetro), es una medida “exagerada” del orden de continuidad de una curva implementada con segmentos polinomiales paramétricos. El interés de un orden de continuidad determinado proviene de la geometría de la curva resultante en la unión. Por lo tanto, es posible pensar que la verdadera medida es el orden de continuidad geométrica  $G^k$ , definido como el máximo orden para el cual una curva es continuamente derivable respecto de las direcciones de los ejes de un sistema de coordenadas arbitrario. Es decir, utilizamos la descripción de la curva como

$$\mathbf{C}(u) = \begin{cases} f_x(u) = c_x^3 u^3 + c_x^2 u^2 + c_x^1 u + c_x^0 \\ f_y(u) = c_y^3 u^3 + c_y^2 u^2 + c_y^1 u + c_y^0 \end{cases}$$

donde  $f_x(u)$  y  $f_y(u)$  representan respectivamente los valores de la posición geométrica de  $\mathbf{C}(u)$  para un valor dado del parámetro  $u$ .

¿Puede una curva ser discontinua en su derivada respecto de  $u$  pero no en su derivada respecto de  $x$  o  $y$ ? Por ejemplo, sean

$$f_x(u) = f_y(u) = \begin{cases} u & \text{si } 0 \leq u \leq 1 \\ u^2 & \text{si } u > 1 \end{cases}$$

La “curva” es la recta identidad. Pero en  $u = 1$  existe una discontinuidad en su derivada paramétrica, la cual no es posible encontrar en su derivada geométrica (por mucho que se la analice).

Por lo tanto, los requisitos generales planteados al comienzo de este Capítulo se refieren, en realidad, a que las curvas y superficies que se buscan deben ser, como mínimo,  $G^2$  continuas (en vez de  $C^2$  continuas). Esto nos da mayor flexibilidad a la hora de plantear una base para resolver un problema de aproximación o interpolación. Geométricamente, dos curvas se unen con continuidad  $G^1$  si sus vectores derivada en el punto de unión tienen la misma *dirección* en ambas curvas en el punto de la unión. La magnitud de estas derivadas (es decir la “velocidad” con que cada curva recorre el punto de la unión) no afecta la continuidad geométrica. Por lo tanto, es posible unir con continuidad  $G^1$  dos curvas en un punto, si sus derivadas en dicho punto son paralelas, es decir, si la curva es instantáneamente acelerada o frenada en dirección de la derivada (excluyendo, por supuesto, el caso en que alguna de las dos derivadas se haga cero).

Lo mismo sucede con la segunda derivada. La interpretación geométrica de la segunda derivada de una curva en un punto es, precisamente, su *curvatura*. Intuitivamente, sin alterar la continuidad  $G^2$ , la curvatura puede modificarse instantáneamente en un punto *solo en la dirección de la tangente*. De hecho, cuando se modifica instantáneamente la velocidad de la curva en dirección de la tangente, también se está modificando su curvatura.

En la formulación original, los coeficientes de modificación instantánea de velocidad y curvatura se denominan  $\beta_1$  y  $\beta_2$ , respectivamente. De esa manera, si en la unión entre dos segmentos de curva polinomial ocurre un salto instantáneo en la derivada, queda expresado como

$$\dot{C}_i(0) = \beta_1 \dot{C}_{i-1}(1),$$

lo cual también produce un cambio instantáneo en la curvatura

$$\ddot{C}_i(0) = \beta_1^2 \ddot{C}_{i-1}(1)$$

(porque el radio de curvatura es proporcional a la velocidad al cuadrado). Si además deseamos modificar instantáneamente la curvatura en dirección de la tangente, vemos que

$$\ddot{C}_i(0) = \beta_1^2 \dot{C}_{i-1}(1) + \beta_2 \ddot{C}_{i-1}(1).$$

Las únicas restricciones son que  $\beta_1$  no puede ser cero, mientras que  $\beta_2$  no puede ser negativo.

Esto nos lleva a repetir el proceso de plantear las bases de Splines  $G^2$  continuos de una manera similar a los Splines  $C^2$  continuos, es decir, a través de cuatro sub-bases.

```

function base(u,b1,b2:real;i:integer):double;
var d:double;
begin
  d:=1/(2*b1*(2+b1*(2+b1))+b2+2);
  case i of
    1 : base:=u*u*u*2*d;
    0 : base:=d*(2+u*(6*b1+u*(3*b2+6*b1*b1-u*2*(b2+b1*(b1+1)+1))));
    -1 : base:=d*(b2+4*b1*(b1+1) + u*(6*b1*(b1*b1-1))
      + u*u*(-3*(b2+2*b1*b1*(b1+1)) + u*2*(b2+b1*(b1*(b1+1)+1))));
    -2 : base:=d*2*b1*b1*b1*(1+u*(-3+u*(3-u)));
  else writeln('error')
  end;
end;

```

Figura 4.52 Bases funcionales de  $\beta$ -Splines.

En este caso, las restricciones de continuidad son

$$\begin{array}{lll}
 0 = b_{i+1}(0) & 0 = \dot{b}_{i+1}(0) & 0 = \ddot{b}_{i+1}(0), \\
 b_{i+1}(1) = b_i(0) & \beta_1 \dot{b}_{i+1}(1) = \dot{b}_i(0) & \beta_1^2 \ddot{b}_{i+1}(1) + \beta_2 \dot{b}_{i+1}(1) = \ddot{b}_i(0), \\
 b_i(1) = b_{i-1}(0) & \beta_1 \dot{b}_i(1) = \dot{b}_{i-1}(0) & \beta_1^2 \ddot{b}_i(1) + \beta_2 \dot{b}_i(1) = \ddot{b}_{i-1}(0), \\
 b_{i-1}(1) = b_{i-2}(0) & \beta_1 \dot{b}_{i-1}(1) = \dot{b}_{i-2}(0) & \beta_1^2 \ddot{b}_{i-1}(1) + \beta_2 \dot{b}_{i-1}(1) = \ddot{b}_{i-2}(0), \\
 b_{i-2}(1) = 0 & \beta_1 \dot{b}_{i-2}(1) = 0 & \beta_1^2 \ddot{b}_{i-2}(1) + \beta_2 \dot{b}_{i-2}(1) = 0,
 \end{array}$$

las cuales, junto con

$$b_{i+1}(u) + b_i(u) + b_{i-1}(u) + b_{i-2}(u) = 1$$

nos permiten obtener

$$\begin{aligned}
 b_{i+1}(u) &= \frac{1}{\delta}(2u^3), \\
 b_i(u) &= \frac{1}{\delta}[2 + 6\beta_1 u + (3\beta_2 + 6\beta_1^2)u^2 - (2\beta_2 + 2\beta_1^2 + 2\beta_1 + 2)u^3], \\
 b_{i-1}(u) &= \frac{1}{\delta}[(\beta_2 + 4\beta_1^2 + 4\beta_1) + 6(\beta_1^3 - \beta_1)u - 3(\beta_2 + 2\beta_1^3 + 2\beta_1^2)u^2 + \\
 &\quad + 2(\beta_2 + \beta_1^3 + \beta_1^2 + \beta_1)u^3], \\
 b_{i-2}(u) &= \frac{1}{\delta}2[\beta_1^3 - 3\beta_1^3 u + 3\beta_1^3 u^2 - \beta_1^3 u^3],
 \end{aligned}$$

con

$$\delta = 2\beta_1^3 + 4\beta_1^2 + 4\beta_1 + \beta_2 + 2 \neq 0.$$

```

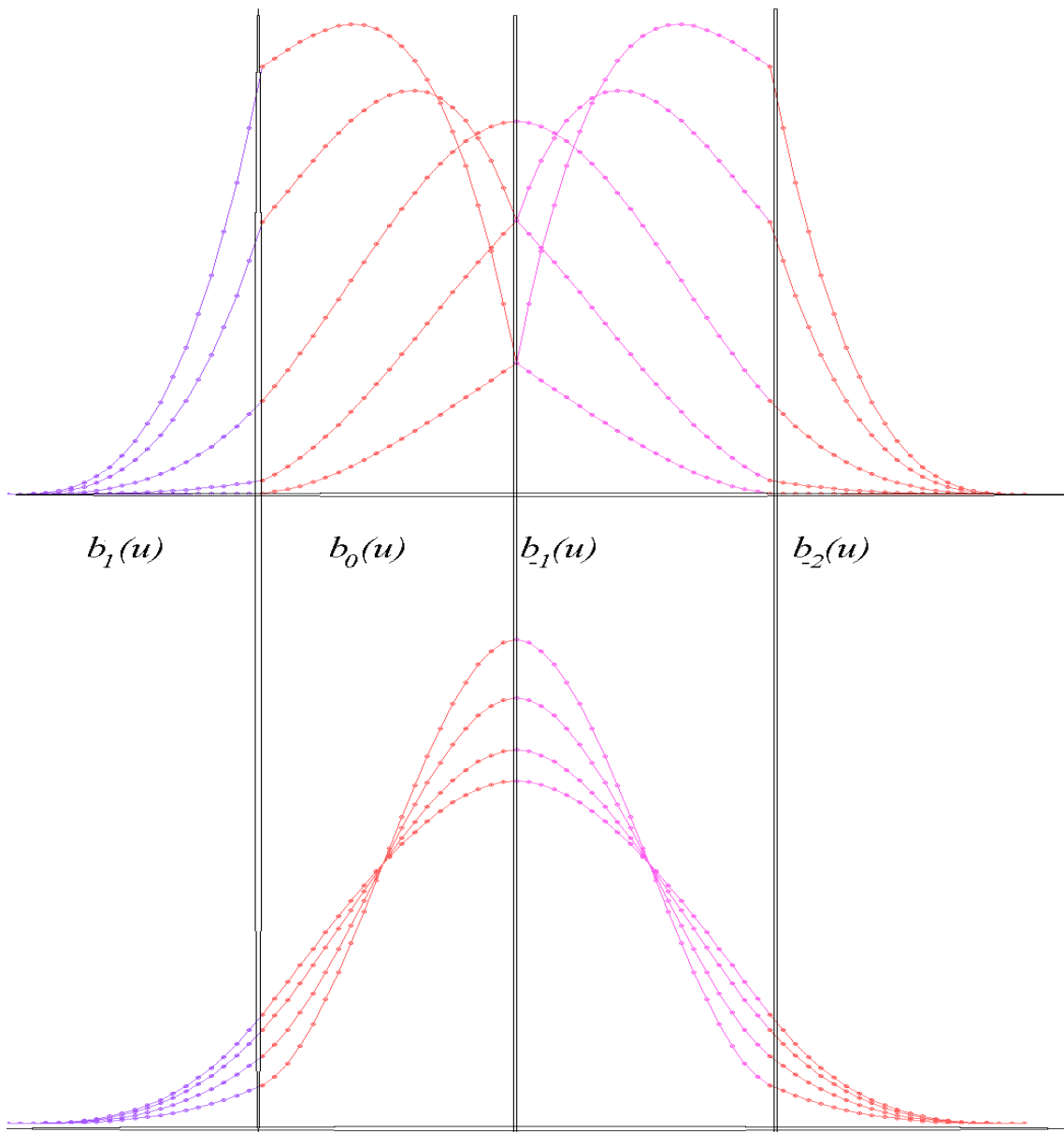
procedure beta_splines(var g:grafo_contr; c:integer);
var i,j,k,index:integer;
    u,v:real;
    p:punto;
begin
    p.z:=1;           p.w:=1;
    b1:=1.5;           b2:=3;           {valores arbitrarios}
    for j:=1 to pts_ctrol do begin
        p.x:=0;        p.y:=0;
        for k:=0 to pasos do begin
            u:=k/pasos;
            p.x:=0;      p.y:=0;
            for i:=-2 to 1 do begin
                v:=base(u,b1,b2,i);
                index:=i+j;
                if index<0 then index:=0           {repite primer y ultimo puntos}
                else if index>pts_ctrol then index:= pts_ctrol;
                p.x:=p.x+g[index].x*v;
                p.y:=p.y+g[index].y*v;
            end;
            graf_linea(p);
        end;
    end;
end;

```

**Figura 4.53** Implementación de curvas  $\beta$ -Splines.

Podemos ver en los trozos de código de la Figura 4.52 la implementación de las bases de  $\beta$ -Splines, factorizando los polinomios para agilizar la implementación, y en la Figura 4.53 los detalles de implementación de curvas  $\beta$ -Splines, mientras que en la Figura 4.54 se observan las bases resultantes al modificar independientemente el valor de  $\beta_1$  y  $\beta_2$ .

Debemos notar que si sustituimos  $\beta_1 = 1$  y  $\beta_2 = 0$  estamos en el caso de los B-Splines (tanto en restricciones como en segmentos base). Estos nuevos parámetros son grados de libertad que tiene la definición de la base, es decir, tenemos una familia biparamétrica de bases.



**Figura 4.54** Bases de  $\beta$ -Splines. La parte superior muestra valores de sesgo menores o mayores de 1 con tensión 0, mientras que la parte inferior muestra valores de tensión mayores de 0 con sesgo 1.

En realidad, toda base que particione la unidad en funciones no negativas con soporte local puede ser una base de Splines, y la continuidad de sus sub-bases define

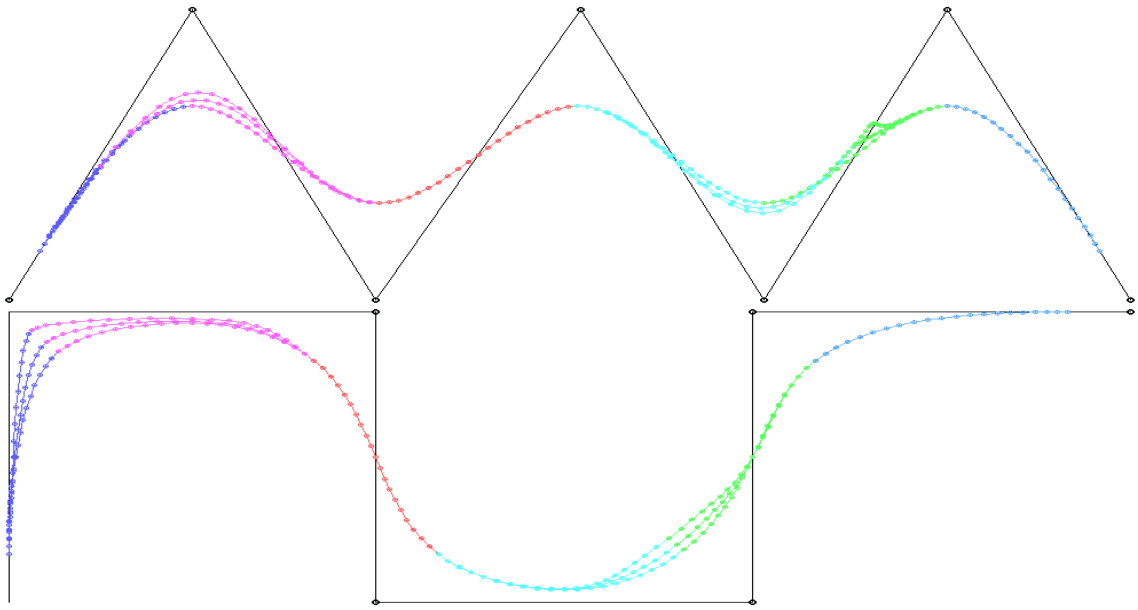


Figura 4.55 Curva  $\beta$ -Splines modificando el sesgo en el segundo y quinto punto.

el orden de continuidad resultante. En el caso de los  $\beta$ -Splines, los parámetros  $\beta_1$  y  $\beta_2$  se denominan parámetros de *sesgo* y *tensión*, respectivamente, dado que  $\beta_1$  permite sesgar la curva hacia izquierda o derecha del grafo de control (según sea mayor o menor que uno), mientras que  $\beta_2$  permite tensarla hacia el grafo de control (ver Figuras 4.55 y 4.57).

Lo importante para la utilidad y versatilidad de este método no es exactamente que la base sea biparamétrica, sino que los parámetros de forma pueden variar en cada punto de control (como en el caso de los pesos en las curvas racionales). Dados dos puntos de control vecinos  $p_i$  y  $p_{i+1}$  es posible encontrar la siguiente función

$$\beta_1(u) = \beta_{1,i} + (\beta_{1,i+1} - \beta_{1,i})[10u^3 - 15u^4 + 6u^5],$$

la cual encuentra el valor de  $\beta_1$  en un punto dado. Lo mismo se hace en  $\beta_2$  y  $\beta_2$  para todo segmento de la curva. De esa manera, tenemos curvas con *control local* en sus parámetros de forma. El trozo de programa de la Figura 4.56 muestra una implementación de estas curvas, asumiendo que los valores de sesgo y tensión de cada punto forman parte del registro asociado.

La utilidad de estas curvas es proveer un mecanismo más intuitivo para el control de la forma de las aproximaciones que lo que se puede obtener con NURBS. De

```

type punto = record x,y,z,w,bt1,bt2:real
                end;

procedure beta_splines(var g:grafo_contr; c:integer);
var i,j,k,index,j2:integer;
    u,v,b1,b2,w:real;
    p:punto;
begin
  for j:=1 to pts_ctrol do begin
    p.z:=1;                p.w:=1;
    for k:=0 to pasos do begin
      u:=k/pasos;
      p.x:=0;                p.y:=0;
      for i:=-2 to 1 do begin
        index:=i+j;
        if index<0 then index:=0
          else if index>pts_ctrol then index:= pts_ctrol;
        if j=pts_ctrol then j2:= pts_ctrol else j2:=j+1;
        w:=u*u*u*(10+u*(-15+6*u));
        b1:=g[j].bt1+(g[j2].bt1-g[j].bt1)*w;
        b2:=g[j].bt2+(g[j2].bt2-g[j].bt2)*w;
        v:=base(u,b1,b2,i);
        p.x:=p.x+g[index].x*v;
        p.y:=p.y+g[index].y*v;
      end;
      graf_linea(p,c+j);
    end;
  end;
end;

```

**Figura 4.56** Implementación de curvas con  $\beta$  local.

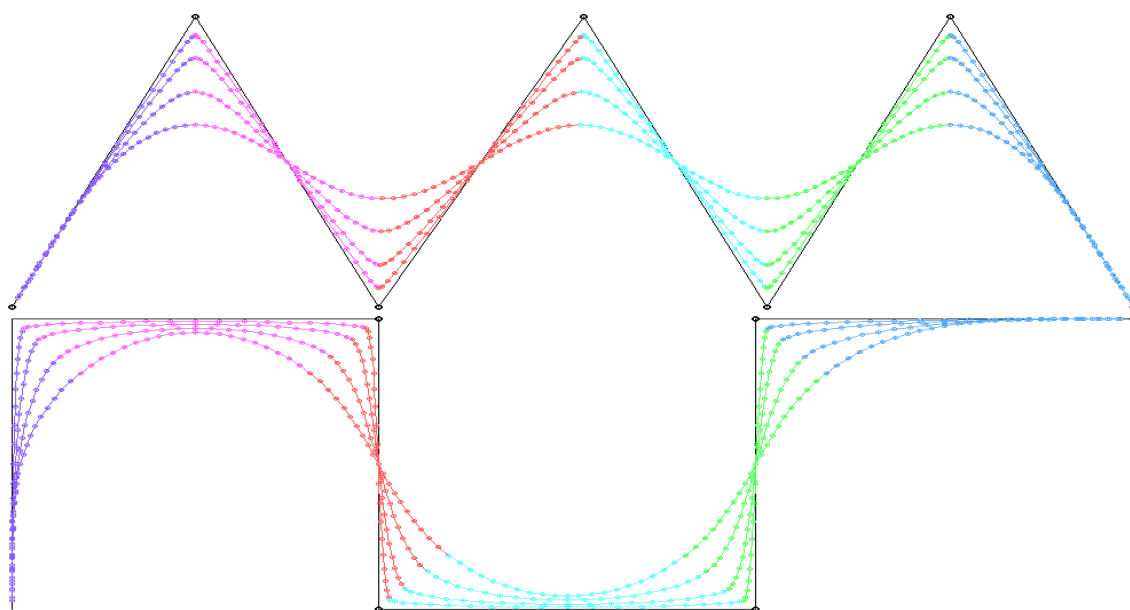


Figura 4.57 Curva  $\beta$ -Splines modificando la tensión.

todas maneras, no es posible encontrar una caracterización geométrica elegante para los parámetros de sesgo y tensión, como hicimos con el peso en las curvas racionales. Esto significa que las curvas resultantes dejan de pertenecer a la órbita familiar de la geometría proyectiva o de la geometría analítica. Entre otras consecuencias, no es posible encontrar una representación equivalente con curvas de Bézier con puntos de unión e interiores.

## 4.6 Ejercicios

1. Implementar los algoritmos de aproximación e interpolación de curvas de Hermite, Bézier (por de Casteljau y por Bernstein) y B-Spline cúbicos uniformes.
2. Aplicar los métodos a un grupo de grafos de control de prueba (por ejemplo, una “onda cuadrada”, una curva cerrada o para aproximar un mapa de la Provincia). Comparar los resultados obtenidos y el tiempo insumido, y encontrar ventajas y desventajas de cada uno.
3. Dibuje su propia firma. ¿Qué método utilizó? ¿Por qué? ¿Cuántos puntos de control fueron necesarios y cuánto tiempo de trabajo requirió encontrarlos? ¿Qué se concluye?
4. Implementar los B-Splines cúbicos uniformes por medio de curvas de Bézier cúbicas. Comparar resultados y tiempos con la implementación de B-Splines propiamente dichos.
5. Implementar B-Splines cúbicos no uniformes por medio de curvas de Bézier. Dado un mismo grafo de control, asignar parametrizaciones no uniformes y comparar los resultados.
6. Implementar Bézier racionales y NURBS. Aplicar a los mismos problemas que en los ejercicios 2 y 3. Comentar los resultados.

## 4.7 Bibliografía recomendada

El tratamiento de este tema es un poco superficial en los textos clásicos de Computación Gráfica. Al mismo tiempo, el gran desarrollo que ha tenido hace que los resultados recientes figuren casi exclusivamente en artículos en revistas o en libros específicos del tema. De todas maneras, existen libros dedicados exclusivamente a curvas y superficies para CAGD que comienzan desde un nivel adecuadamente introductorio y presentan todo lo necesario para conocer el material clásico sobre curvas y superficies de Bézier y B-Splines. La recomendación básica es el libro de Farin [30], el cual es exhaustivo y presenta los temas de relevancia de una manera clara y definitiva. Otro libro recomendable es el de Bartels, Barski y Beatty [6], que sin ser tan riguroso y preciso presenta algunos temas avanzados desarrollados por los autores, y que no es posible encontrar en otros textos.

Un buen material introductorio al tema de las curvas racionales puede leerse en [70], donde se presenta la interpretación proyectiva de las curvas de Bézier racionales y se deriva el sistema de coordenadas baricéntrico asociado a las bases de Bernstein racionales. El mismo autor presenta en [71] una introducción a las curvas y superficies NURBS.

Los  $\beta$ -Splines son un desarrollo relativamente reciente. La discusión acerca de la continuidad paramétrica *versus* la continuidad geométrica está presentada en [5]. Los  $\beta$ -Splines son comparados con los métodos clásicos en [3], y la derivación de la ecuación del control local para los parámetros de forma está en [4]. Recomendamos también a los interesados que consulten [26].

