Curvas y Superficies para modelado geométrico

Es muy difícil definir con precisión lo que es una curva y en general depende del contexto. En Computación Gráfica y en general en CAGD (Computer Aided Geometric Design), aceptaremos que una curva es un conjunto continuo y unidimensional de puntos, es decir que todas las coordenadas reales {x, y, z} varían como funciones continuas de un solo parámetro real. Siempre se puede encontrar alguna chicana que haga las cosas difíciles, por ejemplo podría hacer $x = sen(t)/t$ con cualquier otro juego de coordenadas para y y z y preguntarme si eso es una curva en $x = 0$. Nosotros sólo nos interesaremos por la representación gráfica de una sucesión de puntos y sólo en casos simples, sin elementos raros: si tiene vaivenes o puntas no son infinitos en el entorno de un punto, etc. Una superficie será entonces un conjunto continuo bidimensional de puntos, donde las coordenadas son funciones "bonitas" de dos parámetros.

La definición analítica de una dada curva puede hacerse de varios modos y se relaciona directamente con la forma de representarla gráficamente:

- **Explicita:**

  Es la más conocida desde que nos enseñaron a utilizar las coordenadas cartesianas para graficar funciones. Se provee un mapeo directo de una coordenada hacia las otras, usualmente mediante una ecuación del tipo:

  Curva plana: $y = f(x)$. Superficie 3D: $z = f(x,y)$.

  Por ejemplo, la ecuación $y = m\,x + b$ define una recta en el plano. Si restringimos la variación de x en un cierto intervalo podemos definir un segmento. Definida de esta forma, una curva es muy fácil de graficar ya que podemos variar sistemáticamente x en el intervalo y obtener los correspondientes valores de y evaluando directamente la función.

  Para una curva en 3D deberán darse dos funciones, una para y y otra para z, ambas en función de la variable independiente x.

  No todas las curvas pueden representarse correctamente de esta forma. Por ejemplo, para representar una recta vertical tendríamos que invertir el rol de las variables y expresar x en función de y. Representar un círculo no sería posible, ya que para cada x se tiene dos posibles valores de y, y ningún cambio de base permite salvar este inconveniente. No se puede utilizar esta representación cuando las variables dependientes no son funciones de las independientes.

- **Implícita:**

  La definición es una relación analítica entre las variables pero sin una receta para deducir unas en función de otras:

  Curva plana: $f(x,y) = 0$. Superficie 3D: $f(x,y,z) = 0$.

  Por ejemplo, para definir una circunferencia podemos utilizar la ecuación $x^2 + y^2 = r^2$. Los puntos de la curva serán todos aquellos puntos que satisfacen esta ecuación. Para curvas y superficies estándar (rectas, círculos, planos, toroides, etc), este tipo de definición es más directa y permite visualizar y modificar sencillamente parámetros importantes y específicos de cada curva (radio en un círculo, distancia al origen en un plano, etc.).

  En esta representación, las curvas en 3D se definen mediante un par de funciones de x, y, z, lo que equivale a definirlas como intersección de dos superficies.

  La desventaja es que dada la ecuación, la curva puede ser difícil de graficar, ya que no se obtienen los puntos de la curva en forma directa. Por ejemplo: supongamos que no sabemos lo que representa la ecuación $x^2 + y^2 = 1$; para un dado x, digamos $x = 3$, despejamos y en función de x, obteniendo valores complejos, sin representación en el espacio real.

  Esta forma sólo se utiliza en los casos especiales en que las constantes tienen un significado geométrico concreto y además se conocen los discriminantes que discriminan el tipo de gráfica y el rango de las variables, de acuerdo a los valores de las constantes. En general sólo planos, curvas cónicas y superficies cuádricas.

- **Paramétrica:**

  Es la forma que utilizamos para definir matemáticamente las curvas y superficies. Los valores de cada coordenada se expresan como función de uno o dos parámetros independientes reales, que no son coordenadas:

  Curva plana: $x=f(t); y=g(t)$. Superficie en 3D: $x=f(t,u); y=g(t,u); z=h(t,u)$.

  Para una curva, estas funciones dan un mapeo arbitrario y directo de cada t a cada punto de la curva. Si imaginamos que dibujamos la curva con un lápiz en un papel, el parámetro t puede verse como el tiempo, y las ecuaciones responden a la pregunta: ¿Donde está el lápiz en el tiempo t?

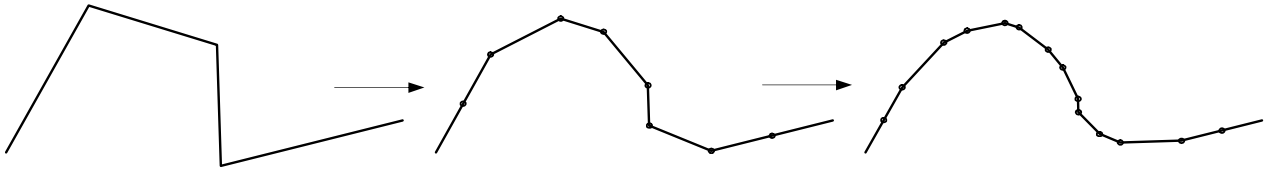
  Por ejemplo, una recta podría expresarse como: $x = a\ t;\ y = b\ t$ y una circunferencia podría expresarse como: $x = r\ cos(t);\ \ y = r\ sin(t)$.

  Una curva en el espacio simplemente agrega $z = h(t)$ al conjunto de ecuaciones.

  Esta forma la preferida en CG por la facilidad con que se obtienen los puntos para graficar y por su generalidad (puede representar todo tipo de curvas).

- **Generativa o Procedural:**

  Esta forma es totalmente diferente de las anteriores pues la curva o superficie es el resultado de un procedimiento o algoritmo; en general modificando una curva o superficie simple de partida. Ej.: Dividir en tres cada tramo de una poligonal y reemplazar los dos tramos que convergen a un vértice original por el que une los dos nuevos vértices cercanos al original:

  Los ejemplos más usados son las superficies por subdivisión y los fractales.

  Normalmente las curvas y superficies procedurales se definen como el límite para infinitas iteraciones del procedimiento que las genera.

Por ahora nos concentraremos en la forma paramétrica, que representaremos en forma vectorial, definiendo las componentes del vector posición de un punto genérico:

Curva: $P(u) = \{x(u),\ y(u),\ z(u)\ \}$

Superficie: $P(u,v)=\{x(u,v),\ y(u,v),\ z(u,v)\}$.

Estas funciones pueden ser de cualquier tipo; sin embargo en CG se suelen preferir las funciones polinómicas por su simplicidad de evaluación (existen algoritmos computacionales muy eficientes) y sus bondades matemáticas (derivadas sencillas y continuas).

Un polinomio vectorial de grado n se puede escribir como:

$P(u) = \sum_0^n \boldsymbol{a}_i\, u^i$ (aquí el superíndice del parámetro indicará siempre potenciación)

Los $n+1$ vectores $\boldsymbol{a}_i$ agrupan los coeficientes de los polinomios en x, y y z, de modo que la anterior es la representación de tres ecuaciones escalares como una vectorial.

En matemática se habla de espacios funcionales, donde una función dada es como un punto de un espacio. En particular, para los polinomios, la base son los monomios u^i y las "coordenadas" son los vectores $\mathbf{a}_i$. La dimensión de este espacio es el grado+1 (el 1 por el término independiente u^0).

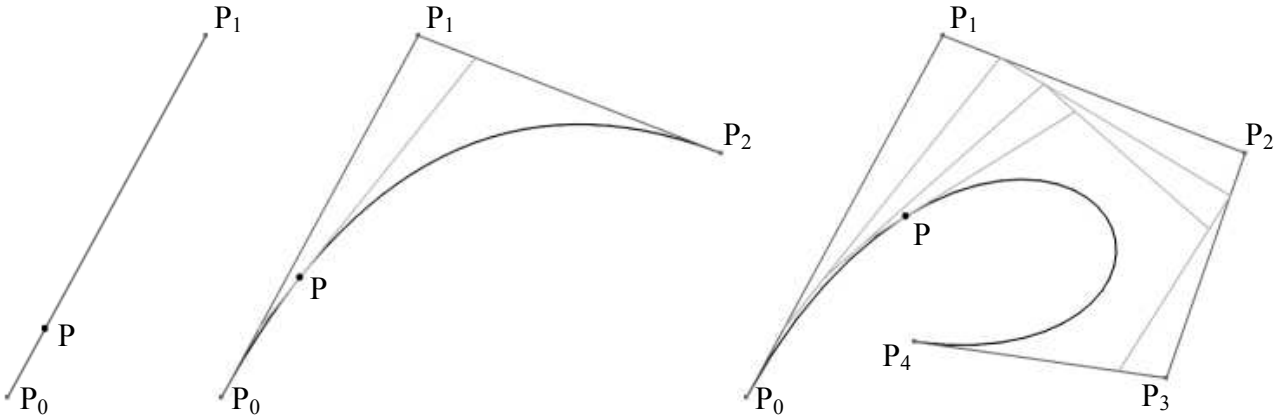
Para el usuario de un programa (CAD) que diseña la curva, no resulta sencillo definir el conjunto de parámetros que hace que la curva tenga la forma que desea. Lo que suele hacerse es imponer un conjunto de puntos que definen la curva por interpolación o por aproximación, y requerir algún grado de suavidad de la curva, es decir que las derivadas varíen de algún modo predeterminado.

Curvas de Bézier

Se denomina curva de Bézier a un método de definición de una curva en serie de potencias. El método consiste en definir algunos **puntos de control**, a partir de los cuales se calculan los puntos de la curva.

Describiremos el método de construcción recursivo conocido como **algoritmo de De Casteljau**.

Para dos puntos, la curva es un segmento recto, definido en forma paramétrica por interpolación de los puntos extremos: $P = (1-u)P_0 + uP_1$. Donde los P_i son los puntos de control y $u \in [0,1]$ el parámetro.



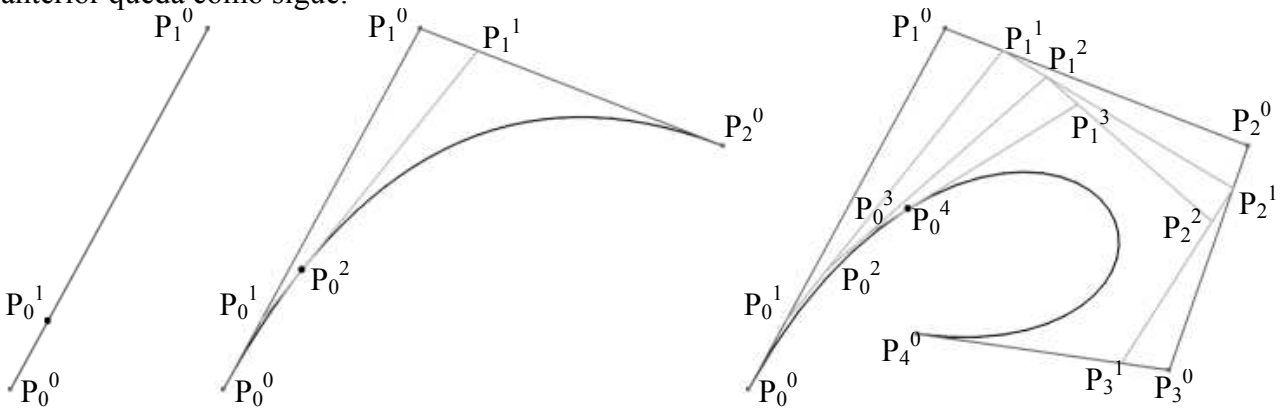
Agregando un punto de control más la curva adquiere más interés: se interpola linealmente en cada uno de los segmentos y luego entre los puntos resultantes, siempre con el mismo valor del parámetro. Para más puntos de control, el proceso se repite en forma iterativa.

La poligonal que forman los puntos de control se conoce como **polígono de control**.

Cambiemos ligeramente la notación para favorecer la interpretación de las propiedades y el código fuente. Indicaremos con un superíndice el nivel de iteración: llamemos P_i^0 a los puntos de control dados (nivel 0) y aumentemos en uno el superíndice con cada iteración:

$$P_i^{j+1} = (1-u)\, P_i^j + u\, P_{i+1}^j$$

Con esta notación, si hay n+1 puntos de control (0 a n) el punto de la curva es el P_0^n, y el dibujo anterior queda como sigue:



Por razones históricas se llama **orden** de una curva de Bézier a la cantidad de puntos de control. Para dos puntos de control, la curva es de segundo orden y primer **grado**, por cada punto de control que se agrega, se agrega además un paso de interpolación, en donde los términos en u quedan multiplicados por u o (1-u) y por lo tanto se aumenta en uno el grado del polinomio. Nosotros no utilizaremos el orden sino el grado pero es necesario definirlo porque OpenGL y algunos textos aun lo utilizan.

$$o = n + 1 \qquad \text{(orden = grado +1)}$$

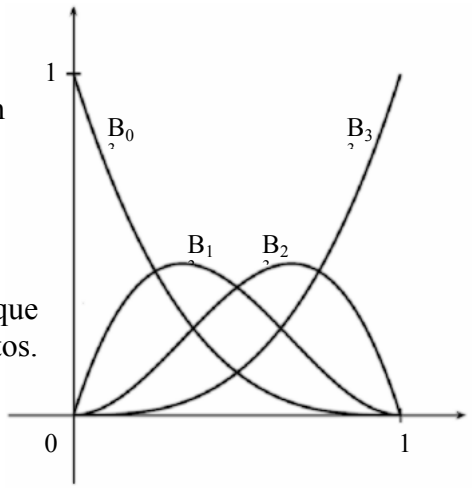
A modo de ejemplo, veamos el desarrollo para una curva de 3er grado (recordar que el superíndice de u indica potencia):

$$\begin{aligned} P_0^3 &= (1-u)\, P_0^2 + u\, P_1^2 \\ &= (1-u)\,[(1-u)\, P_0^1 + u\, P_1^1] + u\,[(1-u)\, P_1^1 + u\, P_2^1] \\ &= (1-u)^2\,[(1-u)\, P_0^0 + u\, P_1^0] + 2(1-u)u\,[(1-u)\, P_1^0 + u\, P_2^0] + u^2\,[(1-u)\, P_2^0 + u\, P_3^0] \\ &= (1-u)^3\, P_0^0 + 3(1-u)^2 u\, P_1^0 + 3(1-u)u^2\, P_2^0 + u^3\, P_3^0 \end{aligned}$$

Esa es la forma en que se realiza el cálculo recursivo de sucesivos puntos de la curva.

En general, la bibliografía utiliza la notación anterior, con el punto variable como una función paramétrica de los puntos de control:

$$P = \sum_0^n B_i^n P_i$$



Los coeficientes B_i^n son los **polinomios de Bernstein**, que pueden ser considerados como la base de un espacio donde el punto se identifica mediante n+1 coordenadas (vectoriales y no escalares) que son los puntos de control.

$$B_i^n = C_i^n (1-u)^{n-i} u^i$$

Los C_i^n son los números combinatorios: la cantidad de formas en que se pueden elegir i elementos distintos de un conjunto de n elementos. Se definen mediante:

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

Esa fórmula permite analizar las propiedades pero es inadecuada para el cómputo por la presencia de factoriales, en su lugar se utiliza el triángulo de Pascal y se calculan en forma recursiva.

En la figura se representan los cuatro polinomios de Bernstein de tercer grado. Puede verse la **simetría** alrededor de u = 0.5, que podría deducirse de las simetrías de los números combinatorios. La simetría implica que se puede revertir la lista de puntos de control y la curva resultará igual.

Cada punto de control se corresponde con un polinomio de Bernstein que actúa entonces como función de forma, aunque en este contexto suelen denominarse ***blending functions*** (funciones mezcladoras) y miden la influencia de cada punto de control en cada punto de la curva. B_0^n toma el valor 1 en u = 0 mientras que $B_n^n = 1$ en u = 1 (en la figura, con polinomios de tercer grado, n=3), eso implica que la curva interpola (pasa por) los puntos de control inicial y final. Además ningún B toma el valor unitario en ningún otro valor de u, por lo que P_1 y P_n son los únicos puntos de control interpolados.

La curva entera cambia al mover un solo punto de control, el punto variable es afectado por todos los puntos de control, esto se llama **control global**. Esta es una propiedad indeseable en CAGD, donde siempre se pretende el **control local** de la curva, es decir que la curva varíe sólo en las cercanías del punto de control movido. Más adelante veremos como se arregla esto con las B-Splines.

Un punto de la curva es combinación afín convexa de los puntos de control, Podemos verlo de dos modos: a) Por ser una cadena de combinaciones convexas b) Mediante el binomio de Newton, podemos ver que los polinomios de Bernstein suman uno.

$$\sum B_i^n = \sum C_i^n (1-u)^{n-i} u^i = [(1-u) + u]^n = 1$$

Cualquier punto (y por lo tanto, toda la curva) estará entonces dentro o en el límite del ***convex-hull*** de los puntos de control. La curva de Bézier, es combinación afín convexa de los puntos de control.

Siendo una combinación afín, tienen **invariancia afín**. Como ya hemos visto, la transformación afín mantiene las combinaciones afines, **para transformar la curva solo hay que transformar sus puntos de control y construirla**, en lugar de tener que transformar una miríada de puntos de la curva.

Unicidad: Dada una curva de Bézier de grado n hay un solo conjunto de n+1 puntos de control que la definen. La demostración se hace por el absurdo, y requiere una demostración de la independencia lineal de los polinomios de Bernstein, que es la de las series de potencias.

Derivadas:

La derivada dP(u)/du representa cuanto cambia el punto al variar el parámetro, es un vector tangente a la curva en el punto en que fué calculada. Todas las derivadas son vectoriales. La primera es la velocidad de recorrido si el parametro es el tiempo, la segunda indica la aceleración y apunta hacia donde se curva la línea, la tercera indica el cambio del plano de la curva.

Siendo polinómicas, las curvas de Bézier tienen continuidad C^∞; es decir que todas sus derivadas existen y son continuas. (Recordemos que C^n indica que las primeras n derivadas existen y son continuas, si n es cero indica que la función es continua).

Para calcular las derivadas basta con derivar los polinomios de Bernstein y se llega fácilmente, a través de la definición del número combinatorio a:

$$dB_i^n/du = n\,(B_{i-1}^{n-1} - B_i^{n-1}) \quad \forall\, i \in [1, n-1]$$

Con los casos especiales cuando el índice da -1 o mayor que el grado:

$$dB_0^n/du = -\,n\,B_0^{n-1}\;;\quad dB_n^n/du = n\,B_{n-1}^{n-1} \qquad \text{(o bien, se considera: } B_{-1}^k = B_{k+1}^k = 0\text{)}$$

$$dP/du = n \sum_0^n (B_{i-1}^{n-1} - B_i^{n-1})\,P_i$$

Alterando los subíndices y utilizando los definidos nulos:

$$dP/du = n \sum_0^n B_{i-1}^{n-1}\,P_i - n \sum_0^n B_i^{n-1}\,P_i = n \sum_0^{n-1} B_i^{n-1}(P_{i+1} - P_i) = n \sum_0^{n-1} B_i^{n-1}\Delta_i$$

Que puede verse como una curva de Bezier de un grado menos (n-1) y definida mediante puntos de control que son n veces las diferencias entre puntos sucesivos. La propiedad del *convex-hull* sirve aquí para acotar las derivadas (variaciones) de la curva. En particular, se conoce como ***variation diminishing*** o variación disminuida al hecho de que la curva no tiene más oscilaciones (*wiggles*) que su polígono de control. No daremos aquí la demostración pero, en 2D, cualquier línea que corta la curva en k puntos cortara k o más veces al polígono de control. Lo mismo sucede en 3D con un plano que corta la curva y el polígono.

En la fórmula anterior tenemos las derivadas como una función paramétrica de los puntos de control (de sus diferencias), pero podemos también derivar localmente y en forma recursiva:

Para una curva de 1er grado:

$$dP_0^1/du = d[(1-u)\,P_0^0 + u\,P_1^0]/du = P_1^0 - P_0^0 = \Delta_0^0$$

Para una de segundo grado:

$$\begin{aligned} dP_0^2/du &= d[(1-u)\,P_0^1 + u\,P_1^1]/du = \Delta_0^1 + (1-u)\,dP_0^1/du + u\,dP_1^1/du = \\ &= \Delta_0^1 + (1-u)\,(P_1^0 - P_0^0) + u\,(P_2^0 - P_1^0) = \\ &= \Delta_0^1 + [(1-u)\,P_1^0 + u\,P_2^0] - [(1-u)\,P_0^0 + u\,P_1^0] = \Delta_0^1 + P_1^1 - P_0^1 \\ &= 2\Delta_0^1 \end{aligned}$$

Si continuamos con el mismo proceso o hacemos inducción podemos ver que:

$$dP_0^n/du = dP/du = n\,\Delta_0^{n-1} = n\,(P_1^{n-1} - P_0^{n-1})$$

De donde podemos ver que el último segmento de la construcción de De Casteljau es **tangente** a la curva. En particular, para u=0 y para u=1 puede verse que la curva es tangente al polígono de control.

Reparametrización:

Para poder representar curvas con un parámetro t variando en cualquier intervalo real $[t_0, t_1]$, hace falta realizar una **reparametrización** afín con un parámetro u en [0, 1]:

$$t \in [t_0, t_1] \quad \Rightarrow \quad t = (1-u)t_0 + ut_1 = t_0 + u(t_1 - t_0) \quad \Rightarrow \quad u = \frac{t-t_0}{t_1-t_0} \in [0,1] \quad \Rightarrow \quad 1-u = \frac{t_1-t}{t_1-t_0}$$

La curva es la misma con u o t. Las derivadas cambian de forma sencilla:

$$\frac{d}{dt} = \frac{du}{dt}\frac{d}{du} = \frac{1}{t_1-t_0}\frac{d}{du} = \frac{1}{\Delta t}\frac{d}{du} \qquad\qquad \frac{d}{du} = \frac{dt}{du}\frac{d}{dt} = \Delta t\,\frac{d}{dt}$$

Notar que cuando observamos una curva determinada, no podemos decir con qué velocidad ha sido dibujada. Por ejemplo: un segmento de recta se puede dibujar desplazando el lápiz con velocidad constante, en este caso el punto para t=0.5 se encontrará justo en la mitad del segmento. Pero también se puede dibujar variando la velocidad (moviendo el lápiz cada vez más rápido, por ejemplo) y en este caso el punto para t=0.5 ya no se encuentra a igual distancia de los extremos.

Una reparametrización (ni lineal, ni afín) que haga que la velocidad de dibujo sea constante es la reparametrización por longitud de arco. Para una curva P(t) estándar, si "vista al microscopio es recta" (entorno de P(t) homeomorfo con R en un dado t), la longitud de arco se define mediante:

$$ds^2 = dP \cdot dP = dx^2 + dy^2 + dz^2 \quad \Rightarrow \quad (ds/dt)^2 = (dP/dt)^2 = P'^2 \quad \Rightarrow \quad s(t) = \int_{t_0}^{t} \sqrt{P'^2}\;dt$$

Esa reparametrización en particular es la más significativa, pero se puede ver que no es fácil de realizar y, en general, ni siquiera es posible.

Las curvas (y superficies) tienen algunas propiedades geométricas intrínsecas, que no dependen de la parametrización y son objeto de estudio de la Geometría Diferencial. Sólo nos importará aquí saber que para las curvas (pensar en un resorte) hay un vector unitario tangente en P que representa la velocidad intrínseca, una circunferencia osculadora, tangente a la curva en P, con el centro y el radio instantáneo de curvatura y que está relacionada con la segunda derivada; además, el cambio del plano de la circunferencia está indicado por otro vector, la torsión, relacionado con la tercera derivada.

Subdivisión:

La secuencia de puntos $P_0^i(u^*)$, para un dado $u^* \in (0,1)$ fijo, sirve como polígono de control para el tramo de la curva entre 0 y u^*. Es decir que cualquier punto de la curva sirve para dividirla en dos curvas de Bézier, definiendo los puntos de control intermedios por medio de los puntos calculados en el algoritmo de De Casteljau.

Hagamos la reparametrización de $[0,u^*]$ con $t \in [0,1]$, es decir que si en la curva original tenemos un valor de $u < u^*$, en la nueva tendremos $t = u/u^*$.

Para demostrar que cada tramo coincide con la curva original, en el caso general se puede demostrar fácilmente que P(u)=P(t(u)). Para el caso de las curvas de hasta tercer grado se ve directamente en el dibujo, porque los extremos y las derivadas extremas alcanzan para definir la curva. La curva original tenía por derivada respecto de u en P_0^0 al vector $3(P_1^0-P_0^0)$; ahora, respecto de t, es $3(P_0^1-P_0^0)$. Dado que $t=u/u^*$ las derivadas se relacionan mediante $u^*d/dt=d/du$, y también los vectores están relacionados según $(P_0^1-P_0^0)=u^*(P_1^0-P_0^0)$; las derivadas son iguales. Por simetría, lo mismo sucede para el tramo final. Siendo iguales los extremos y las derivadas, la curva coincide con la nueva.

En la figura de la derecha se muestra que el *convex-hull* de cada tramo se reduce. No lo demostraremos aquí, pero el "ancho" del *convex-hull* se reduce cuadráticamente con el proceso de subdivisión. La subdivisión recurrente es el método natural para encontrar la intersección de la curva con una línea o para rasterizar la curva. Las subdivisiones se realizan mientras el "ancho" del *convex-hull* sea mayor que un determinado valor, en un caso el error admisible y en el otro un píxel.

Splines por unión de curvas de Bézier. Continuidad Geométrica y Paramétrica:

Las curvas de Bézier son adecuadas para representar tramos aislados de una curva libre cualquiera. Para representar la curva entera, se puede hacer una unión "suave" de varias curvas de Bézier. La unión suave de curvas independientes se denomina spline.

Consideraremos que la curva entera, una *piecewise Bézier curve* o curva de Bézier por tramos, tiene un único parámetro que varía en forma continua entre tramos. Es decir que el parámetro inicial de cada tramo es igual al parámetro final del tramo anterior. El conjunto (arbitrario pero creciente) de estos parámetros límite se conoce como *knot vector* o vector de nudos.

Cuando analizamos la continuidad o suavidad en curvas definidas por tramos, estamos analizando en realidad la continuidad en los puntos donde se unen los distintos tramos, ya que en el interior la continuidad es C^∞. El grado de suavidad requerido depende de las necesidades. Por ejemplo, para la visualización de una curva, normalmente basta con que las partes tengan la misma recta tangente en el punto de unión, pero no es necesario que la derivada respecto del parámetro unificado sea continua (C^1), siendo vectores, solo basta con que sean proporcionales y vayan en el mismo sentido. Esta necesidad define la continuidad geométrica G^1. Este tipo de continuidad es menos restrictiva que la paramétrica y puede verse como una continuidad intrínseca, independiente de la parametrización, en la que se ignora el módulo de la velocidad de la curva.

En general, podemos decir que hay continuidad geométrica G^n cuando hay continuidad C^n en una reparametrización por longitud de arco. Pero, sabiendo las dificultades con la parametrización por longitud de arco diremos que deben variar suavemente los vectores derivados intrínsecos.

En el caso particular de G^1 bastará con que la velocidad no se anule y que en el punto de unión, la velocidad final de un tramo tenga la misma dirección y sentido que la inicial del otro tramo.

En general, la continuidad visual es G^1, pero cuando lo que importa o lo que se ve es la "velocidad" con que varia una función (posición, color, normal, etc.) se requiere continuidad paramétrica C^1 para que sea "visualmente placentero". Para una animación, si la cámara se mueve con continuidad paramétrica C^1, no se notarán cambios bruscos de velocidad de la cámara. Cuando se diseña una autopista, por ejemplo, se requiere continuidad G^2 para que no haya cambios bruscos de la curvatura y por lo tanto de la fuerza centrífuga.

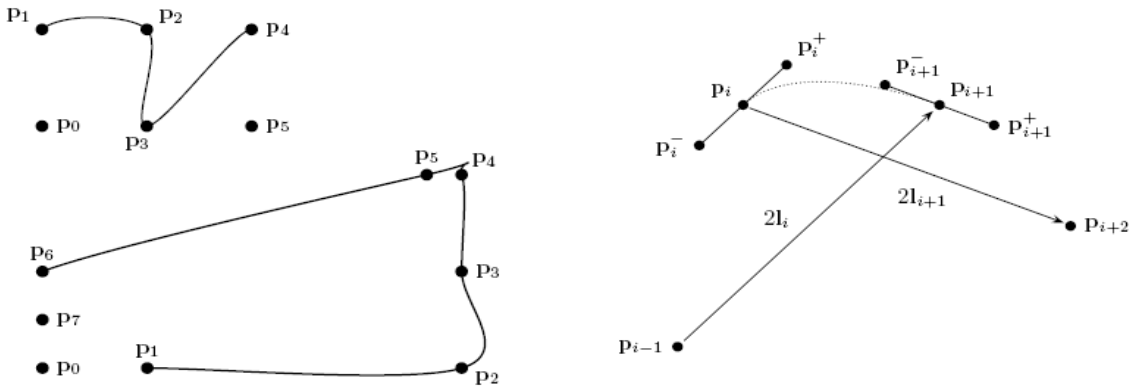
Volviendo a la unión de curvas de Bézier, para lograr la continuidad G^1 sólo se requiere que el segmento inicial del polígono de control de cada tramo sea colineal, contiguo y opuesto al segmento final de la curva anterior, independientemente del grado de cada tramo. La continuidad C^1 (**idénticas** derivadas) es más restrictiva y requiere que los segmentos, multiplicados por el grado, sean iguales y opuestos a cada lado de la juntura.

La relación entre la curvatura y la posición de los puntos de control es mucho más complicada, esperemos hasta las B-splines para lograr continuidad C^2 en forma sencilla.

Interpolación con curvas de Bézier:

Podemos interpolar puntos mediante las splines de Catmull-Rom, que son una sucesión de curvas de Bézier de tercer grado, unidas con continuidad paramétrica C^1.

Las splines de Catmull-Rom, interpolan una lista de m+1 puntos P_i, a excepción de los puntos inicial P_0 y final P_m.



Entre cada par de puntos intermedios P_i y P_{i+1}, se define una curva de Bézier mediante puntos de control agregados que garanticen la continuidad paramétrica. La idea es hacer que la derivada en cada punto sea la "velocidad media" que se requiere para ir del punto anterior al posterior. (Por eso no se interpolan los extremos). El parámetro se hace variar en forma continua entre 0 y m, tomando el valor i cuando la curva pasa por P_i, es decir que $P(i)= P_i$. En los puntos dados hacemos:

$$l_i = \frac{dP(i)}{du} = \frac{P_{i+1} - P_{i-1}}{(i+1)-(i-1)} = \frac{P_{i+1} - P_{i-1}}{2}$$

Para lograr esa derivada, ubicamos dos nuevos puntos de control: P_i^- y P_i^+ definidos mediante:

$$P_i^- = P_i - l_i/3 \qquad P_i^+ = P_i + l_i/3$$

Como se puede ver fácilmente, aquí los puntos tienen **control local**, el movimiento de un punto interpolante afecta a lo sumo a cuatro tramos de la curva.

Habiendo asignado la misma velocidad media a ambos tramos, es posible que para un tramo muy corto la curva "no tenga tiempo de frenar" (*overshoot*) y genere un rulo. Se ve un ejemplo entre P_4 y P_5 en la figura. Cuando hay tramos de distinta longitud (y por lo tanto, en general) es preferible resignar la continuidad paramétrica y obtener una curva G^1, utilizando un factor de corrección para la velocidad en cada tramo (que suele ser la longitud del tramo). Hay varias formas de hacerlo y varios nombres para el resultado, el mas conocido es el de Overhauser. También hay otras formas de definir la dirección de la recta tangente en los puntos de unión.

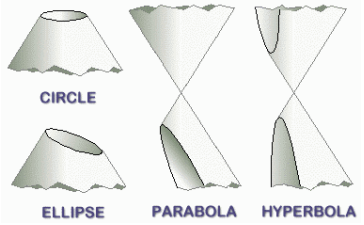
Curvas de Bézier racionales:

Las curvas racionales son el resultado de la proyección perspectiva de una curva con una dimensión más. La curva se define en coordenadas homogéneas R^{d+1} {x(u), y(u), z(u), w(u)} pero se dibuja en el espacio proyectivo P^d dividiendo por el polinomio w(u); de ahí la denominación de "racional": son cocientes de polinomios.

Existen dos justificaciones para utilizar curvas (y superficies) racionales.

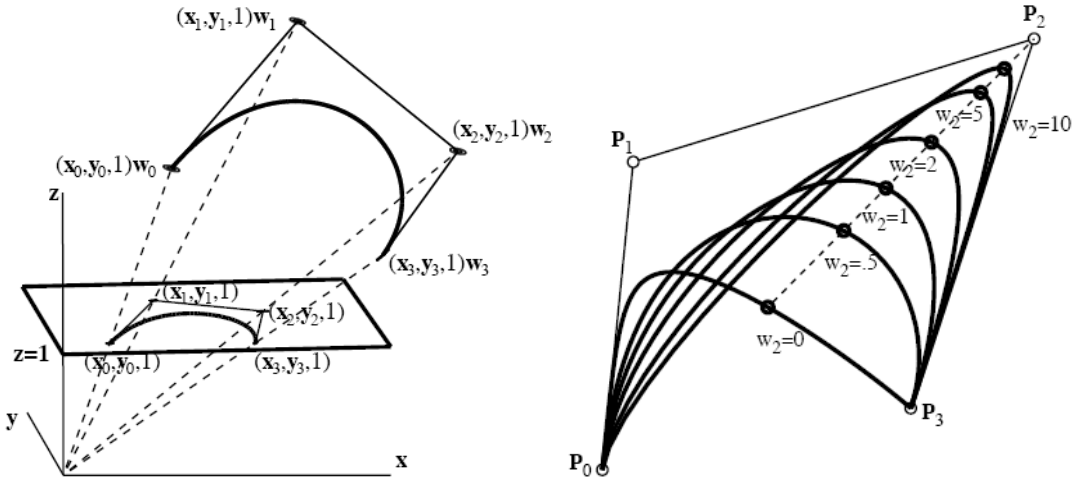
La primera es que las curvas así definidas poseen invariancia proyectiva. Recordemos que las curvas definidas mediante combinación afín tienen invariancia afín y por lo tanto lineal, y recordemos también que las transformaciones proyectivas son transformaciones lineales en una dimensión superior. Por lo tanto, una curva definida en una dimensión superior se puede transformar linealmente en R^{d+1} mediante la transformación (lineal) de sus puntos de control, para realizar la división perspectiva por w(u) al final del proceso.

La otra justificación (la principal, quizás) es la posibilidad de dibujar curvas cónicas, la más importante es la circunferencia, aunque para distintas aplicaciones técnicas se requieren a veces arcos de hipérbola, parábola o elipse.



Las curvas cónicas se obtienen al cortar un cono por medio de un plano (que en general no pasa por el vértice, ni es tangente a la superficie). Dependiendo del ángulo se obtienen las distintas curvas. Si el plano es perpendicular al eje se obtiene una circunferencia, si comienza a inclinarse respecto del eje se obtienen elipses de excentricidad cada vez mayor, hasta que, al ser el plano paralelo a la generatriz, se obtiene una parábola, para una inclinación mayor se obtienen hipérbolas.

Las curvas de Bézier de segundo grado (no-racionales) son siempre parábolas y no hay ninguna forma de que una curva de Bézier (de cualquier grado) represente una cónica diferente.



Las curvas racionales se pueden definir sin dificultad en coordenadas homogéneas:

$$P^* = \sum B_i^n P_i^*$$

El problema es que le estamos imponiendo al usuario una notación complicada: los puntos de control $P_i^* = \{P_i,1\}w_i = \{w_ix_i,w_iy_i,w_iz_i,w_i\} = \{w_iP_i,w_i\}$ tienen cuatro dimensiones que el usuario no ve ni necesita entender. Una interfaz grafica adecuada debe pedir al usuario que defina los puntos de control que ve: $P_i = \{x_i,y_i,z_i\}$, mediante las coordenadas que el ve (las proyectivas son wP) y esconder el carácter homogéneo mediante un número adicional w_i que funciona como peso, aunque sabemos que es la coordenada w de P^*. En OpenGL, si se interpreta que el usuario quiere un punto "en {x,y,z} y con peso w", el punto definido en el espacio homogéneo debe ser glVertex4d(wx,wy,wz,w); en cambio, si se interpreta que quiere definir un punto en el infinito "en dirección de {0,0,0} a {x,y,z}", debe ser definido mediante glVertex4d(x,y,z,0).

El punto variable de la curva proyectada en tres dimensiones es:

$$P = \frac{\sum B_i^n w_i P_i}{\sum B_j^n w_j} = \sum \frac{w_i B_i^n}{\sum B_j^n w_j} P_i$$

A la izquierda se ve claramente el sentido de "peso" adicional que se da en muchos textos a la coordenada w y justifica la denominación w por *weight*. A la derecha podemos ver que las curvas de

Bézier racionales siguen siendo combinación afín (y convexa, si los pesos son positivos) de los puntos de control proyectados y posee las propiedades que se derivan de ello (realice el análisis).

Se puede hacer una extensión del algoritmo de De Casteljau con pesos:

$$P^{*}{}_i^{j} = (1-u)\, P^{*}{}_i^{j-1} + u\, P^{*}{}_{i+1}^{j-1} \quad \Rightarrow \quad P_i^{j} = \frac{(1-u)\, w_i^{j-1} P_i^{j-1} + u\, w_{i+1}^{j-1} P_{i+1}^{j-1}}{(1-u)\, w_i^{j-1} + u\, w_{i+1}^{j-1}}$$

El denominador es el peso del punto P_i^j, haciendo la expansión desde j-1 hasta 0 :

$$w_i^j = \sum C_i^j\, (1-u)^{j-i}\, u^i\, w_i = \sum B_i^j\, w_i$$

A diferencia del caso polinómico (no-racional) aquí sí hay peligro de división por cero si se admiten pesos nulos o negativos.

Las derivadas se dificultan por la regla del cociente, pero se puede deducir que:

$$\frac{d}{du} P_0^n = n \frac{w_0^{n-1} w_1^{n-1}}{(w_0^n)^2} (P_1^{n-1} - P_0^{n-1})$$

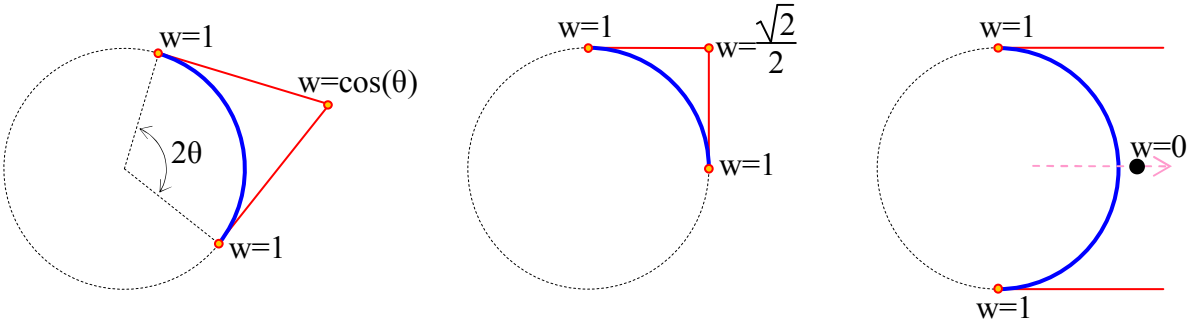
Expresión que en sí no interesa, pero sirve para mostrar que la recta tangente a la curva proyectada es la proyección de la recta tangente a la curva, esto es así en todo el proceso recursivo de De Casteljau. En particular, lo que importa es que las tangentes inicial y final están definidas, también en el caso racional, por los segmentos inicial y final del polígono de control.

$$\frac{d}{du} P_0^n(0) = n \frac{w_1}{w_0} (P_1 - P_0)$$

La excepción es cuando el denominador resulta nulo, donde habrá que hacer un análisis de límites.

Dados los puntos y las tangentes inicial y final de una curva de Bézier de segundo grado, el punto de control central estará en la intersección de las tangentes. Para una curva no racional ya está todo dicho, pero para una curva racional aún se puede modificar la curva mediante los pesos.

La figura muestra tres formas de construir arcos de circunferencia con curvas racionales de Bézier de segundo grado. La primera imagen muestra un método general, la segunda está particularizada para un cuarto de circunferencia y la tercera para media circunferencia. La del medio es la más utilizada.



Las tres muestran el polígono de control y los pesos asignados a los puntos. Como en toda curva de Bézier segundo grado, el punto de control del medio está en la intersección de las dos tangentes; en la figura derecha se puede ver que las tangentes se unen en el punto ideal.

Es importante saber que, como se aclaró anteriormente, hay dos formas de representar un punto de coordenadas {X,Y} con coordenadas homogéneas: si usamos {x,y,w}, el punto está en {X,Y}={x/w,y/w}; la forma más usual es {wX,wY,w} que muestra explícitamente la posición del punto, pero impide utilizar pesos nulos. En las figuras de los arcos de circunferencia, las dos primeras tienen los puntos de control definidos como {wX,wY,w}; pero en la tercera, aún cuando el centro no sea el origen de coordenadas, el punto de control en el infinito sólo puede representarse como {x,0,0}, con cualquier x mayor que cero, especificando una dirección desde el origen hacia la derecha.

En todos los casos, con radio 1, se puede demostrar fácilmente que $x^2(u)+y^2(u)=1$ es decir que efectivamente dan arcos de circunferencia.

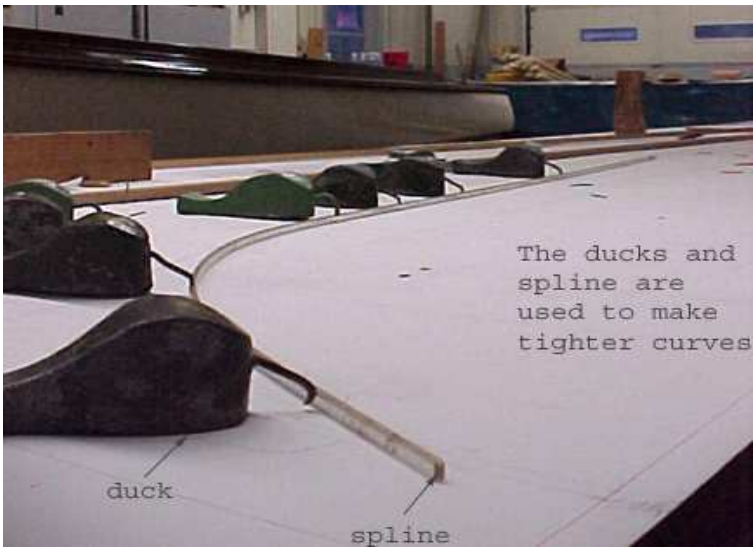
Obviamente no se puede definir una circunferencia completa con una sola curva de Bézier, pero con los mismos puntos de control y peso negativo para el central se puede trazar el otro tramo. En el caso de peso nulo, habrá que utilizar un vector hacia la izquierda para definir el punto central. Aún así, como regla general, no conviene confiar en que una determinada implementación de software soporte pesos nulos o negativos; OpenGL, por ejemplo, no garantiza resultados en tal caso. Conviene siempre utilizar tres o cuatro arcos (el mismo girado).

B-Splines

Una *spline* es simplemente una curva continua obtenida mediante unión suave de tramos curvos simples. Ya vimos un ejemplo con las splines de Catmull-Rom, que son unión C^1 de curvas de Bézier.

La B de B-splines se refiere a "base": se utiliza una base, en general polinómica, a diferencia de los intentos originales de simular una varilla flexionada mediante "puntos de control", usando las ecuaciones diferenciales de la elasticidad.

Las B-splines ganaron la batalla sobre todos los otros tipos definidos (¡muchos!) y hoy la variante: Non−Uniform Rational B−Spline o NURBS es la más utilizada.



Se trata de curvas no−interpolantes, en general cúbicas, unidas con continuidad C^2 y cuyos puntos de control poseen control local. Pueden pensarse como un método de unión de curvas de Bézier de grado n con continuidad C^{n-1}, pero ahorrándonos muchos puntos de control innecesarios como los que se introdujeron al interpolar con curvas de Bezier.

Normalmente, las B-splines y las NURBS se definen mediante las *blending functions* (como los polinomios de Bernstein para las curvas de Bezier) usando el algoritmo recursivo de Cox−De Boor para calcularlas. De ese modo, se dan una serie de recetas algebraicas, sin contenido geométrico.

Existen muchas formas de abordar el tema. Nosotros las estudiaremos mediante la **forma polar** y ***blossoming***, lo que implica un cambio de notación que parece bastante rebuscado, pero finalmente hace mucho más fácil "entender" el tema. Para programar, en cambio, cualquier receta es adecuada.

Blossoming y Forma Polar

La forma polar no es más que una extraña forma de notación para los polinomios.

Definamos una función *f* multiafín y simétrica, de n parámetros. Para no perdernos, fijemos en tres el número de parámetros:

- Por simétrica entendemos que no cambia si se permutan libremente los parámetros:
  $f(a, b, c) = f(a, c, b) = f(c, a, b) = \ldots.$
- Por multiafín entendemos que si todos los parámetros son iguales excepto uno, se puede interpolar linealmente entre los dos valores del parámetro distinto:
  $f(a, (1-u)b+uc, d) = (1-u)\, f(a, b, d) + u\, f(a, c, d)$

En la ultima ecuación, u es un parámetro (en principio, entre 0 y 1). La ecuación se lee así: *f* de a, un número entre b y c, d; es igual a la interpolación lineal de las (conocidas) *f* de a, b, d y *f* de a, c, d.

Se puede pensar *f* como una rutina que devuelve un real x si le damos tres reales {a,b,c}, los reales de entrada significan lo mismo: $f(3,1,5)$ devuelve lo mismo que $f(5,3,1)$. La rutina sólo puede calcular el resultado interpolando entre dos datos o resultados previos; ej.: si conozco $f(3,1,3)$ y $f(3,1,9)$ y quiero calcular $f(3,1,5)$, debo interpolar el 5 entre 3 y 9:

$$f(3,1,5) = 4/6\, f(3,1,3) + 2/6\, f(3,1,9) \qquad (9-3=6,\ \text{de 3 a 5 hay 2, de 5 a 9 hay 4})$$

La fórmula general, para interpolar $t \in [t_i, t_{i+1}]$ es una simple interpolación lineal:

$$f(a,b,c,\ldots,t) = \frac{(t_{i+1} - t)\, f(a,b,c,\ldots,t_i) + (t - t_i)\, f(a,b,c,\ldots,t_{i+1})}{t_{i+1} - t_i}$$

Los datos de partida son unos pocos valores conocidos de *f* y a partir de ellos se pueden calcular sucesivos valores mediante múltiples y sucesivas interpolaciones lineales, por lo que el resultado es un polinomio en el parámetro.

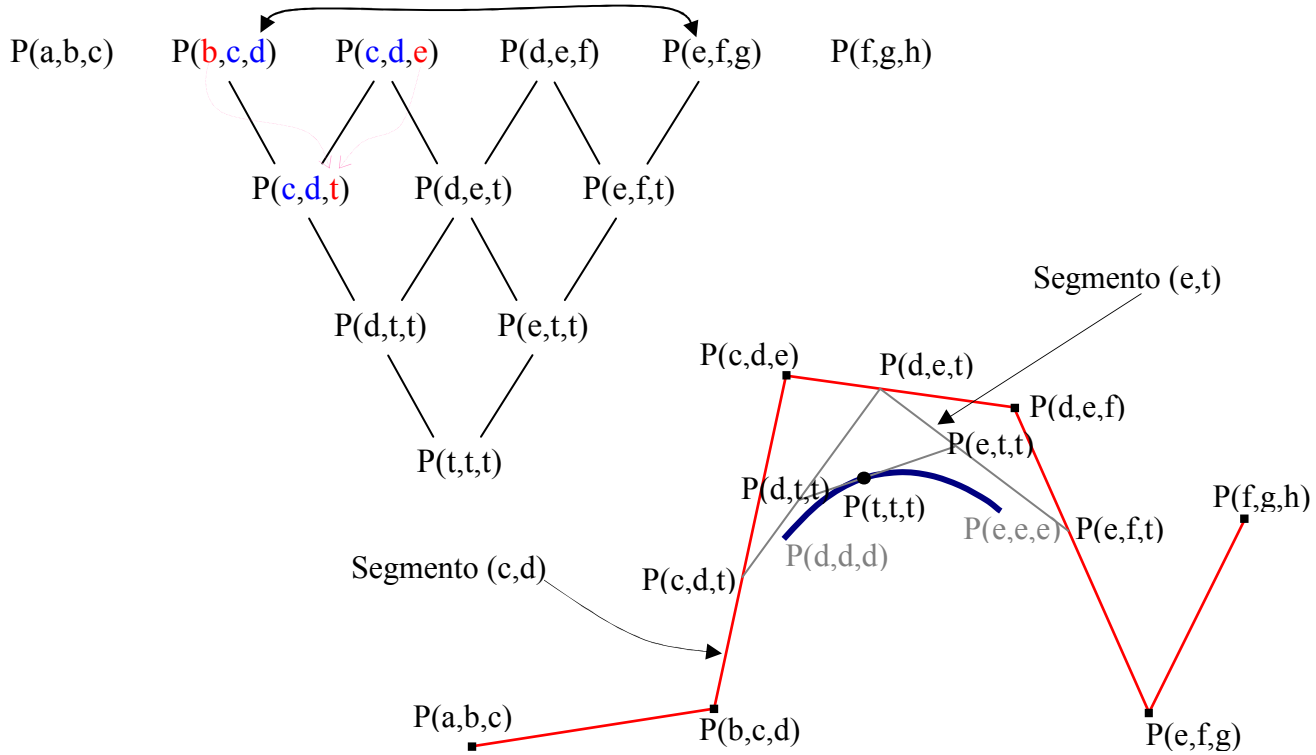
En nuestro caso el resultado no es un solo valor escalar *f* sino varios, un vector de coordenadas y datos reales {x,y,z(,w,R,G,B,temperatura,presión,…)}, cuyas componentes y valores asociados reciben todas el mismo tratamiento en simultaneo; pero consideremos que es simplemente un punto interpolado. El punto se obtiene mediante una sucesión de combinaciones afines de los puntos de control dados, es decir que es una combinación afín de los datos, cuyas funciones de forma o pesos son polinomios en función de un solo parámetro y por lo tanto define una curva unidimensional.

No desarrollaremos la matemática, solo vamos a usar la notación y las propiedades como una receta:

- P(t, t, t) es el punto en la curva a calcular.
- {a,b,c,d…} es un conjunto no-decreciente de valores del parámetro (a ≤ b ≤ c ≤ d ≤ …) que viene dado y llamamos *knot-vector* o vector de nudos (notar que pueden repetirse).
- Partimos de los puntos de control conocidos que llamaremos P(a, b, c), P(b, c, d), P(c, d, e)….

Blossoming es un método de cálculo del punto de la curva por medio de interpolaciones lineales recursivas de los puntos de control dados. El algoritmo de De Casteljau es un ejemplo.

El siguiente esquema muestra la marcha del cálculo necesario para obtener un tramo de una curva de tercer grado mediante *blossoming:*



En este ejemplo tenemos seis puntos de control y un vector de nudos {a,b,c,d,e,f,g,h}. Supongamos que queremos calcular el punto de la curva para un dado valor de t, comprendido entre d y e. En el primer nivel, correspondiente a los puntos de control, se interpola un punto en cada segmento cuyos extremos tengan a d o a e. Por ejemplo para calcular P(c,d,t) se utiliza la formula de arriba entre los dos puntos que contienen c y d: P(b,c,d) y P(c,d,e); el valor de t se interpola entre los no comunes:

$$P(c,d,t) = \frac{(e - t)\,P(b,c,d) + (t - b)\,P(c,d,e)}{e - b}$$

Del mismo modo, en cada nivel, se interpola con t entre dos puntos previos que tienen dos parámetros iguales y uno distinto, que para un punto esta por debajo de t y para el otro por encima. Ejemplo:

$$P(d,t,t) = \frac{(e - t)\,P(c,d,t) + (t - c)\,P(d,e,t)}{e - c}$$

El cálculo termina en P(t,t,t) que es el punto buscado de la curva de tercer grado. El cálculo hace primero una interpolación lineal entre dos puntos; es un polinomio lineal en t. Para el segundo nivel la interpolación de los puntos interpolados será un polinomio cuadrático y para el tercero resulta cúbico.

Los puntos de control sin d ni e no influyen en este tramo de la curva (control local), que va desde P(d,d,d) hasta P(e,e,e) que no son puntos de control (no−interpolante).

Para el primer tramo de la curva, t varía entre c y d; se calcula con los primeros cuatro puntos de control (orden=grado+1) que son P(a,b,c) a P(d,e,f). Del miSmo modo, para el último tramo, entre P(c,d,e) y P(f,g,h), el parámetro t varía entre e y f.

Notar:

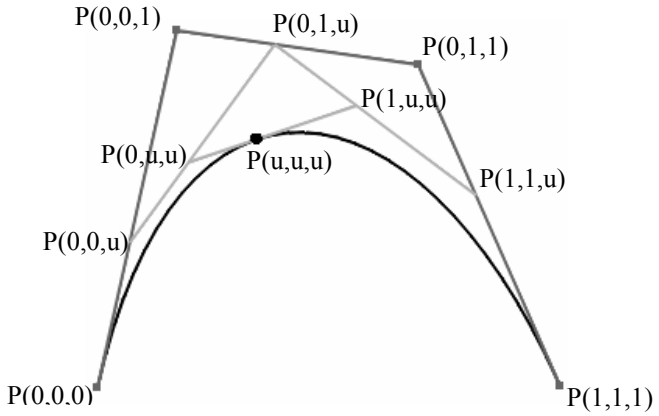
1) Si bien el rango de parámetros es [a,h], sólo hay puntos de la curva en [c,f], no se puede definir la curva (sin extrapolar) fuera de ese rango.
2) La curva sí depende de los puntos con parámetros entre a y g los valores extra definen "como arranca y como termina" la curva es decir, las condiciones de borde (valores y derivadas extremas)
3) El esquema anterior es para un tramo de la curva ($t \in [d,e]$) pero se puede extender al resto y más allá, tanto como se quiera. Agregando un punto de control más P(g,h,i) se obtiene un tramo más [f,g] y así sucesivamente. La "profundidad" del esquema se mantiene siempre igual (tres niveles de interpolación ≡ tercer grado).
4) Para calcular un punto en una curva de tercer grado: P(t,t,t), cuando hay mas de cuatro puntos de control, se toman los cuatro que contienen los valores que encierran a t (d y e en este caso).
5) Cada segmento recto (del polígono de control o interpolado) tiene dos parámetros constantes y uno variable, Por ejemplo el que une P(c,d,t) y P(d,e,t) puede denominarse "segmento (d,t)" y, en él, el argumento variable varía entre c y e (si trazamos la curva manualmente, dibujamos rayitas regularmente entre c y e y marcamos t)
6) Un punto de la curva correspondiente a un knot (ej.: P(d,d,d)) requiere una interpolación menos:

$$P(c,d,d) = \frac{(e - d)\,P(b,c,d) + (d - b)\,P(c,d,e)}{e - b} \qquad P(e,d,d) = \frac{(f - d)\,P(c,d,e) + (d - c)\,P(d,e,f)}{f - c}$$

$$P(d,d,d) = \frac{(e - d)\,P(c,d,d) + (d - c)\,P(e,d,d)}{e - c}$$

Para recordar y recalcar que los knots o argumentos o parámetros son simples números en secuencia no decreciente, comenzamos con ejemplos numéricos.

Un caso particular de spline es una curva de Bézier. Se caracteriza por el modo de repetición de los *knots*, en una de tercer grado el vector es {0,0,0,1,1,1} (para grado n, hay un knot repetido n veces y otro mayor, también n veces; pero pueden ser otros números, usamos 0 y 1 para no dividir por la diferencia). Los puntos de control serán P(0,0,0), P(0,0,1), P(0,1,1) y P(1,1,1). Los puntos inicial y final tienen los tres valores repetidos, por lo tanto son puntos de la curva. Calculemos un punto cualquiera de parámetro $u \in [0,1]$:



$$P(0,0,u) = (1-u)\,P(0,0,0) + u\,P(0,0,1) \qquad P(0,u,1) = (1-u)\,P(0,0,1) + u\,P(0,1,1) \qquad P(u,1,1) = (1-u)\,P(0,1,1) + u\,P(1,1,1)$$

$$P(0,u,u) = (1-u)\,P(0,0,u) + u\,P(0,u,1) \qquad P(u,u,1) = (1-u)\,P(0,u,1) + u\,P(u,1,1)$$

$$P(u,u,u) = (1-u)\,P(0,u,u) + u\,P(u,u,1)$$

Como vemos, es exactamente el algoritmo de De Casteljau.

La subdivisión (que vimos para curvas de Bézier) consiste en dividir la curva en dos partes, utilizando para ello los puntos de control recién calculados:

{P(0,0,0), P(0,0,u), P(0,u,u), P(u,u,u)} —— {P(u,u,u), P(u,u,1), P(u,1,1), P(1,1,1)}

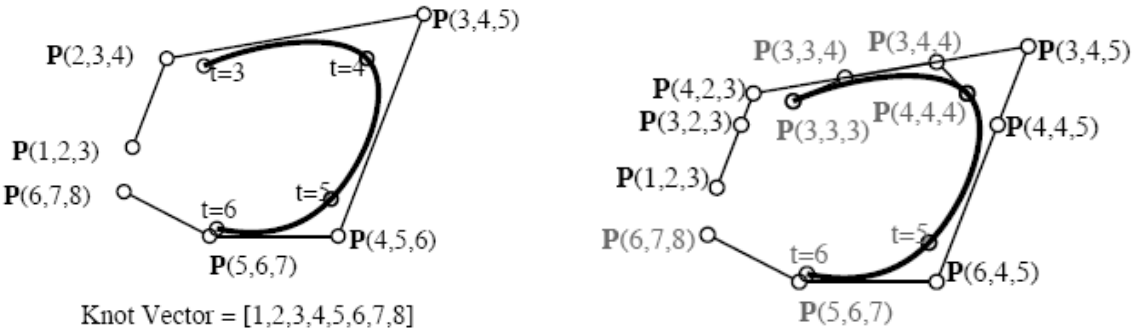
Se puede ver enseguida que ambos tramos están definidos del mismo modo como curvas de Bézier.

En las figuras que siguen se muestra una B-spline de tercer grado definida por el *knot-vector* {1,2,3,4,5,6,7,8} y los seis puntos de control {P(1,2,3), P(2,3,4), P(3,4,5), P(4,5,6) P(5,6,7), P(6,7,8)}. El punto variable de la curva es P(t,t,t) con t ∈ [3,6] (el último del primer PC y el primero del último).

La curva consta de tres segmentos con el parámetro variando en [3,4], [4,5] y [5,6] respectivamente.

Los puntos de control del primer tramo (tomado como curva aislada) son: {P(3,3,3), P(3,3,4), P(3,4,4), P(4,4,4)}, es decir que es una curva de Bézier. Todos los tramos de una B-spline son curvas de Bézier.

El primer punto P(3,3,3) se obtiene por interpolación de P(3,2,3) y P(3,3,4), que no vienen dados. Para obtener P(3,2,3) se interpola entre P(1,2,3) y P(2,3,4) (se interpola el 3 entre 1 y 4). El otro punto, P(3,3,4) se encuentra de igual manera, interpolando entre P(2,3,4) y P(3,4,5). Dos interpolaciones.



Ejemplo de cálculo: Obtenemos el punto P(3,2,3) interpolando entre P(1,2,3) y P(2,3,4). Dado que los dos números 2 y 3 son iguales, el parámetro variable es t=3, entre t_i=1 y t_{i+1}= 4:

$$P(3,2,3) = \frac{(4 - 3)\,P(1,2,3) + (3 - 1)\,P(2,3,4)}{4 - 1} = \frac{P(1,2,3) + 2P(2,3,4)}{3}$$

Puede deducirse de la forma de interpolación, que un tramo de la curva [t_i, t_{i+1}] está influenciado solamente por los puntos de control que tienen a t_i o a t_{i+1} como parámetros en la forma polar. Por ejemplo el tramo [4,5] está influenciado por los puntos de control que van desde P(2,3,4) a P(5,6,7), el primero que tiene al 4 es P(2,3,4) y el último que tiene al 5 es P(5,6,7).

Knot-Vector:

El vector de nudos es una secuencia no decreciente con valores del parámetro. En cada *knot*, una función base se anula y otra comienza a crecer; por lo tanto definen límites del soporte de las *blending functions*. Si la secuencia está equiespaciada ($k_{i+1} - k_i$ = cte.), la curva se denomina spline uniforme, en el caso contrario, será no-uniforme. La secuencia puede tener valores repetidos que se conocen como nudos múltiples o *multiple knots*.

Cada punto de control de una curva de grado n está identificado con n *knots* y en cada segmento hay un *knot* variable y dos fijos. Las variaciones del *knot* variable a izquierda y derecha de un punto de control, se relacionan con la velocidad con que gana o pierde influencia el punto. Supongamos un tramo de *knots* {…,2,3,4,5,6,…}, un segmento 3-4 con el parámetro variando entre 2 y 5 seguido por un segmento 4-5, con el parámetro variando entre 3 y 6; si en su lugar usamos {…,2,3,4,5,30,…}, el punto gana influencia más rápido y la pierde más lentamente (construir el ejemplo). Las variaciones anteriores eran 3 al llegar y 3 al salir (3/3), las nuevas, 3 y 27 respectivamente (3/27).

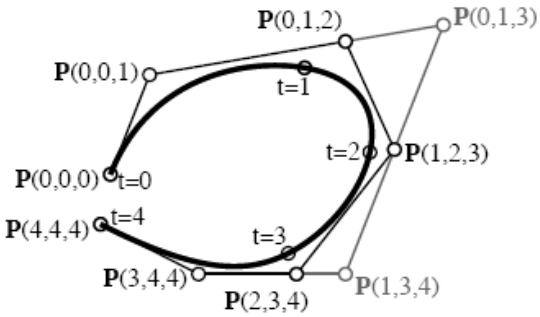
El vector de nudos se impone al definir la curva. En los programas de CAD suele ser transparente al usuario quien solamente define los puntos de control; el programa suele poner una secuencia uniforme de *knots* en forma automática, o bien puede espaciarlos en proporción a las distancias entre los puntos de control (recordar el *overshooting* en la interpolación con splines). El usuario suele intervenir solamente en los programas de animación o cuando debe controlar la velocidad del recorrido por la curva, las splines sirven ahí para definir las trayectorias de los objetos, la cámara y hacia donde mira.

Los puntos libres al principio y al final son para imponer las condiciones de borde: punto y derivadas de partida y llegada. Normalmente se suelen imponer **condiciones de borde de Bézier**: que la curva interpole los puntos extremos y sea tangente a los segmentos extremos. Para ello, en lugar de los puntos libres al principio y al final se utilizan los puntos de control de inicio del primer arco de Bézier y final del último. Por ejemplo, si al inicio de la curva en la figura de arriba, en lugar de P(1,2,3) y P(2,3,4) se usan P(3,3,3) y P(3,3,4). La secuencia de *knots* sería: {3,3,3,4,5,6,7,8}. Haciendolo también

al final nos quedará: {3,3,3,4,5,6,6,6} y los puntos de control {P(3,3,3), P(3,3,4), P(3,4,5), P(4,5,6), P(5,6,6), P(6,6,6)}, dando una spline no-uniforme que interpola sólo los puntos de control extremos.

Por razones históricas (basadas quizás en la estabilidad del cálculo de la base de polinomios) el software (OpenGL) y los libros utilizan un *knot* de más al principio y otro al final. Se suele poner una copia del primero y del último respectivamente. No tienen ninguna influencia en la forma de la curva.

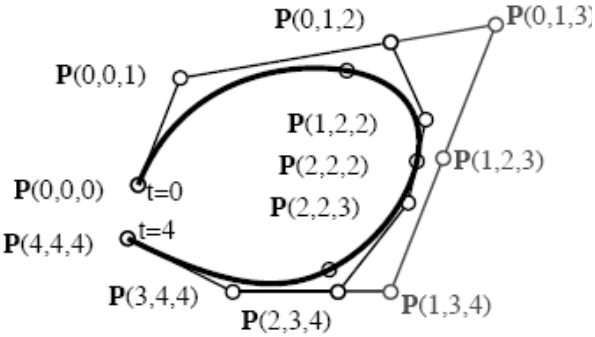
La **inserción de *knots*** se utiliza para "mejorar" una curva que se define primero en forma gruesa y se va refinando hasta adoptar la forma deseada. En el ejemplo de la figura, se parte de una B-spline y se inserta un nuevo nudo intermedio de valor 2. La curva de Bézier del intervalo [1,3] se subdividió en dos al insertar un nuevo *knot*. El polígono de control se reemplaza localmente con uno más "refinado". Para un valor arbitrario del parámetro habrá que hacer las cuentas de la interpolación lineal. Los nuevos puntos de control pueden luego moverse libremente para reajustar la curva. La inserción de un nuevo nudo es para darle al usuario un nuevo punto para mover y así ajustar la curva; pero el nuevo punto debe insertarse de modo que no modifique la forma actual de la curva



Old Knot Vector = [0,0,0,1,3,4,4,4]
New Knot Vector = [0,0,0,1,2,3,4,4,4]

El **algoritmo de De Boor** se utiliza para calcular el punto variable de la curva y trazarla. Consiste en encontrar el punto P(t,t,t) y el método equivale a insertar n veces el nudo t. Para la curva del ejemplo anterior, insertando tres veces el nudo en 2 se obtiene el punto p(2,2,2) de la curva.

Este mismo truco es el que se utiliza para la **subdivisión**: se calcula un punto de la curva y los necesarios puntos de control al final del primer tramo y al principio del siguiente.



Old Knot Vector = [0,0,0,1,3,4,4,4]
New Knot Vector = [0,0,0,1,2,2,2,3,4,4,4]

Derivadas:

Las derivadas salen tediosa pero fácilmente, se hace lo mismo que en las curvas de Bézier pero allí el incremento del parámetro era uno. Aqui, si $t \in [t_i, t_{i+1}]$:

$$\frac{d}{dt} P(t,t,t) = n \frac{P(t,t,t_{i+1}) - P(t,t,t_i)}{t_{i+1} - t_i}$$ (Recordarlo como "n $\Delta P/\Delta t$")

No lo demostraremos pero es fácil de ver y muy importante conocer que **por cada repetición de un nudo se pierde un grado de continuidad** de la curva en el punto correspondiente. Este asunto es muy importante pues es la técnica que se utiliza para definir puntos de quiebre en el interior de una spline.

Como se explicaba, al insertar el nudo la curva no cambia, pero al mover uno de los nuevos puntos de control se nota la pérdida de suavidad. Por ejemplo, en el algoritmo de De Boor se introdujeron *knots* repetidos sin modificar la continuidad de la curva. Pero si el usuario mueve el punto P(t,t,t) arrastra la curva con continuidad C^0. En la figura anterior, si se mueve el punto P(2,2,2), la tangente a un lado será el segmento P(1,2,2,)-P(2,2,2) y al otro lado será P(2,2,2)-P(2,2,3), ya no serán coincidentes.

Para las NU**R**BS (racionales) las derivadas no son sencillas, por la regla del cociente; pero por ser el método de construcción igual al de Bezier, el resultado es similar y conceptualmente idéntico: en la medida en que no haya pesos nulos ni negativos las tangentes se proyectan.

Superfices

Existen muchas variantes útiles para construir superficies, la más sencilla de comprender es el **producto cartesiano o tensorial** de curvas de Bézier o de NURBS. Para definir una superficie como un producto cartesiano, debemos asumirla como una función de dos parámetros (u,v):

$$P(u,v) = \sum_i \sum_j B_i^n(u)\, B_j^m(v)\, P_{i,j} = \sum_i B_i^n \left(\sum_j B_j^m P_{i,j}\right) = \sum_j B_j^m \left(\sum_i B_i^n P_{i,j}\right)$$

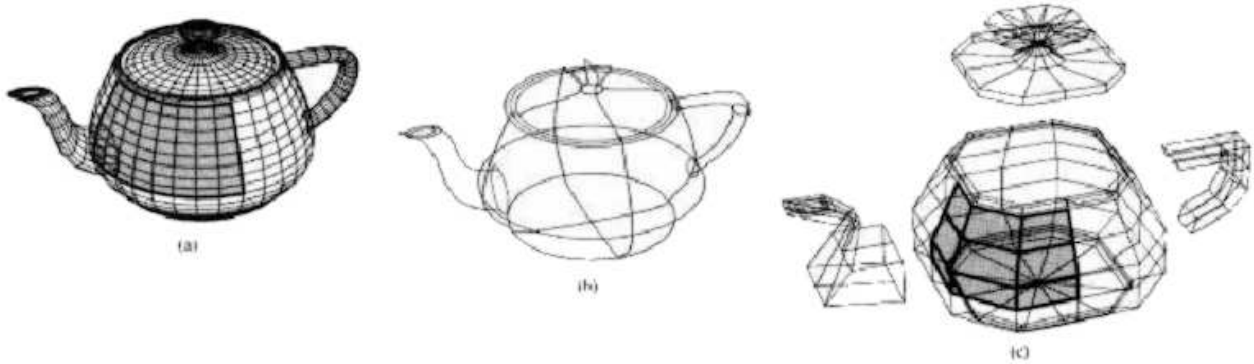
Para un dado valor del parámetro v fijo se obtiene una curva isoparamétrica con u variable, y lo mismo sucede si se fija u y se varía v. Las alternativas equivalentes de la derecha indican que la superficie puede pensarse como combinación de "curvas de control" (término inexistente), en el mismo sentido en que las curvas son combinaciones de puntos de control.

Los puntos de control de la superficie forman una **grilla rectangular** de (n+1) x (m+1) puntos que pueden ubicarse libremente en el espacio, pero están "interconectados" como en una cuadrícula. Para una superficie NURBS, deberá definirse además un vector de nudos en cada dirección.

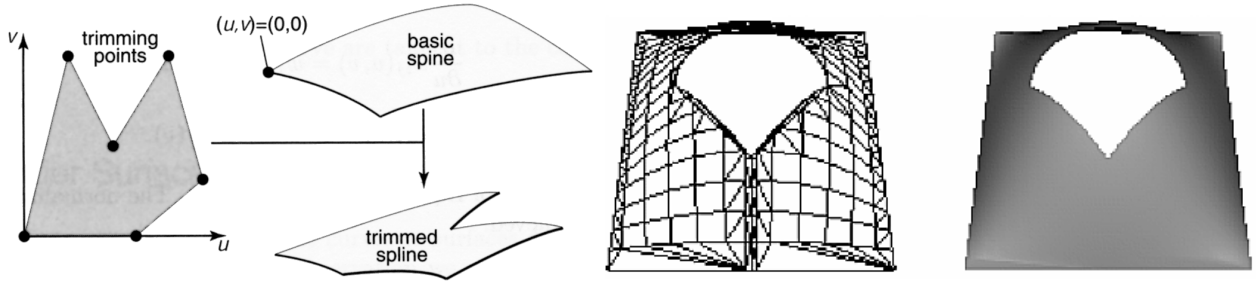
Si en una dirección las splines son rectas, se definen sólo los puntos de control de las dos curvas extremas, en ese caso tendremos una superficie reglada. Si ambas son rectas es bilineal. Si un conjunto de curvas son círculos coaxiales, es decir que los puntos de control de una curva "generatriz" se copian en un *array* circular alrededor de un eje, tendremos una superficie de revolución. Así hay varios casos especiales y útiles.

En las superficies, las condiciones de borde son mucho más importantes que para las líneas, porque al renderizar se verán efectos lumínicos indeseables si la normal no pasa suavemente de un tramo a otro.

La tetera, que tantas veces hemos visto (*Utah teapot*), está construida por medio de *patches* o parches de superficies de Bezier.



La unión del pico a la tetera no está realizada "correctamente", el pico atraviesa el cuerpo. Una solución a esto son las ***trimmed* NURBS**, que permiten recortar una superficie NURBS por medio de splines definidas en el **espacio de parámetros**: Una superficie NURBS está definida por medio de un arreglo rectangular de puntos de control en el espacio del dibujo (o el homogéneo) y dos *knot-vectors* que definen el rango en el espacio de los parámetros. El espacio de parámetros se representa en el plano (u,v) y es un rectángulo limitado por los *knots* extremos en cada dirección. Es allí donde se definen una o mas curvas planas que recortan el exterior y huecos en el rectángulo original. La superficie se renderiza sólo entre los límites que definen esas curvas.

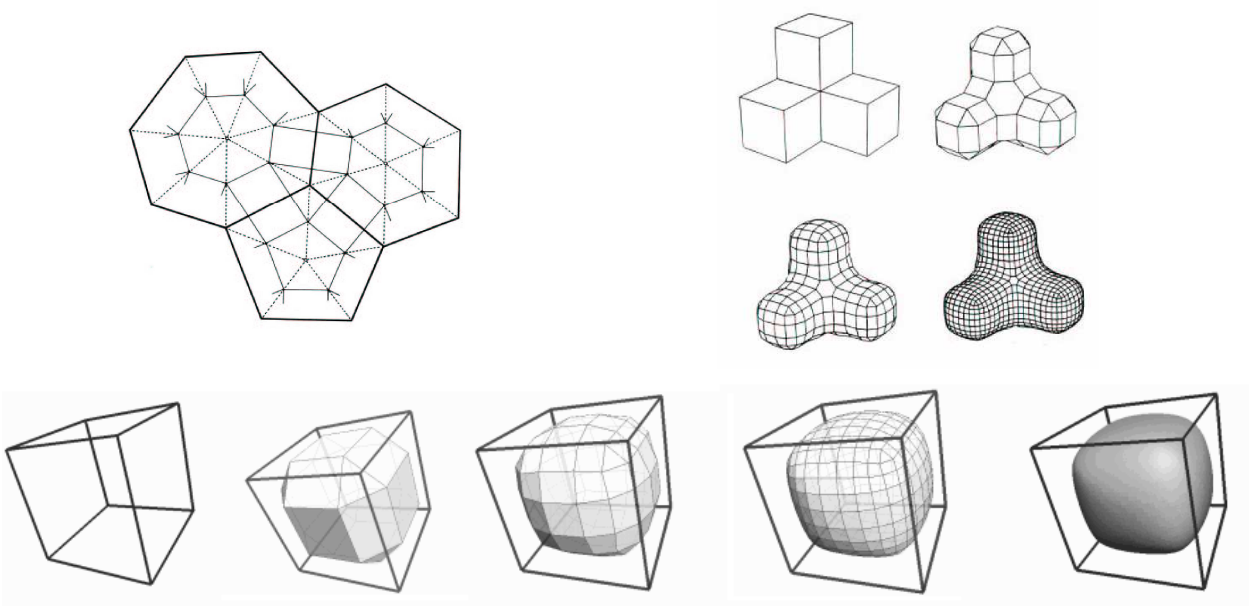


En la figura de arriba se muestra, a la izquierda un ejemplo de recorte exterior y a la derecha está el ejemplo del Red-Book que sólo tiene un recorte interior. Como puede verse se hace necesario renderizar una triangulación pues algunos cuadriláteros quedan recortados. También puede verse la pésima calidad de la triangulación que provee GLU y que se nota en los reflejos en la parte inferior.

Las *trimmed* NURBS proveen una gran ventaja para realizar las uniones, pero no solucionan todo el problema, pues el borde recortado, en el espacio, tiene una formulación muy complicada como para ser usada por otra superficie NURBS de grado no muy alto, por lo cual la mayoría de los programas suelen cometer errores en las uniones que pueden ser muy serios para otras aplicaciones.

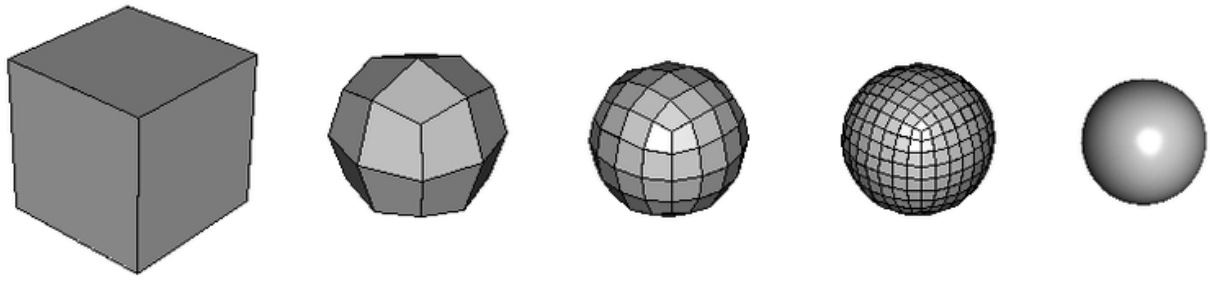
Hay muchos tipos de superficies, pero casi todos pueden traducirse o representarse como NURBS, a veces recortadas. De hecho la mayoría de los paquetes de software (al menos los más técnicos) transforman cualquier otra posibilidad en una NURBS para el intercambio de archivos. Hay también superficies de Bézier triangulares en vez de cuadriláteras, en ellas todo lo que hemos dicho sobre coordenadas cartesianas se reemplaza por coordenadas de área. Pero si hay algún candidato al reemplazo de las NURBS estas son las ***subdivision surfaces***, que tienen una interfaz de usuario más razonable, no requieren del molesto e incomprendido vector de nudos, y admiten cualquier distribución de puntos de control, no necesariamente una grilla rectangular.

La idea general de las *subdivision surfaces* es partir de una forma muy basta para refinarla en un proceso recursivo, cuyo límite es la superficie buscada. Hay muchos métodos para construirlas, veremos dos: Doo-Sabin y Catmull-Clark.



En el método de Doo-Sabin, que se muestra arriba, en cada paso se une el centro (promedio de vértices) de cada cara con cada vértice de la cara, luego se pone un punto en el centro de cada uno de los segmentos nuevos. La superficie original se reemplaza, en cada paso, por la que forman los nuevos puntos. Resulta una generalización de las superficies de Bézier de 2º grado.

El proceso de Catmull-Clark, debajo, es ligeramente distinto. Conserva los puntos originales pero los mueve. Es una generalización de las superficies de tercer grado con continuidad C^2.

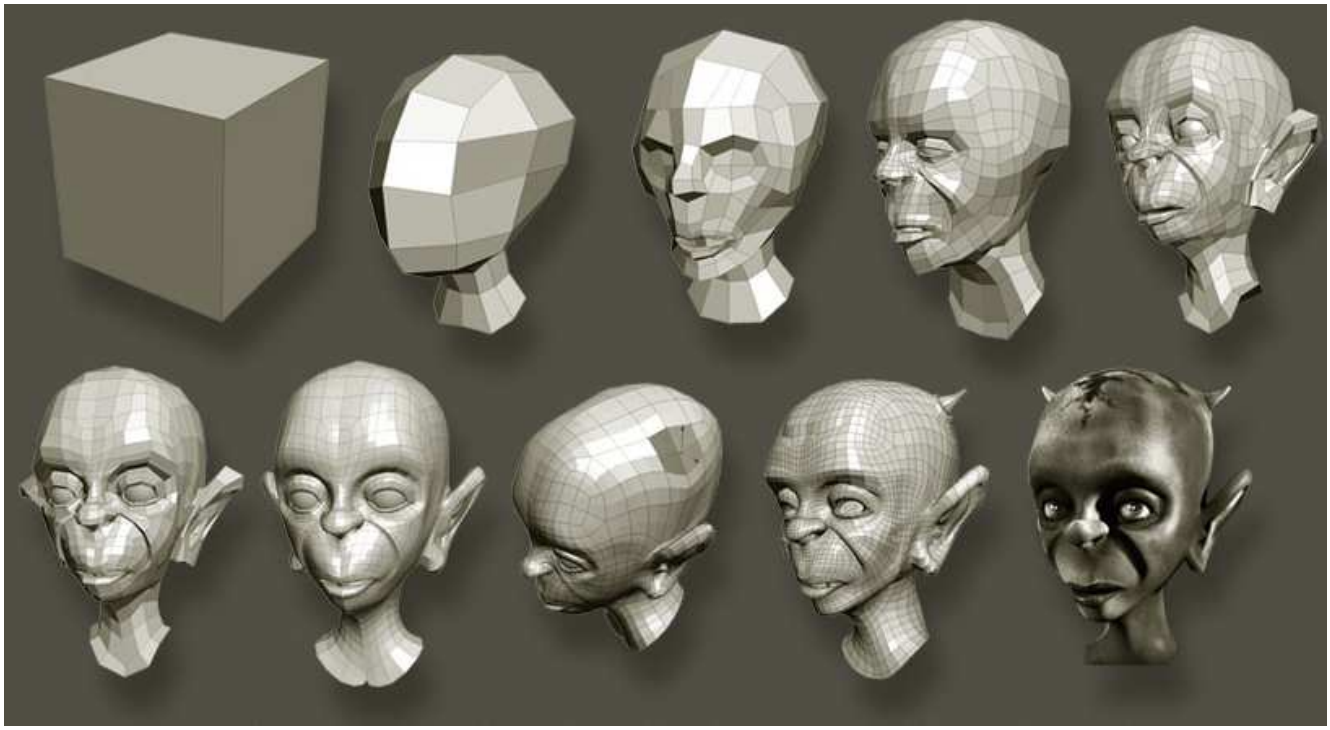


Para cada cara ponemos un punto en el centro. Para cada arista también. El punto de la arista se mueve al promedio entre el punto en que estaba y los dos centroides de las caras adyacentes en esa arista.

El vértice también se mueve pero es más complicado: Si llamamos P al punto original, F al promedio de los puntos centrales de las n caras que tienen a P como vértice, R al promedio de los puntos medios de segmentos conectados con P; la nueva posición del punto P será:

$$P' = \frac{F + 2R + (n-3)P}{n}$$ (notar la necesaria suma uno de los pesos)

Gracias a (o culpa de) la dificultad para unir superficies NURBS con continuidad geométrica, una buena parte del software que se esta desarrollando y utilizando en este momento, se basa en estos algoritmos. El método que se utilice depende fuertemente de la aplicación, pero en CAD y aún más para la creación de caracteres para animaciones, se están utilizando mucho las *subdivision surfaces*. La cantidad de métodos de subdivisión es más grande que la aquí mostrada y la teoría subyacente también es mucho más complicada, mucho más que para las NURBS.



Glenn Southern: (http://southerngfx.co.uk)