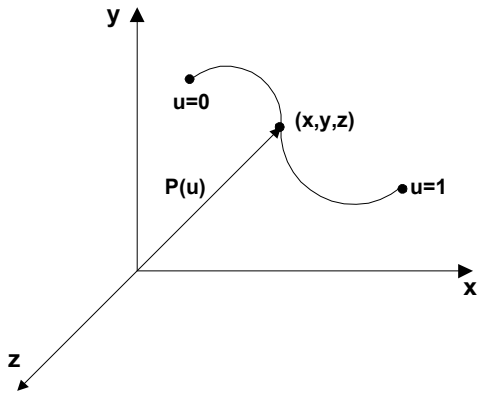
CAPITULO 6

Curvas y Superficies

6.1 Representacion Paramétrica

Representación paramétrica:



Sea:

$$x(u) = a_{3x}u^3 + a_{2x}u^2 + a_{1x}u + a_{0x}$$
$$y(u) = a_{3y}u^3 + a_{2y}u^2 + a_{1y}u + a_{0y}$$
$$z(u) = a_{3z}u^3 + a_{2z}u^2 + a_{1z}u + a_{0z}$$

$$\hat{P}(u) = \hat{P}\big(x(u), y(u), z(u)\big) = \hat{a}_3u^3 + \hat{a}_2u^2 + \hat{a}_1u + \hat{a}_0$$
$$u \in [0,1]$$

$$\hat{P}(u) = \left[\begin{array}{c|c|c|c} a_{3x} & a_{2x} & a_{1x} & a_{0x} \\ a_{3y} & a_{2y} & a_{1y} & a_{0y} \\ a_{3z} & a_{2z} & a_{1z} & a_{0z} \end{array}\right] \begin{bmatrix} u^3 \\ u^2 \\ u^1 \\ 1 \end{bmatrix}$$

a_3, a_2, a_1 y a_0 se llaman coeficientes algebraicos.

$$P(u) = a3u3 + a2u2 + a1u + a0 \qquad [1]$$

para $u = 0, \quad u = 1, \quad P^u(0)$ se tiene

$$P(0) = a_0$$
$$P(1) = a_3 + a_2 + a_1 + a_0$$
$$P^u(0) = a_1$$
$$P^u(1) = 3a_3 + 2a_2 + 1a_1$$

Resolviendo para a_0, a_1, a_2, a_3 se tiene:

$$a_0 = P(0)$$
$$a_1 = P^u(0)$$
$$a_2 = -3P(0) + 3P(1) - 2P^u(0) - P^u(1)$$
$$a_3 = 2P(0) - 2P(1) + P^u(0) + P^u(1)$$

Reemplazando en (1) y arreglando factores, se tiene:

$$P(u) = \big(2u^3 - 3u^2 + 1\big)P(0) + \big(-2u^3 + 3u^2\big)P(1) + \big(u^3 - 2u^2 + u\big)P^u(0) + \big(u^3 - u^2\big)P^u(1)$$

Llamemos:

$$F_1(u) = 2u^3 - 3u^2 + 1$$

$$F_2(u) = -2u^3 + 3u^2$$
$$F_3(u) = u^3 - 2u^2 + u$$
$$F_4(u) = u^3 - u^2$$

Luego:

$$P(u) = F_1P(0) + F_2P(1) + F_3P^u(0) + F_4P^u(1)$$

P(0), P(1), $P^u(0)$, $P^u(1)$ se llaman coeficiente geométricos.

En forma Matricial

$$P = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} \begin{bmatrix} a_3 & a_2 & a_1 & a_0 \end{bmatrix}^T$$

Sea :

$$U = \begin{bmatrix} u3 & u2 & u1 & 1 \end{bmatrix}$$
$$A = \begin{bmatrix} a_3 & a_2 & a_1 & a_0 \end{bmatrix}$$

Por otro lado:

$$P = \begin{bmatrix} F_1 & F_2 & F_3 & F_4 \end{bmatrix} \begin{bmatrix} P_0 & P_1 & P_o^u & P_1^u \end{bmatrix}^T$$
$$P = FB$$

$$F = \begin{bmatrix} (2u^3 - 3u^2 + 1) & (-2u^3 + 3u^2) & (u^3 - 2u^2 + u) & (u^3 - u^2) \end{bmatrix}$$

$$F = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} \underbrace{\begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}}_{=M}$$

$$F = UM$$
$$P = UMB \quad \text{para} \quad P = UA$$

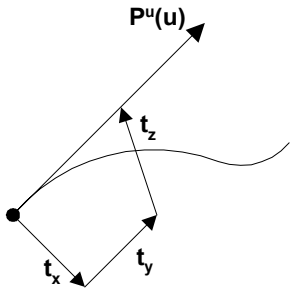
Luego

$$A = MB$$
$$P(u) = UMB \qquad u \in [0,1]$$

Las curvas anteriores definidas por los puntos de comienzo y finales, además de sus pendientes, se llaman curvas de Hermite.

6.2 Vectores Tangentes

Otra forma de escribir una tangente a una curva:



Donde t_x, t_y, t_z son los cosenos directores, tales que:

$$|t| = \sqrt{t_x^2 + t_y^2 + t_z^2} = 1$$

Se define:

$$k = \left|P^u\right|$$

luego $P^u = k * t$

Así los grados de libertad de la curva están dados por:

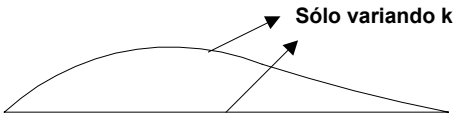
- 6 por los puntos P_0 y P_1.
- 4 por los cosenos directores de ambas tangentes.
- 2 por las magnitudes de los vectores tangentes.

En total 12 grados de libertad.

$$B = \begin{bmatrix} P_0 & P_1 & P_0^u & P_1^u \end{bmatrix}^T$$

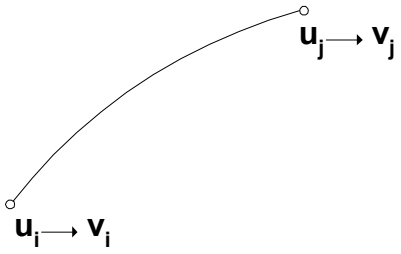
Se puede escribir como:

$$B = \begin{bmatrix} P_0 & P_1 & k_0 t_0 & k_1 t_1 \end{bmatrix}^T$$



6.3 Reparametrización

Algunas veces se requiere cambiar los parámetros de una curva, sin variar la posición ni la forma de ésta. Veamos el caso general:



$$B = \begin{bmatrix} P_i & P_j & P_i^u & P_j^u \end{bmatrix}^T \quad \rightarrow \quad B = \begin{bmatrix} q_i & q_j & q_i^v & q_j^v \end{bmatrix}^T$$

Como no debe haber cambio de posición, se tiene que cumplir:

$$q_i = P_i$$
$$q_j = P_j$$

¿Qué pasa con las pendientes?

La relación que existe entre *u* y *v*, esto es *v=f(u)*, debe ser de tales características que no altérela forma de la curva. Esta se cumple si hay una relación lineal entre ello, entonces podemos suponer que:

$$v = au + b$$

Así

$$dv = a\,du$$
$$v_i = au_i + b$$
$$v_j = au_j + b$$
$$v_j - v_i = a(u_j - u_i)$$
$$a = \frac{v_j - v_i}{u_j - u_i}$$

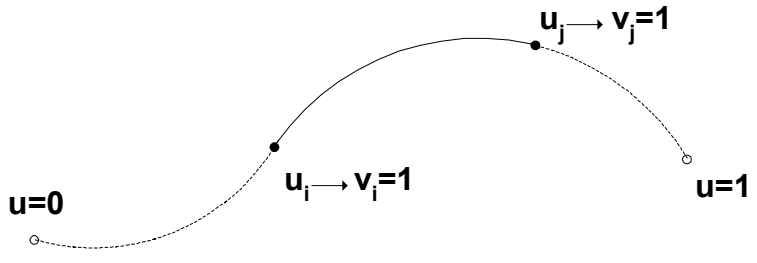
$$\frac{du}{dv} = \frac{1}{a} \quad \Rightarrow \quad P^v = \frac{1}{a} P^u$$

Luego:

$$q_i^v = \frac{u_j - u_i}{v_j - v_i} P_i^u$$

$$q_j^v = \frac{u_j - u_i}{v_j - v_i} P_j^u$$

6.4 Forma Truncada



Usemos las fórmulas anteriores

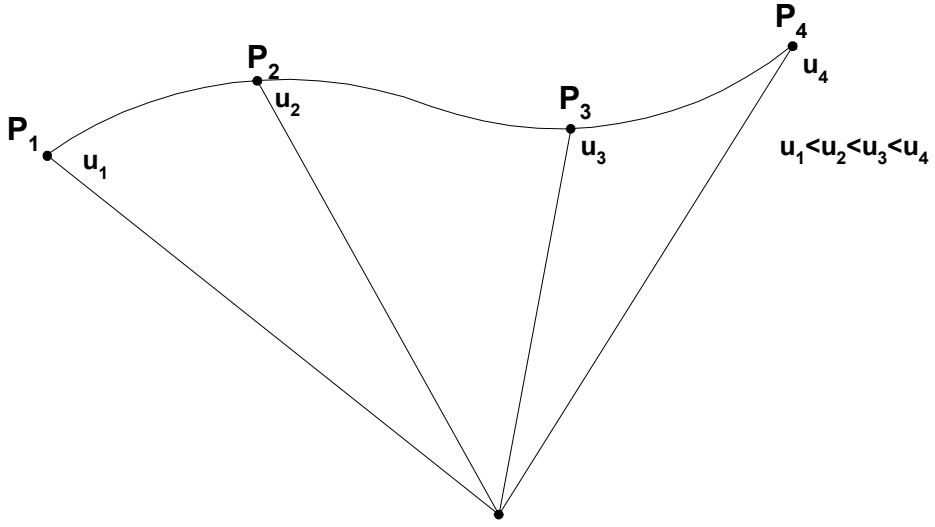
$$q_i = P_i \Rightarrow \quad q_0 = P_i$$
$$q_j = P_j \Rightarrow \quad q_1 = P_j$$

$$q_i^v = \frac{u_j - u_i}{v_j - v_i} P_i^u = \left(u_j - u_i\right) P_i^u$$

$$q_0^v = \left(u_j - u_i\right) P_i^v$$

$$q_j^v = \left(u_j - u_i\right) P_j^v \Rightarrow \quad q_1^v = \left(u_j - u_i\right) P_j^v$$

6.5 Forma de Cuatro Puntos



Supongamos una matriz *k*, tal que:

$$\begin{bmatrix} q_0 & q_1 & q_0^u & q_1^u \end{bmatrix}^T = k \begin{bmatrix} P_1 & P_2 & P_3 & P_4 \end{bmatrix}^T \qquad (1)$$

$$P(u) = UMB$$

Luego:

$$\begin{bmatrix} P_1 & P_2 & P_3 & P_4 \end{bmatrix}^T = \begin{bmatrix} U_1 & U_2 & U_3 & U_4 \end{bmatrix}^T MB$$

De aquí

$$B = M^{-1}\begin{bmatrix} U_1 & U_2 & U_3 & U_4 \end{bmatrix}^{T^{-1}} \begin{bmatrix} P_1 & P_2 & P_3 & P_4 \end{bmatrix}^T$$

De (1) se tiene que:

$$k = M^{-1}\begin{bmatrix} U_1 & U_2 & U_3 & U_4 \end{bmatrix}^{T^{-1}}$$
$$B = k\begin{bmatrix} P_1 & P_2 & P_3 & P_4 \end{bmatrix}^T$$

De donde:

$$\begin{bmatrix} P_1 & P_2 & P_3 & P_4 \end{bmatrix} = k^{-1}B$$

Para *u=0, u=1/2, u=2/3, u=1,* se obtiene:

$$k = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ -11/2 & 9 & -9/2 & 1 \\ -1 & 9/2 & -9 & 11/2 \end{bmatrix}$$

$$k = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 20/27 & 7/27 & 4/27 & -2/27 \\ 7/27 & 20/27 & 2/27 & -4/27 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

Sea

$$\overline{P} = \begin{bmatrix} P_1 & P_2 & P_3 & P_4 \end{bmatrix}^T$$

$$P = UMB \qquad B = k\overline{P}$$
$$P = UMK\overline{P} \qquad con\ Mk = N$$

$$P = UN\overline{P}$$

$$N = \begin{bmatrix} -9/2 & 27/2 & -27/2 & 9/2 \\ 9 & -45/2 & 18 & -9/2 \\ -11/2 & 9 & -9/2 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

Expandiendo se llega a

$$P = \left(-4{,}5u^3 + 9u^2 - 5{,}5u + 1\right)P_1 + \left(13{,}5u^3 - 22{,}5u^2 + 9u\right)P_2 + \left(13{,}5u^3 + 18u^2 - 4{,}5u\right)P_3 + \left(4{,}5u^3 - 4{,}5u^2 + u\right)P_4$$

6.6 Curvas de Bezier

En general, una curva de Bezier se puede aproximar a cualquier número de fuentes de control. El número de fuentes de control determina el grado del polinomio. La curva de Bezier es más fácilmente determinada por funciones"Blending".

Supongamos que tenemos *n+1* posiciones con puntos de control: $P_k = (x_k, y_k, z_k)$ con *k* variando entre 0 y *n*.

Se puede describir una curva que pasa por los puntos P_0 y P_n, y se aproxima a los puntos intermedios:

$$P(u) = \sum_{k=0}^{n} P_k \, BEZ_{k,n}(u) \qquad u \in [0,1]$$

$BEZ_{k,n}(u)$ son los polinomios de Bernstein

$$BEZ_{k,n}(u) = C(n,k)\,u^k\,(1-u)^{n-k}$$

Donde:

$$C(n,k) = \frac{n!}{k!(n-k)!}$$

En forma sucesiva se puede definir:

$$BEZ_{k,n}(u) = (1-u)\,BEZ_{k,n}(u) + u\,BEZ_{k-1,n-1}(u) \qquad n > k \geq 1$$

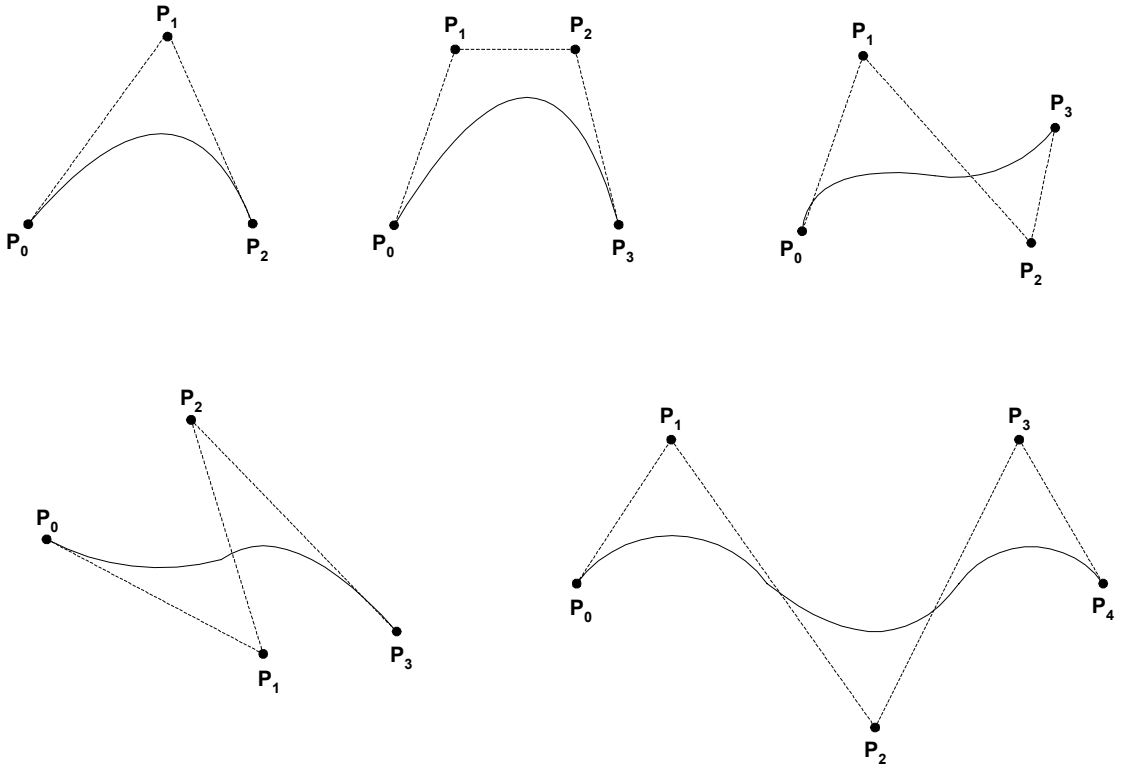
P(u) representa una ecuación paramétrica como:

$$x(u) = \sum_{k=0}^{n} xk\ BEZ_{k,n}(u)$$

$$y(u) = \sum_{k=0}^{n} yk\ BEZ_{k,n}(u)$$

$$z(u) = \sum_{k=0}^{n} zk\ BEZ_{k,n}(u)$$

Como regla, una curva de bezier es un polinomio de grado uno menos que el número de puntos de control.



6.6.1 *Propiedades de curvas de Bezier*

Una propiedad útil es que la curva siempre pasa por el primer y último punto de control.

Condiciones de borde:

$$P(0) = P_0$$
$$P(1) = P_n$$

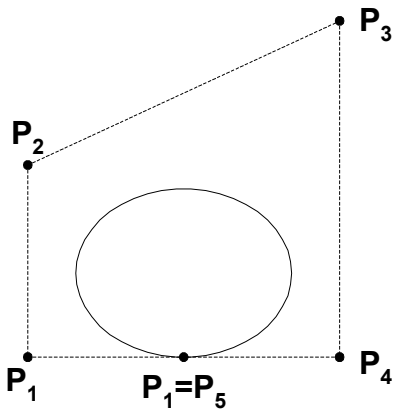
Las derivadas:

$$P'(0) = -nP_0 + nP_1$$
$$P'(1) = -nP_{n-1} + nP_n$$

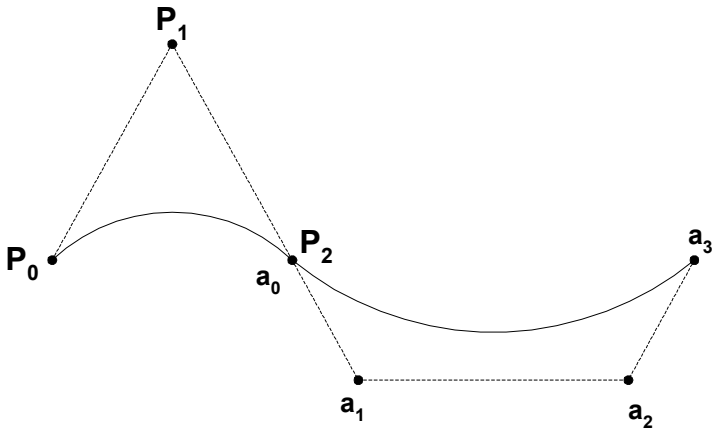
Segundas derivadas:

$$P''(0) = n(n-1)\left[(P_2 - P_1) - (P_1 - P_0)\right]$$
$$P''(1) = n(n-1)\left[(P_{n-2} - P_{n-1}) - (P_{n-1} - P_n)\right]$$

Otra propiedad importante de cualquier curva de Bezier es que siempre está situada dentro del convex hull.



6.6.2 *Composición de Curvas*



$a_0=P_2$ y los tres puntos P_1, (a_0,P_2), a_1 deben ser colineales para tener continuidad de 1° derivada.

6.6.3 *Curvas de Bezier cúbicas*

Son las más usadas y se generan con cuatro puntos de control:

$$BEZ_{0,3}(u) = (1-u)^3$$
$$BEZ_{1,3}(u) = 3u(1-u)^2$$
$$BEZ_{2,3}(u) = 3u^2(1-u)$$
$$BEZ_{3,3}(u) = u^3$$

$$P'(0) = 3(P_1 - kP_0) \qquad P'(1) = 3(P_3 - P_2)$$
$$P''(0) = 6(P_0 - 2P_1 + P_2) \qquad P''(1) = 6(P_1 - 2P_1 + P3)$$

En forma Matricial:

$$P(u) = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} \bullet M_{BEZ} \bullet \begin{bmatrix} P_1 \\ P^2 \\ P^3 \\ P^4 \end{bmatrix}$$

$$M_{BEZ} = \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

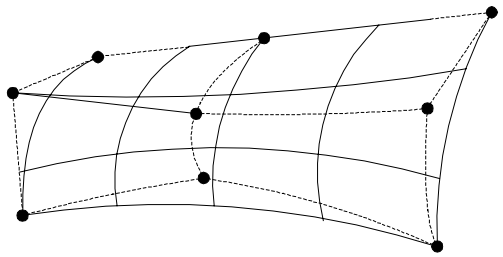
6.7 Superficies de Bezier

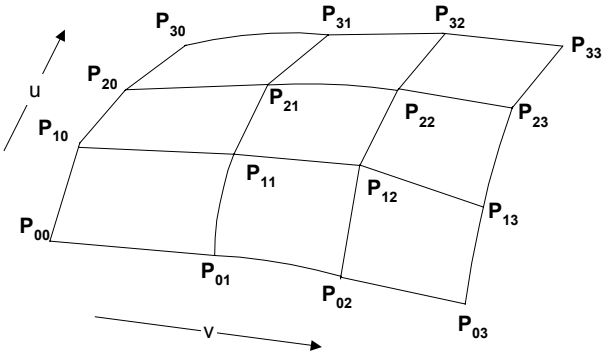
Se pueden utilizar dos grupos ortogonales de curvas de Bezier para diseñar una superficie.

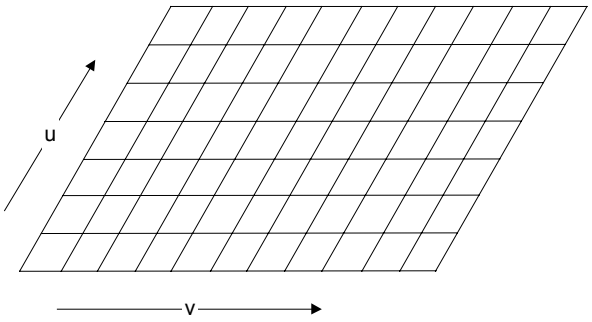
La ecuación de la superficie en forma paramétrica es:

$$P(u,v) = \sum_{j=0}^{m} \sum_{k=0}^{n} P_{j,k} \; BEZ_{j,m}(v) \;\; BEZ_{k,n}(u)$$

Con $P_{j,k}$ que especifica una ubicación de *(m+1)* por *(n+1)* puntos de control.







La forma Matricial es:

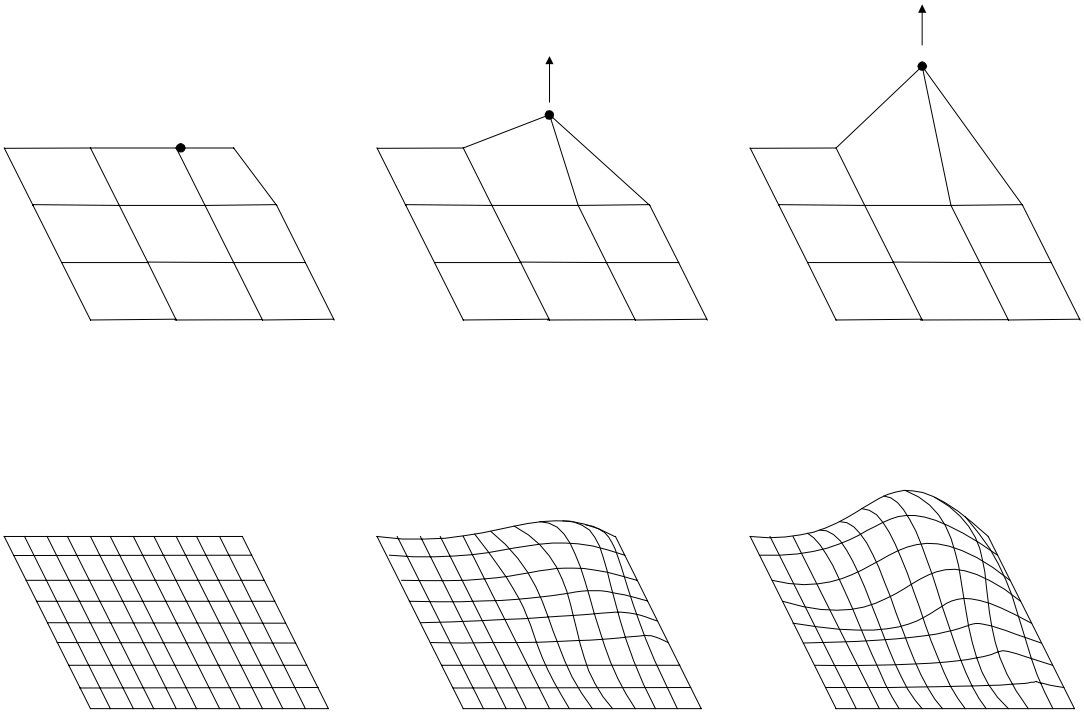
$$P(u,v) = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} \begin{bmatrix} M^{BEZ} & P & M_{BEZ}^T \end{bmatrix} \begin{bmatrix} v^3 \\ v^2 \\ v \\ 1 \end{bmatrix}$$

donde

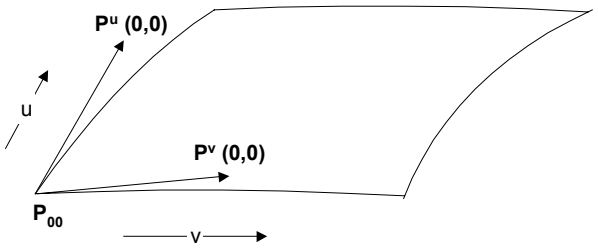
$$M_{BEZ} = \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

$$P = \begin{bmatrix} P_{00} & P_{01} & P_{02} & P_{03} \\ P_{10} & P_{11} & P_{12} & P_{13} \\ P_{20} & P_{21} & P_{22} & P_{23} \\ P_{30} & P_{31} & P_{32} & P_{33} \end{bmatrix}$$

Efecto de levantar un punto de control



Vectores tangentes a un vértice



$$P^u(0,0) = 3(P_{10} - P_{00})$$
$$P^v(0,0) = 3(P_{01} - P_{00})$$
$$P^{uv}(0,0) = 9(P_{00} - P_{01} - P_{10} + P_{11})$$

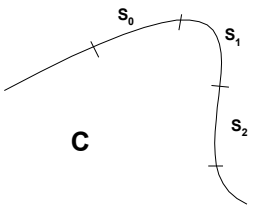
La normal a un vértice se calcula por el producto cruz entre dos tangentes.

Por ejemplo:

$$a = P_{01} - P_{00}$$
$$b = P_{10} - P_{00}$$
$$N = a \times b$$

6.8 Curvas B-splines

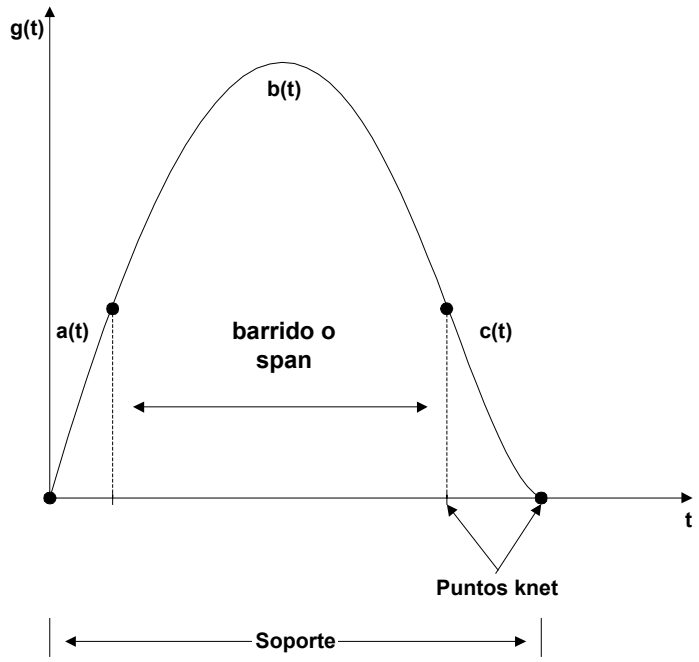
Una curva Spline está formada por curvas polinomiales, por arcos o sectores.



Por ejemplo, la curva *C* está formada por los arcos *S0, S1* y *S2*. A su vez *S0, S1* y *S2* podrían ser curvas de Bezier, las cuales están definidas por los polinomios de Bernstein.

6.8.1 *Construcción de Splines usando polinomios como base.*

De la siguiente curva:



$$a(t) = \frac{1}{2}t^2$$

$$b(t) = \frac{3}{4} - \left(t - \frac{3}{2}\right)^2$$

$$c(t) = \frac{1}{2}(3-t)^2$$

Una Spline de grado *m* tiene continuidad *m-1* en cada punto knot. Es decir $g(t)_m \in C^{m-1}$.

Considérese el construir una curva P(t) basándose en una serie de funciones gk(t) polinomiales. Por ejemplo:

$$P(t) = \sum_{k=0}^{4} P_k \; g_k(t)$$

donde

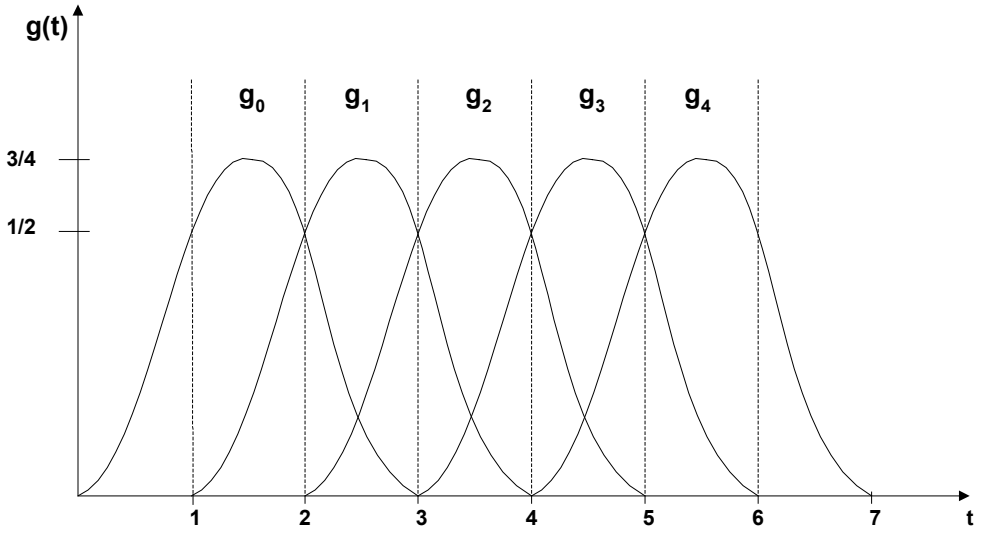
$$gk(t) = g(t-k) \qquad k = 0, 1, 2, 3, 4$$

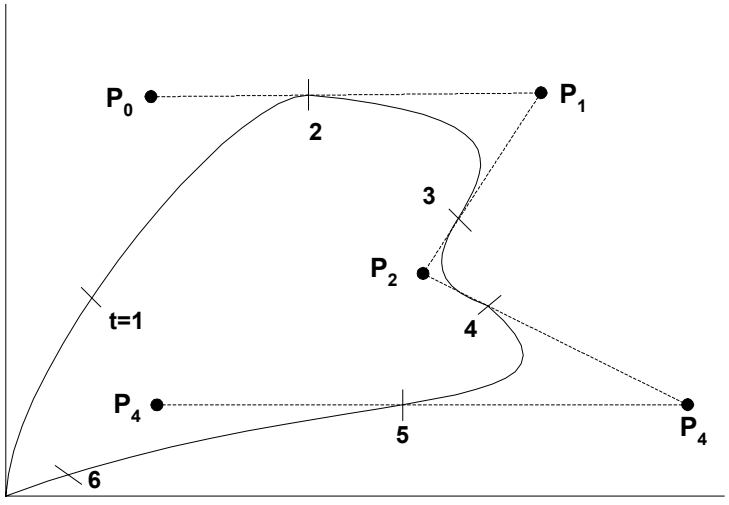
Se tiene que P(t) en cada "span" está dado por una suma de polinomios, con un peso P_k para cada uno de ellos.

Ejemplo:

$$P(t) = P_0\ g(t) \qquad 0 \le t \le 1$$
$$P(t) = P_0\ g(t) + P_1\ g(t-1) \qquad 1 \le t \le 2$$
$$P(t) = P_0\ g(t) + P_1\ g(t-1) + P_2\ g(t-2) \qquad 2 \le t \le 3$$
$$\downarrow$$
$$etc$$

Esto se puede graficar como:





De lo anterior se desprende que en un stan o barrido cualquier parte de esa curva equivale a una suma de polinomios ponderados por los puntos de control.

Se puede pensar entonces que puede existir un polinomio general, el cual sirva como base para cada caso.

6.8.2 Curva B-spline

La ecuación para una curva B-spline está dada por:

$$P(u) = \sum_{k=0}^{n} P_k \; B_{k,d}(u) \qquad u_{mín} \le u \le u_{máx}$$

$$2 \le d \le n+1$$

Donde *Pk* es un conjunto de *n+1* puntos de control. El rango del parámetro u depende ahora de como se eligen los parámetros de la B-splin.

Las funciones Blending $B_{k,d}$ son polinomios de grado *d-1*.

Las funciones Blendig están definidas por la fórmula recursiva de Cox-de Boor:

$$B_{k,1}(u) = \begin{cases} 1 & si\ u_k \le u \le u_{k+1} \\ 0 & otro\ caso \end{cases}$$

$$B_{k,d}(u) = \frac{u - u_k}{u_{k+d-1} - u_k} B_{k,d-1}(u) + \frac{u_{k+d} - u}{u_{k+d} - u_{k+1}} B_{k+1,d-1}(u)$$

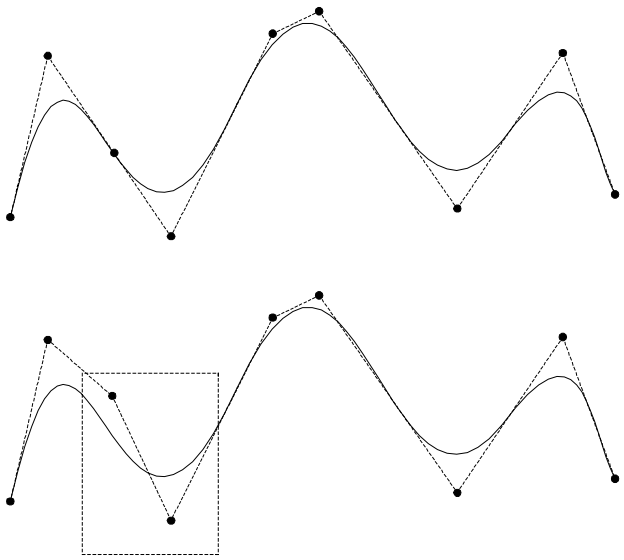
Donde cada función Blending está definida sobre *d* intervalos del rango total de *u*.

El conjunto de subintervalos finales u_j se llama vector knot.

Los valores para $u_{mín}$ y $u_{máx}$ dependen del número de puntos de control, el valor elegido para *d* y de como se eligen los subintervalos (vector knot).

El valor 0/0 se asume como 0 en caso de ocurrir en la fórmula anterior.

Las B-splines permiten que se cambie el número de puntos de control sin cambiar el grado del polinomio. También se pueden añadir o modificar los puntos de control para manipular las formas de las curvas.



Cambio en el punto de control

Las curvas B-splines tienen las siguientes propiedades:

1. La curva polinomial tiene el grado *d –1* y continuidad C^{d-2} sobre el rango de *u*.
2. Para *n+1* puntos de control, la curva se describe con *n+1* funciones Blending.
3. Cada función Blending $B_{k,d}$ está definida sobre *d* subintervalos del rango total de *u*, comenzando en el valor not u_k.
4. El rango del parámetro *u* está dividido en *n+d* subintervalos por los *n+d+1* valores especificados por el vector knot.
5. Con valores knot marcados como $\{u_0, u_1, ...u_{n+d}\}$ la curva B-spline resultante está definida en el intervalo u_{d-1} hasta el valor knot u_{n+1}.
6. Cada sección de la curva spline (entre dos valores sucesivos de knot) está influenciada por *d* puntos de control.
7. Cualquier punto de control puede afectar la forma de *a* lo más *d* secciones de curvas.

En general hay tres clases de vectores knot: uniforme, uniforme abierto y no-uniforme. Las b-spline se describen comúnmente de acuerdo al tipo de vector knot.

6.8.3 *B-splines uniformes*

Cuando el espacio entre valores knot es constante, la curva resultante se llama B-spline uniforme.

Por ejemplo:

$$\{-1.5, -1.0, -0.5, 0.0, 0.5, 1.0, 1.5, 2.0\}$$

A menudo los valores knot son normalizados entre 0 y 1.

$$\{0.0, 0.2, 0.4, 0.6, 0.8, 1.0\}$$

B-splines uniformes tienen funciones Blending uniformes. Esto es, para valores dados de *n* y *d*, todas las funciones Blending tienen la misma forma.

Ejemplo:

$$n = d = 3$$

El vector knot debe contener *d+n+1=7* valores knot.

$$\{0, 1, 2, 3, 4, 5, 6\}$$

y el rango de *u* es de 0 a 6, con *n+d=6* subintervalos.

La curva polinomial es de grado *d-1=2*.

Con *n+1=4* puntos de control, la curva está descrita por *n+1=4* funciones Blending.

Cada una de las cuatro funciones Blending span *d=3* subintervalos del rango total de *u*.

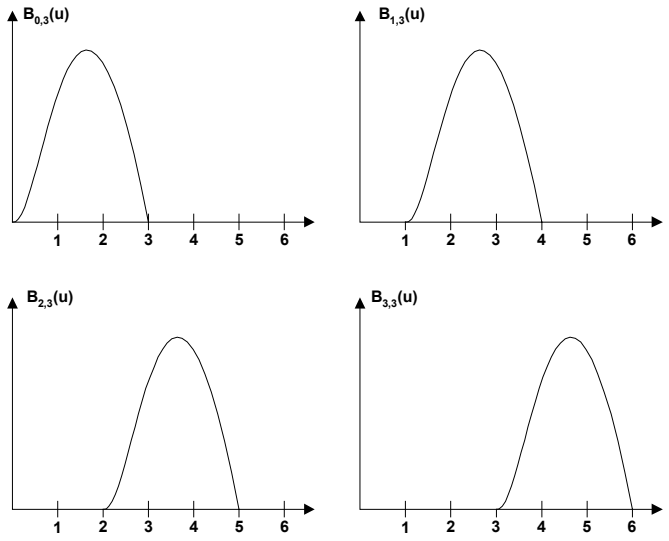
Así

$$B_{0,3}(u)=\begin{cases}\frac{1}{2}u^2 & 0\le u\le 1\\ \frac{1}{2}u(2-u)+\frac{1}{2}(u-1)(3-u) & 1\le u\le 2\\ \frac{1}{2}(3-u)^2 & 2\le u\le 3\end{cases}$$

$$B_{1,3}(u)=\begin{cases}\frac{1}{2}(u-1)^2 & 1\le u\le 2\\ \frac{1}{2}(u-1)(3-u)+\frac{1}{2}(u-2)(4-u) & 2\le u\le 3\\ \frac{1}{2}(4-u)^2 & 3\le u\le 4\end{cases}$$

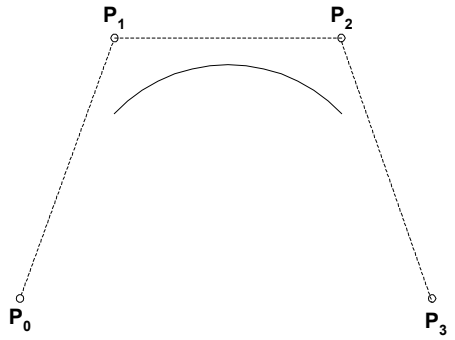
$$B_{2,3}(u)=\begin{cases}\frac{1}{2}(u-2)^2 & 2\le u\le 3\\ \frac{1}{2}(u-2)(4-u)+\frac{1}{2}(u-3)(5-u) & 3\le u\le 4\\ \frac{1}{2}(5-u)^2 & 4\le u\le 5\end{cases}$$

$$B_{3,3}(u)=\begin{cases}\frac{1}{2}(u-3)^2 & 3\le u\le 4\\ \frac{1}{2}(u-3)(5-u)+\frac{1}{2}(u-4)(6-u) & 4\le u\le 5\\ \frac{1}{2}(6-u)^2 & 5\le u\le 6\end{cases}$$



El primer punto de control multiplica a la función $B_{0,3}(u)$. Así, cambiando la posición del primer punto de control sólo afecta a la curva hasta u=3.

Todas las funciones Blending están presentes entre u_{d-1}=2 y u_{n+1}=4. Este es el rango de la curva polinomial. En este rango, la suma de las funciones es igual a 1.



Puesto que el rano de la curva polinomial resultante s entre 2 y 4, se pueden determinar los puntos de comienzo y término, evaluando las funciones blending en estos puntos.

$$P_{comienzo} = \frac{1}{2}(P_0 + P_1)$$

$$P_{final} = \frac{1}{2}(P_2 + P_3)$$

También las derivadas están dadas por:

$$P'_{comienzo} = P_1 - P_0$$
$$P'_{final} = P_3 - P_2$$

6.8.4 B-splines abiertas y uniformes

En este caso de B-spline el vector knot es uniforme, excepto en el principio y fin de éste.
Por ejemplo.

$$\{0,0,1,2,3,3\} \quad para\ d = 2\ y\ n = 3$$
$$\{0,0,0,0,1,2,2,2,2\} \quad para\ d = 4\ y\ n = 4$$

Para cualquier valor de *d* y *n* se puede generar un vector knot como

$$uj = \begin{cases} 0 & 0 \le j \le d \\ j - d + 1 & d \le j \le n \\ n - d + 2 & j > n \end{cases}$$

Para valores de *j* en el rango de *0* a *n+d*.

Con esta asignación los primeros *d* knots tienen un valor 0 y los últimos *d* knots tienen un valor *n-d+2*.

La curva polinomial para una B-spline abierta por el primer y último punto de control. En esto es similar a la curva de Bezier.

Ejemplo:

$$d = 3\ y\ n = 4 \qquad (cinco\ puntos\ de\ control)$$

$$knot\ \ vector = \{u_0, u_1, u_2, u_3, u_4, u_5, u_6, u_7, u_8\}$$
$$\{0, 0, 0, 1, 2, 3, 3, 3\}$$

El rango total de *u* está dividido en seis subintervalos, y cada función Blending está definida en tres subintervalos.

$$B_{0,3}(u) = (1-u)^2 \qquad 0 \le u < 1$$

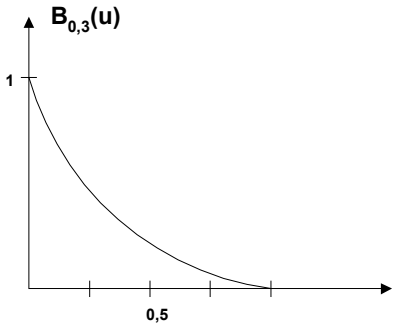
Está definida desde $u_0=0$ a $u_3=1$

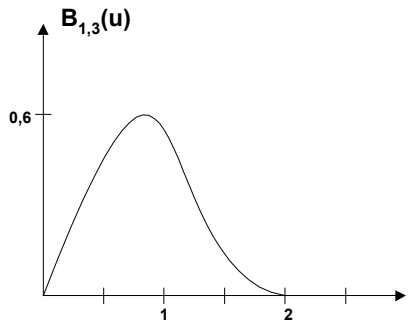
$$B_{1,3}(u) = \begin{cases} \frac{1}{2}u(4-3u) & 0 \le u < 1 \\ \frac{1}{2}(2-u)^2 & 1 \le u < 2 \end{cases}$$

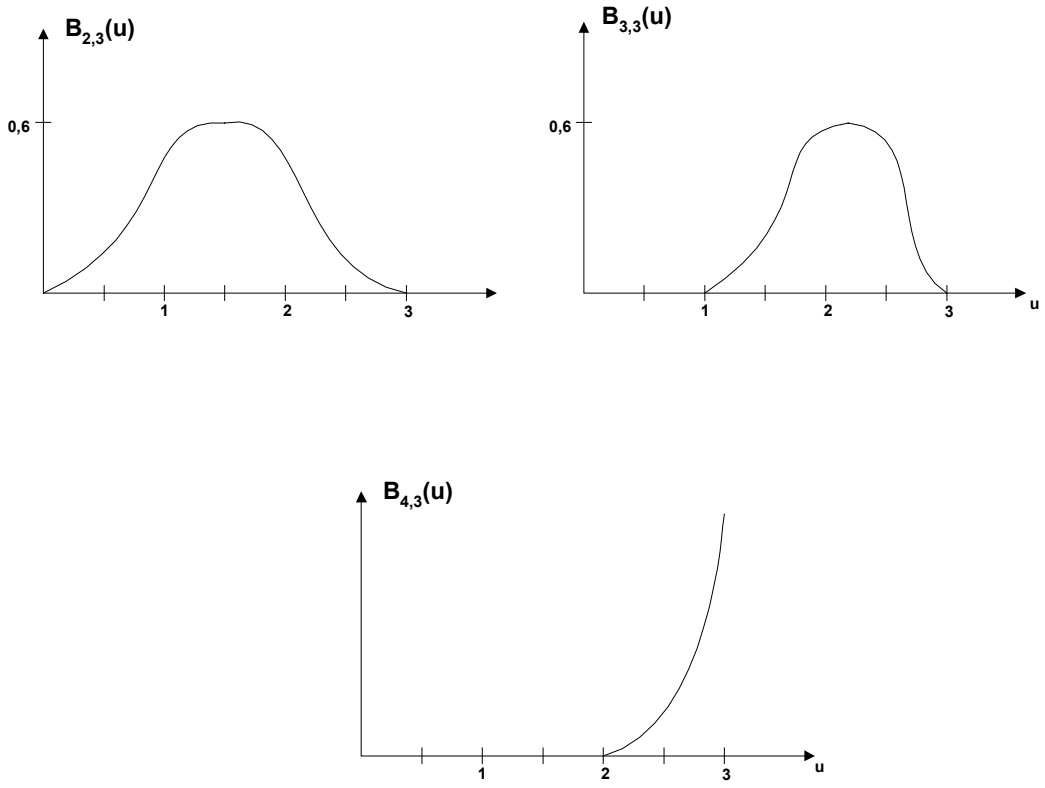
$$B_{2,3}(u) = \begin{cases} \frac{1}{2}u^2 & 0 \le u < 1 \\ \frac{1}{2}u\,(2-u) + \frac{1}{2}(u-1)\,(3-u) & 1 \le u < 2 \\ \frac{1}{2}(3-u)^2 & 2 \le u < 3 \end{cases}$$

$$B_{3,3}(u) = \begin{cases} \frac{1}{2}\,(u-1)^2 & 1 \le u < 2 \\ \frac{1}{2}(3-u)(3u-5) & 2 \le u < 3 \end{cases}$$

$$B_{4,3}(u) = (u-2)^2 \qquad 2 \le u < 3$$







6.9 Superficies

Esto es similar al caso Bezier. En efecto, la ecuación de la superficie es:

$$P(u,v) = \sum_{k_1=0}^{n_1} \sum_{k_2=0}^{n_2} P_{k1,k2}\ B_{k1,d1}(u)\ B_{k2,d2}(u)$$

6.9.1 Splines Racionales

Una función racional es simplemente la razón entre dos polinomios. Así, una spline racional es la razón de dos funciones splines.

$$P(u) = \frac{\sum_{k=0}^{n} W_k \; P_k \; B_{k,d}(u)}{\sum_{k=0}^{n} W_k \; B_{k,d}(u)}$$

Donde P_k es el conjunto de *n+1* puntos de control. Los parámetros W_k son factores de peso para los puntos de control. Mientras mayor sea el valor de un W_k, más cerca del punto de control está la curva. Esto es W_k empuja la curva hacia el punto de control.

Cuando todos los W_k son 1, entonces tenemos las B-splines conocidas porque el denominador de *P(u)* es 1 (la suma de las funciones Blending).

Las splines racionales tienen dos importantes ventajas sobre las no racionales. Primero, ellas pueden representar exactamente funciones cónicas como círculos y elipses. Funciones no racionales que son polinomios sólo pueden aproximar estas funciones cónicas. La otra ventaja es que son invariantes con respecto a una transformación de vista en perspectiva. Esto significa que se puede aplicar una transformación a los puntos de control y se obtiene la correcta vista de ella.

Construir una spline racional se lleva a cabo de la misma manera que para las no racionales. Dado el conjunto de puntos de control, el grado del polinomio, los factores de peso y el vector knot, se aplican las ecuaciones de recurrencia.

6.9.2 NURBS

NURBS es el término que se utiliza para indicar B-splines racionales no uniformes.

Para dibujar secciones cónicas con NURBS, se usa una función cuadrática spline (*d*=3) con tres puntos de control.

Se puede hacer esto con una spline definida por el vector knot abierto:

$$\{0, 0, 0, 1, 1, 1\}$$

las funciones de peso son:

$$W_0 = W_2 = 1$$

$$W_1 = \frac{r}{1-r^2} \qquad 0 \le r < 1$$

y

$$P(u) = \frac{P_0\, B_{0,3} + \left(\frac{r}{(1-r)}\right) P_1\, B_{1,3} + P_2\, B_{2,3}}{B_{0,3} + \left(\frac{r}{(1-r)}\right) B_{1,3} + B_{2,3}}$$

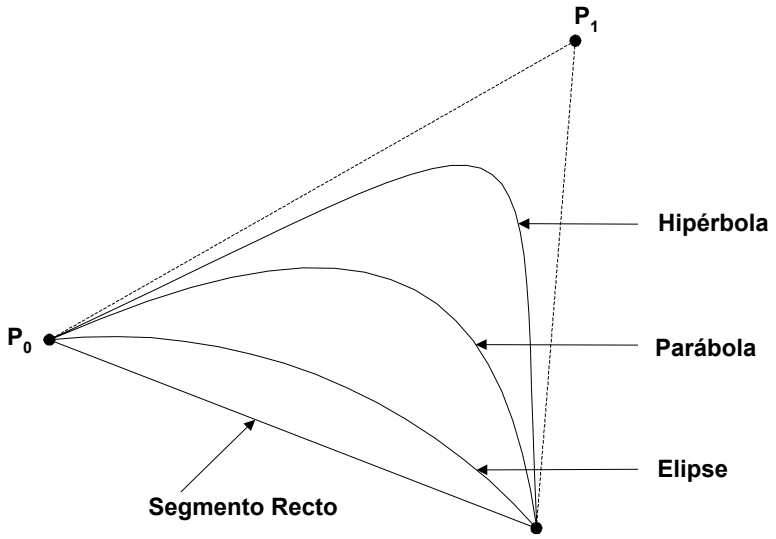
Se obtienen varias cónicas con los valores siguientes:

$$r > \frac{1}{2},\ W_1 > 1 \qquad (hiperbola)$$

$$r = \frac{1}{2},\ W_1 = 1 \qquad (parábola)$$

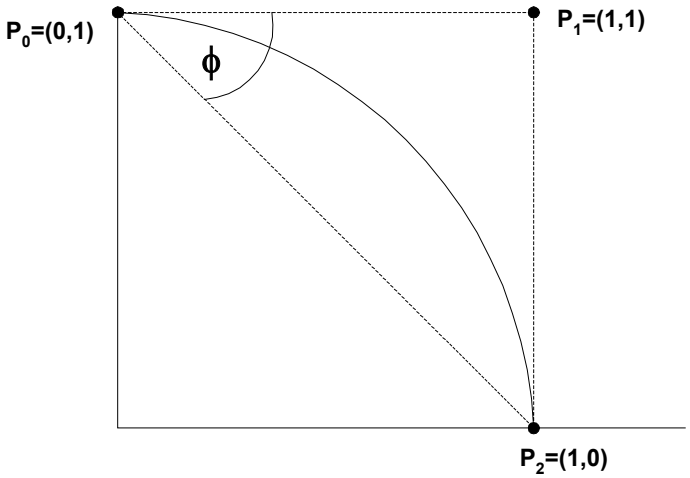
$$r = \frac{1}{2},\ W_1 < 1 \qquad (elipse)$$

$$r = 0,\ W_1 = 0 \qquad (segmento\ recto)$$



Se puede también generar un cuarto de círculo eligiendo $W_1=\cos\phi$ y eligiendo los puntos de control:

$$P_0 = (0,1) \qquad P_1 = (1,1) \qquad P_2 = (1,0)$$



Ejemplo 1 Utilizando OpenGl

Curvas

OpenGL contiene bastantes funciones que facilitan el dibujo de curvas y superficies de Bezier especificando los puntos de control y el rango de los parámetros u y v. Entonces, invocando la función evaluadora apropiada (el evaluador), se generarán los puntos que definen la curva o superficie.

Para dibujar una curva de bezier se debe proceder de la siguiente forma

Especificar los puntos de control, por ejemplo:

```
GLint nNumPoints = 4;

GLfloat ctrlPoints[4][3]=
{{  -4.0f, 0.0f, 0.0f}, // Punto final
{ -6.0f, 4.0f, 0.0f},   // Punto de control
{  6.0f, -4.0f, 0.0f},  // Punto de control
{  4.0f, 0.0f, 0.0f }}; // Punto final
```

Definimos la Curva Bezier

```
glMap1f(GL_MAP1_VERTEX_3,          // Tipo de dato generado
0.0f,                              // Rango menor de u
100.0f,                            // Rango superior de u
3,                                 // Dist. entre los puntos
nNumPoints,                        //num. de puntos de control
&ctrlPoints[0][0]);                // Matriz de puntos de control
glEnable(GL_MAP1_VERTEX_3);        // Activa el evaluador
// Usa una franja de líneas para "conectar los puntos"
glBegin(GL_LINE_STRIP);
        for(i = 0; i <= 100; i++)
                {
                // Evalúa la curva en este punto
                glEvalCoord1f((GLfloat) i);
                }
glEnd();
```

El primer parámetro de glMapf1, GL_MAP1_VERTEX_3, activa el evaluador para generar ternas de coordenadas de vértice (x,y,z), en oposición a GL_MAP1_VERTEX_4 que generaría las coordenadas y una componente alfa. Los dos siguientes parámetros especifican los valores mínimo y máximo del valor paramétrico u de esta curva. El cuarto parámetro especifica el número de valores de coma flotante entre los vértices de la matriz de puntos de control. El último parámetro es un puntero a un buffer que contiene los puntos de control empleados en definir la curva. Aquí pasamos un puntero al primer elemento de la matriz. Una vez que se ha creado el mapeado de la curva, llamamos al evaluador para hacer uso de este mapeado.

La función glEvalCoord1f toma un único argumento: un valor paramétrico a lo largo de la curva. Esta función evalúa entonces la curva en este valor y llama internamente a glVertex para dibujar ese punto. Haciendo un bucle en el dominio de la curva e invocando a glEvalCoord para producir vértices, podemos dibujar la curva con una simple tira de líneas.

OpenGL puede facilitar aún más las cosas. Definimos una rejilla con la función glMapGrid, que le dice a OpenGL que cree una rejilla de puntos separados sobre el dominio u. Entonces llamamos a glEvalMesh para conectar los puntos usando la primitiva especificada (GL_LINES o GL_POINTS). Por ejemplo:

```
// Usa funciones de más alto nivel para mapear una rejilla, y evaluar todo

// Mapa de una rejilla de 100 puntos de 0 a 100
        glMapGrid1d(100,0.0,100.0);

// Evalua la rejilla, usando líneas
        glEvalMesh1(GL_LINE,0,100);
```

Reemplaza por completo a este código:

```
// Usa una franja de líneas para "conectar los puntos"
```

```
glBegin(GL_LINE_STRIP);
        for(i = 0; i <= 100; i++)
                {
                // Evalúa la curva en este punto
                glEvalCoord1f((GLfloat) i);
                }
glEnd();
```

El siguiente ejemplo muestra la creación de una simple curva de bezier con 5 puntos de control.

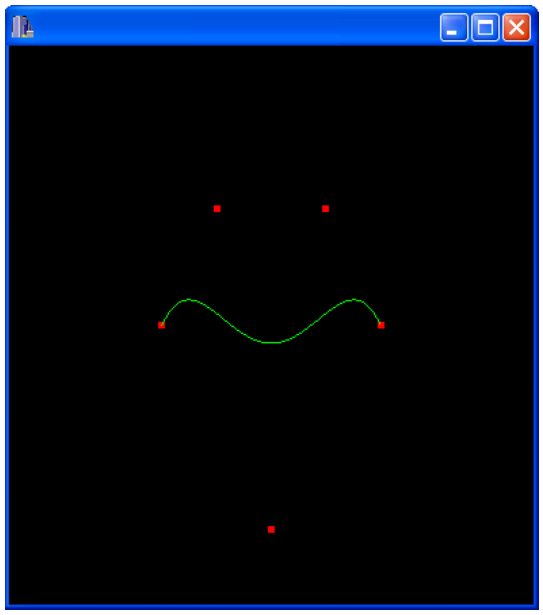
Definimos nuestros puntos de control

```
GLint nNumPoints = 5;

GLfloat ctrlPoints[5][3]= {{ -4.0f, 0.0f, 0.0f},        // Punto final
                           { -2.0f, 4.0f, 0.0f},        // Punto de control
                           {  0.0f, -7.0f, 0.0f},       // Punto de control
                        {  2.0f, 4.0f, 0.0f},           // Punto de control
                           {  4.0f,  0.0f, 0.0f }};     // Punto final

void __fastcall TFormMain::IdleLoop(TObject*, bool& done)
{
   done = false;
   RenderGLScene();
   SwapBuffers(hdc);
}
//---------------------------------------------------------------------------
void __fastcall TFormMain::FormCreate(TObject *Sender)
{
   hdc = GetDC(Handle);
   SetPixelFormatDescriptor();
   hrc = wglCreateContext(hdc);
   if(hrc == NULL)
     ShowMessage(":-)~ hrc == NULL");
   if(wglMakeCurrent(hdc, hrc) == false)
     ShowMessage("Could not MakeCurrent");
   w = ClientWidth;
   h = ClientHeight;
   glEnable(GL_DEPTH_TEST);
   glEnable(GL_CULL_FACE);
   glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
}
//---------------------------------------------------------------------------
void __fastcall TFormMain::SetPixelFormatDescriptor()
{
   PIXELFORMATDESCRIPTOR pfd = {
     sizeof(PIXELFORMATDESCRIPTOR),
       1,
       PFD_DRAW_TO_WINDOW | PFD_SUPPORT_OPENGL | PFD_DOUBLEBUFFER,
       PFD_TYPE_RGBA,
```

```
        24,
        0,0,0,0,0,0,
        0,0,
        0,0,0,0,0,
        32,
        0,
        0,
        PFD_MAIN_PLANE,
        0,
        0,0,
    };
    PixelFormat = ChoosePixelFormat(hdc, &pfd);
    SetPixelFormat(hdc, PixelFormat, &pfd);
}
//---------------------------------------------------------------------------
void __fastcall TFormMain::FormResize(TObject *Sender)
{
    GLfloat rango;
    w = ClientWidth;
    h = ClientHeight;

    if(h == 0)
      h = 1;

    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    rango=8;
    glFrustum(-rango,rango, -rango, rango, 100000,250000);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glTranslatef(0,0,-120000);
}
//---------------------------------------------------------------------------
void __fastcall TFormMain::RenderGLScene()
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    DrawObjects();
    glFlush();
}
//---------------------------------------------------------------------------
void __fastcall TFormMain::DrawObjects()
{
     //Define la Bezier,  Sólo necesitamos ejecutar ésto una vez y puede
     // estar en la función de inicio
     glMap1f(GL_MAP1_VERTEX_3,          // Tipo de dato generado
     0.0f,                              // Rango menor de u
     100.0f,                            // Rango superior de u
     3,                                 // Distancia entre los puntos almacenados
     nNumPoints,                        // numero de puntos de control
     &ctrlPoints[0][0]);                // Matriz de puntos de control

     // Activa el evaluador
```

```
      glEnable(GL_MAP1_VERTEX_3);
      glColor3f(0,1,0);
      glMapGrid1d(100,0.0,100.0);

      // Evalua la rejilla, usando líneas
        glEvalMesh1(GL_LINE,0,100);
         glMapGrid1d(100,0.0,100.0);

      // Evalua la rejilla, usando líneas
           glEvalMesh1(GL_LINE,0,100);
      // Dibuja los puntos d control
         DrawPoints();
}
//---------------------------------------------------------------------------
void __fastcall TFormMain::FormPaint(TObject *Sender)
{
   glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
   glFlush();
   DrawObjects();
}
//---------------------------------------------------------------------------
```

El resultado se muestra en la siguiente figura



Ejemplo 2 Utilizando OpenGl

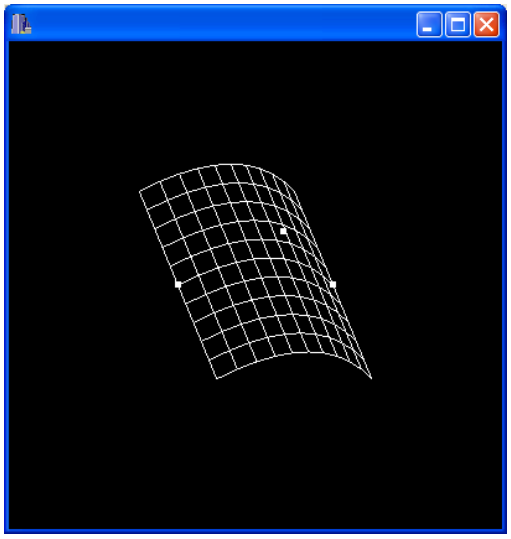
Superficies

Crear una superficie 3D de Bezier se parece mucho a la versión en 2D. Además de definir puntos a lo largo del dominio de u, también debemos definirlos a lo largo del dominio de v. El siguiente ejemplo muestra la malla alámbrica de una superficie 3D de Bezier. El primer cambio respecto al ejemplo anterior es que hemos definido tres grupos adicionales de puntos de control para la superficie a lo largo del dominio de v. Para mantener la simplicidad de la superficie, los puntos de control son iguales excepto en el valor Z. Esto creará una superficie uniforme, como si sencillamente extrajéramos una Bezier 2D a lo largo del eje Z.

```
// Número de puntos de control para esta curva
GLint nNumPoints = 3;

GLfloat ctrlPoints[3][3][3]= {{{  -4.0f, 0.0f, 4.0f},                    //V=0
                                { -2.0f, 4.0f, 4.0f},
                                {  4.0f, 0.0f, 4.0f }},

                                {{  -4.0f, 0.0f, 0.0f},                  //V=1
                                { -2.0f, 4.0f, 0.0f},
                                {  4.0f, 0.0f, 0.0f }},

                                {{  -4.0f, 0.0f, -4.0f},                 //V=2
                                { -2.0f, 4.0f, -4.0f},
                                {  4.0f, 0.0f, -4.0f }}};
void RenderScene(void)
        {
        glClear(GL_COLOR_BUFFER_BIT);
        glMatrixMode(GL_MODELVIEW);
        glPushMatrix();

        // Rota la malla sobre sí misma para que sea más fácil de ver
        glRotatef(45.0f, 0.0f, 1.0f, 0.0f);
        glRotatef(60.0f, 1.0f, 0.0f, 0.0f);
        glMap2f(GL_MAP2_VERTEX_3,
        0.0f,
        10.0f,
        3,
        3,
        0.0f,
        10.0f,
        9,
        3,
        &ctrlPoints[0][0][0]);

       // Activa el evaluador
```

```
glEnable(GL_MAP2_VERTEX_3);

// Utiliza las funciones de mayor nivel para mapear
// en una rejilla, luego evalúa todo

// Mapea una rejilla de 10 puntos de 0 a 10
glMapGrid2f(10,0.0f,10.0f,10,0.0f,10.0f);

// Evalua la rejilla, usando líneas
glEvalMesh2(GL_LINE,0,10,0,10);

// Dibuja los puntos de control
DrawPoints();
glPopMatrix();
glFlush();
}
```

Ahora, nuestro código de generación, además de rotar la figura para conseguir un efecto mejor, llamamos a glMap2f en lugar de glMap1f. Esto especifica puntos de control a lo largo de dos dominios (u y v) en vez de sólo uno (u). El resultado final es el mostrado en la siguiente figura:



El listado de la funciones principales de nuestro programa es el siguiente:

```
GLint nNumPoints = 3;

GLfloat ctrlPoints[3][3][3]= {{{  -4.0f, 0.0f, 4.0f},          //V=0
                               { -2.0f, 4.0f, 4.0f},
                               {  4.0f, 0.0f, 4.0f }},
```

```
                                {{  -4.0f, 0.0f, 0.0f},          //V=1
                                { -2.0f, 4.0f, 0.0f},
                                {  4.0f, 0.0f, 0.0f }},

                                {{  -4.0f, 0.0f, -4.0f},         //V=2
                                { -2.0f, 4.0f, -4.0f},
                                {  4.0f, 0.0f, -4.0f }}};


TFormMain *FormMain;



__fastcall TFormMain::TFormMain(TComponent* Owner)
    : TForm(Owner)
{
    Application->OnIdle = IdleLoop;
    size = 50.0f;
}
//---------------------------------------------------------------------------
void __fastcall TFormMain::IdleLoop(TObject*, bool& done)
{
    done = false;
    RenderGLScene();
    SwapBuffers(hdc);
}
//---------------------------------------------------------------------------
void __fastcall TFormMain::FormCreate(TObject *Sender)
{
    hdc = GetDC(Handle);
    SetPixelFormatDescriptor();
    hrc = wglCreateContext(hdc);
    if(hrc == NULL)
      ShowMessage(":-)~ hrc == NULL");
    if(wglMakeCurrent(hdc, hrc) == false)
      ShowMessage("Could not MakeCurrent");
    w = ClientWidth;
    h = ClientHeight;

    glEnable(GL_DEPTH_TEST);
    glEnable(GL_CULL_FACE);

    glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
}
//---------------------------------------------------------------------------
void __fastcall TFormMain::SetPixelFormatDescriptor()
{
    PIXELFORMATDESCRIPTOR pfd = {
      sizeof(PIXELFORMATDESCRIPTOR),
        1,
        PFD_DRAW_TO_WINDOW | PFD_SUPPORT_OPENGL | PFD_DOUBLEBUFFER,
        PFD_TYPE_RGBA,
        24,
```

```
        0,0,0,0,0,0,
        0,0,
        0,0,0,0,0,
        32,
        0,
        0,
        PFD_MAIN_PLANE,
        0,
        0,0,
    };
    PixelFormat = ChoosePixelFormat(hdc, &pfd);
    SetPixelFormat(hdc, PixelFormat, &pfd);
}
//---------------------------------------------------------------------------
void __fastcall TFormMain::FormResize(TObject *Sender)
{
    GLfloat rango;
    w = ClientWidth;
    h = ClientHeight;

    if(h == 0)
      h = 1;

    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    rango=6;
     glFrustum(-rango,rango, -rango, rango, 100000,250000);
     glMatrixMode(GL_MODELVIEW);
     glLoadIdentity();
     glTranslatef(0,0,-150000);
}
//---------------------------------------------------------------------------
void __fastcall TFormMain::DrawObjects()
{

      glRotatef(45.0f, 0.0f, 1.0f, 0.0f);
      glRotatef(60.0f, 1.0f, 0.0f, 0.0f);
      glMap2f(GL_MAP2_VERTEX_3,
      0.0f,
      10.0f,
      3,
      3,
      0.0f,
      10.0f,
      9,
      3,
      &ctrlPoints[0][0][0]);
      glEnable(GL_MAP2_VERTEX_3);
      glMapGrid2f(10,0.0f,10.0f,10,0.0f,10.0f);
      glEvalMesh2(GL_LINE,0,10,0,10);
      DrawPoints();
}
```

```
//---------------------------------------------------------------------------
void __fastcall TFormMain::FormPaint(TObject *Sender)
{
   glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
   glFlush();
   DrawObjects();
}
//---------------------------------------------------------------------------


void __fastcall TFormMain::Salir1Click(TObject *Sender)
{
exit(0);
}
//---------------------------------------------------------------------------

void DrawPoints(void)
     {
     int i;

     glPointSize(5.0f);

     glBegin(GL_POINTS);
               for(i = 0; i < nNumPoints; i++)
                         glVertex2fv(ctrlPoints[i][i]);
     glEnd();
     }
```

Ejemplo 3 Utilizando OpenGl

NURBS

También podemos crear una superficie con características de materiales, para esto procedemos agregando las propiedades de material e iluminación que hemos visto en capítulos anteriores, la siguiente figura muestra un ejemplo de ello

Podemos usar evaluadores en nuestros procesos fundamentales para evaluar superficies de bezier de cualquier grado, pero con las curvas más complejas será necesario ensamblar superficies independientes. A medida que añadimos más puntos de control, es más difícil crear una curva que mantenga una

buena continuidad. Podemos disponer de un nivel de control mayor con las funciones NURBS de la librería glu. NURBS significa *Non-Uniform Rational B-Spline* (concatenado-B racional no uniforme).

La potencia real de las NURBS es exprimir la influencia de los cuatro puntos de control de un segmento cualquiera de la curva para producir la suavidad necesaria. Este control se realiza a través de una secuencia de valores denominados *nudos*.

Para cada punto de control definimos dos valores nodales. El rango de valores para nudos se ajusta al dominio paramétrico de u y v, y no pueden ser descendientes. Esto se debe a que los valores de nudo determinan la influencia de los puntos de control que caen dentro de ese rango en el espacio u/v. La secuencia de nudos define la intensidad de la influencia de cada punto en el dominio. Si se repite el valor de un nudo, los puntos cercanos a este valor paramétrico tiene una mayor influencia.

La repetición de los valores de nudo se denomina *multiplicidad nodal*. Una mayor multiplicidad nodal decrece la curvatura de la curva o superficie dentro de esa región.

Las funciones NURBS de la librería glu proporcionan utilidades de alto nivel para generar superficies. No debemos llamar explícitamente a los evaluadores o establecer los mapas y rejillas. Para generar una NURBS, primero creamos un objeto NURBS al que nos referiremos cada vez que llamemos a las funciones NURBS relacionadas para modificar la apariencia de la curva o superficie.

La función gluNewNurbsRenderer crea un generador para la NURBS, y gluDeleteNurbsRenderer lo destruye. Los siguientes fragmentos demuestran el uso de estas funciones:

```
//Puntero a un objeto NURBS
GLUnurbsObj *pnurb=NULL;

//Definición del objeto NURBS
pnurb=gluNewNurbsRenderer();
...
//Efectuamos nuestras modificaciones NURBS
...
//Borra el objeto NURBS si se creó
if (pNurb)
    gluDeleteNurbsRenderer(pNurb);
```

Una vez que hemos creado un generador NURBS, podemos definir varias propiedades NURBS de alto nivel, como esta:

```
//Definición de la tolerancia de muestreo
gluNurbsProperty(pNurb, GLU_SAMPLIG_TOLERANCE, 25.of);

//Creamos una superficie sólida rellenando
// (usamos GLU_OUTLINE_POLYGON para crear una malla poligonal
gluNurbsProperty(pNurb, GLU_DISPLAY_MODE, (GLfloat)GLU_FILL);
```

Normalmente invocaremos estas funciones en nuestra rutina de inicio, mejor que hacerlo repetidamente en nuestro código. En este ejemplo, GLU_SAMPLING_TOLERANCE define cómo es de suave la malla que define la superficie, y GLU_FILL le dice a OpenGL que rellene la malla en lugar de generar una figura alámbrica.

La definición de la superficie se pasa en forma de matrices de puntos de control y secuencias de nudos a la función gluNurbsSurface. Como se muestra aquí, esta función también está encerrada entre llamadas a gluBeginSurface y gluEndSurface.

```
// Genera la NURBS
// Comienza la definición NURBS
   gluBeginSurface(pNurb);

// Evalua la superficie
gluNurbsSurface(pNurb,  // Puntero al generador NURBS
      8, Knots,          // Núm. de nudos y matriz de nudos direcc.  u
      8, Knots,          // Núm. de nudos y matriz de nudos  direcc. v
      4 * 3,             // Distancia entre puntos de control direcc.  u
      3,                 // Distancia entre puntos de control  direcc. v
      &ctrlPoints[0][0][0],   // Puntos de control
      4, 4,                   // Ordenes u y v de superficie
      GL_MAP2_VERTEX_3);      // Tipo de superficie
// superficie terminada
gluEndSurface(pNurb);
```

Podemos hacer más llamadas a gluNurbsSurface para crear cualquier número de superficies NURBS, pero las propiedades que definimos para el generador NURBS seguirán teniendo efecto. Usando los puntos de control y los valores nodales mostrados en el siguiente segmento de código, produciremos la superficie NURBS que hay a continuación.

```
   // La malla se extiende cuatro unidades de -6 a +6 a lo largo
  // de los ejes x e y,  descansa en el plano Z    u  v  (x,y,z)

GLfloat ctrlPoints[4][4][3]= {{{  -6.0f, -6.0f, 0.0f},      // u = 0,   v = 0
                {            -6.0f, -2.0f, 0.0f},   //          v = 1
               {  -6.0f,  2.0f, 0.0f},              //          v = 2
               {  -6.0f,  6.0f, 0.0f}},             /          v = 3

               {{  -2.0f, -6.0f, 0.0f},             // u = 1    v = 0
                {  -2.0f, -2.0f, 8.0f},             //          v = 1
                {  -2.0f,  2.0f, 8.0f},             //          v = 2
               {  -2.0f,  6.0f, 0.0f}},             //          v = 3

               {{  2.0f, -6.0f, 0.0f },             // u =2 v = 0
                {   2.0f, -2.0f, 8.0f },            //          v = 1
               {   2.0f,  2.0f, 8.0f },             //          v = 2
                {   2.0f,  6.0f, 0.0f }},           //          v = 3

               {{  6.0f, -6.0f, 0.0f},              // u = 3    v = 0
               {   6.0f, -2.0f, 0.0f},              //          v = 1
```

```
                {   6.0f,  2.0f, 0.0f},          //          v = 2
                {   6.0f,  6.0f, 0.0f}}};        //          v = 3

    // Secuencia de nudos de las NURBS
    GLfloat Knots[8] = {0.0f, 0.0f, 0.0f, 0.0f, 1.0f, 1.0f, 1.0f, 1.0f};
```

El siguiente ejemplo muestra las funciones principales de la creación y visualización de una superficie NURBS

```
GLfloat points[13][13][3] = {…  Puntos….}
GLint numPoints;
    GLint nNumPoints = 3;
 GLfloat ctrlPoints[4][4][3]= {{{  -6.0f, -6.0f, 0.0f},   // u = 0,          v = 0
                {          -6.0f, -2.0f, 0.0f},        //          v = 1
               {   -6.0f,  2.0f, 0.0f},       //                   v = 2
               {   -6.0f,  6.0f, 0.0f}},      //                   v = 3

               {{  -2.0f, -6.0f, 0.0f},       // u = 1v = 0
                {   -2.0f, -2.0f, 8.0f},      //                   v = 1
                {   -2.0f,  2.0f, 8.0f},      //                   v = 2
               {   -2.0f,  6.0f, 0.0f}},      //                   v = 3

               {{   2.0f, -6.0f, 0.0f },      // u =2              v = 0
                {    2.0f, -2.0f, 8.0f },     //                   v = 1
               {    2.0f,  2.0f, 8.0f },      //                   v = 2
                {    2.0f,  6.0f, 0.0f }},    //                   v = 3

               {{   6.0f, -6.0f, 0.0f},       // u = 3v = 0
               {    6.0f, -2.0f, 0.0f},       //                   v = 1
               {    6.0f,  2.0f, 0.0f},       //                   v = 2
               {    6.0f,  6.0f, 0.0f}}};     //                   v = 3

// Secuencia de nudos de las NURBS

GLfloat Knots[8] = {0.0f, 0.0f, 0.0f, 0.0f, 1.0f, 1.0f, 1.0f, 1.0f};

TFormMain *FormMain;



//---------------------------------------------------------------------------
__fastcall TFormMain::TFormMain(TComponent* Owner)
    : TForm(Owner)
{
    Application->OnIdle = IdleLoop;
    size = 50.0f;
}
//---------------------------------------------------------------------------
void __fastcall TFormMain::FormCreate(TObject *Sender)
{
```

```
    hdc = GetDC(Handle);
    SetPixelFormatDescriptor();
    hrc = wglCreateContext(hdc);
    if(hrc == NULL)
      ShowMessage(":-)~ hrc == NULL");
    if(wglMakeCurrent(hdc, hrc) == false)
      ShowMessage("Could not MakeCurrent");
    w = ClientWidth;
    h = ClientHeight;

    glEnable(GL_DEPTH_TEST);
    glEnable(GL_CULL_FACE);

    glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
}
//---------------------------------------------------------------------------
void __fastcall TFormMain::SetPixelFormatDescriptor()
{
    PIXELFORMATDESCRIPTOR pfd = {
      sizeof(PIXELFORMATDESCRIPTOR),
        1,
        PFD_DRAW_TO_WINDOW | PFD_SUPPORT_OPENGL | PFD_DOUBLEBUFFER,
        PFD_TYPE_RGBA,
        24,
        0,0,0,0,0,0,
        0,0,
        0,0,0,0,0,
        32,
        0,
        0,
        PFD_MAIN_PLANE,
        0,
        0,0,
    };
    PixelFormat = ChoosePixelFormat(hdc, &pfd);
    SetPixelFormat(hdc, PixelFormat, &pfd);
}
//---------------------------------------------------------------------------
void __fastcall TFormMain::FormResize(TObject *Sender)
{
    GLfloat rango;
    w = ClientWidth;
    h = ClientHeight;

    if(h == 0)
      h = 1;

    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    rango=30;
     glFrustum(-rango,rango, -rango, rango, 100000,250000);
     glMatrixMode(GL_MODELVIEW);
```

```
    glLoadIdentity();
    glTranslatef(35,-15,-150000);
    glRotatef(30,0,1,1);
}
//---------------------------------------------------------------------------
void __fastcall TFormMain::RenderGLScene()
{

   glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
   DrawObjects();
   glFlush();
}
//---------------------------------------------------------------------------
void __fastcall TFormMain::DrawObjects()
{

  GLUnurbsObj *pNurb=NULL;

//Definición del objeto NURBS
pNurb=gluNewNurbsRenderer();

 gluNurbsProperty(pNurb, GLU_SAMPLING_TOLERANCE, 5.0f);

gluNurbsProperty(pNurb, GLU_DISPLAY_MODE, (GLfloat)GLU_FILL);
gluBeginSurface(pNurb);

gluNurbsSurface(pNurb,
       8, Knots,
       8, Knots,
       4 * 3,
       3,
       &ctrlPoints[0][0][0],
       4, 4,
       GL_MAP2_VERTEX_3);


//Borra el objeto NURBS si se creó
if (pNurb)
    gluDeleteNurbsRenderer(pNurb);

createSurface();
drawSurface();
}
//---------------------------------------------------------------------------
void __fastcall TFormMain::FormPaint(TObject *Sender)
{
   glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
   glFlush();
   DrawObjects();
}
//---------------------------------------------------------------------------
oid __fastcall TFormMain::Salir1Click(TObject *Sender)
{
```

```
exit(0);
}
//--------------------------------------------------------------------------

void createSurface()
{
  GLfloat materialDiffuse[] = { 1.0, 0.0, 0.0, 1.0 };
  GLfloat materialSpecular[] = { 1.0 ,0.0, 0.0, 1.0 };
  GLfloat materialShininess[] = { 80 };
  GLfloat specular[]= {1.0f, 1.0f, 1.0f, 1.0f};

   GLfloat specref1[]={1.0f,1.0f,1.0f,1.0f};

  glMaterialfv( GL_FRONT, GL_DIFFUSE, materialDiffuse );
  glMaterialfv( GL_FRONT, GL_SPECULAR, materialSpecular );
  glMaterialfv( GL_FRONT, GL_SHININESS, materialShininess );
  glMaterialfv(GL_FRONT, GL_SPECULAR, specref1);


  glEnable( GL_LIGHTING );
  glEnable( GL_LIGHT0 );
  glDepthFunc( GL_LEQUAL );
  glEnable( GL_DEPTH_TEST );
  glEnable( GL_AUTO_NORMAL );
  glEnable( GL_NORMALIZE );

  nurbSurface = gluNewNurbsRenderer();
  gluNurbsProperty( nurbSurface, GLU_SAMPLING_TOLERANCE, 25.0 );
  gluNurbsProperty( nurbSurface, GLU_DISPLAY_MODE, GLU_FILL );
}

void drawSurface()
{
  GLfloat knots[26] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1 };

  glColor3f( 0.0, 1.0, 0.0 );
  gluBeginSurface( nurbSurface );
  gluNurbsSurface( nurbSurface, 26, knots, 26, knots,
        13*3, 3, &points[0][0][0], 13, 13, GL_MAP2_VERTEX_3 );
  gluEndSurface( nurbSurface );

  }
```

El resultado final es el mostrado en la siguiente figura: