Modelado de Curvas y Superficies

La definición matemática de curva depende del contexto, en Computación Gráfica, y en general en CAGD (*Computer Aided Geometric Design*), **una curva es la imagen de una función continua desde un intervalo real al espacio euclídeo**. Resulta un conjunto continuo y unidimensional de puntos, cuyas coordenadas (dos o tres) varían como funciones continuas de un parámetro real.

Siempre se puede encontrar alguna chicana matemática que haga las cosas difíciles, por ejemplo $x = t$, $y = \sin(1/t)$, con el parámetro $t \in (0, 1]$. Simplemente ignoraremos esas cosas; nos concentraremos en la representación gráfica de las curvas útiles, sin elementos raros, sin infinitos vaivenes o puntas; pero es necesario aclarar que a veces esas rarezas generan cosas tan elegantes y útiles como las curvas y superficies fractales, que se utilizan ¡y mucho! en computación gráfica, ese caso sí lo mencionaremos.

Una superficie será un conjunto continuo de puntos; donde las coordenadas son también funciones "bonitas", pero de dos parámetros, es decir: bidimensional. Como imagen de función es un poco más complicada que la curva, pues el dominio no es un intervalo de la recta numérica R sino una región del plano R^2 que puede ser un poco rara, pero no demasiado; si tiene agujeros no son infinitos ni tangentes entre sí, etc. Con esas previsiones podemos decir que **una superficie es la imagen de una función continua desde una región conexa [1] del plano real** R^2 **hacia el espacio euclídeo tridimensional** E^3.

La definición analítica de una dada curva o superficie 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 explícito y directo de una coordenada en función de las otras:

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

  No todas las curvas o superficies pueden representarse correctamente de esta forma. Por ejemplo, para representar una recta vertical tendríamos que invertir el rol de las variables y expresar x como una constante en función de y. Representar un círculo no sería posible mediante una sola función, ya que para cada x hay dos valores de y. No se puede utilizar esta representación cuando las variables dependientes no son *funciones* (en el sentido estricto) de las independientes.

- **Implícita:**

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

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

[1] Para nosotros una región es "conexa" si dos puntos interiores o frontera de la región se pueden conectar mediante una curva de puntos interiores (no frontera).

También aquí, las curvas en 3D se definen mediante un par de ecuaciones en x, y, z, lo que equivale a definirlas como la intersección de dos superficies.

Ya hemos usado esta forma para definir planos $ax+by+cz+d=0$ y circunferencias $x^2+y^2=r^2$. Los puntos de la curva o superficie serán todos aquellos que satisfacen la ecuación. Para curvas y superficies estándar (círculos, planos, toroides, etc.), este tipo de definición es más directa y permite visualizar o modificar parámetros específicos de cada curva (radio de un círculo, normal y posición de un plano, etc.) o saber de qué lado está un punto si el resultado es mayor o menor que cero.

Una de las pocas ventajas es que permite definir familias de curvas o superficies; por ejemplo, variando r en el círculo o la esfera o d en el plano.

Sin saber el significado de los parámetros, puede ser difícil o imposible de graficar; ya que no se obtienen los puntos en forma directa. Por ejemplo: supongamos que no sabemos lo que representa la ecuación $x^2 + y^2 = 1$ ¿Cómo calculamos sus puntos? para un dado x, digamos $x = 3$, despejamos y en función de $x, obteniendo valores complejos, sin representación en el espacio real.

Esta forma no se utiliza para graficar, salvo en los casos especiales que ya utilizamos o aquellos en los que se puede subdividir y explicitar unas coordenadas en *función* de otras en un dominio conocido, en general sólo planos, curvas cónicas y superficies cuádricas.

- **Paramétrica:**

  Es la forma que más utilizaremos para definir analíticamente las curvas y superficies. Los valores de cada coordenada se expresan como ”función” de uno o dos parámetros reales independientes:

  - Curva: $x = f(t), \quad y = g(t), \quad z = h(t)$
  - Superficie: $x = f(u, v), \quad y = g(u, v), \quad z = h(u, v)$

  Para una curva, estas funciones dan un mapeo directo de cada t a un punto de la curva. Si dibujamos la curva con un lápiz en un papel, el parámetro t puede verse como el tiempo y las ecuaciones responden a la pregunta: ¿Dónde está el lápiz en el tiempo t? Para una superficie, también sabemos inmediatamente dónde ubicar el punto que corresponde al par (u, v) del dominio; para un dado u fijo, las funciones (de v) describen una **curva isoparamétrica** (o isocurva) con v variable; y viceversa.

- Ejemplos:

  - Recta por el origen: $x = at; \quad y = bt$ (puede ser vertical).
  - Circunferencia: $x = r\cos(t), \quad y = r\sin(t)$.
    - Una hélice agrega $z = t$ a las ecuaciones de la circunferencia.
  - Un ejemplo de superficie puede ser el Elipsoide:
    - $x = a\cos(u)\cos(v), \quad y = b\sin(u)\cos(v), \quad z = c\sin(v)$,

      que es una esfera de radio r si $a = b = c = r$.

  Esta forma es 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 y superficies (excepto fractales y rarezas).

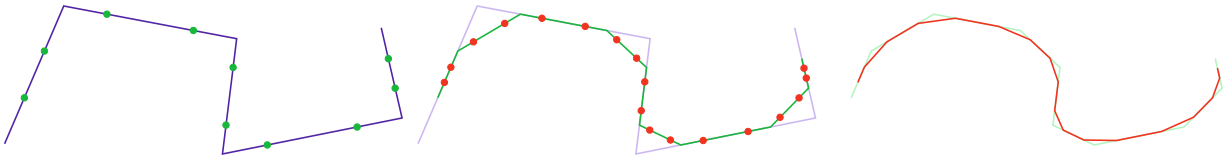
- **Generativa o Procedural:**

  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.

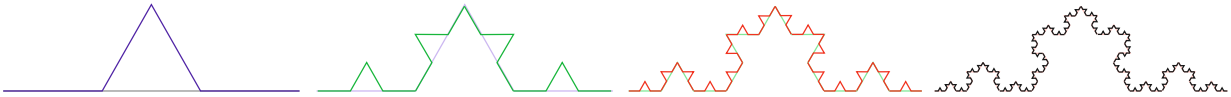
  Normalmente las curvas y superficies procedurales se definen como el límite para infinitas iteraciones del procedimiento que las genera. Veamos un par de ejemplos sencillos:

  Ejemplos:

  - *subdivision curve*: 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.

  - *Fractal de Koch*: Dividir en tres cada tramo y agregar un vértice que forme un triángulo equilátero con el par central

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

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

Las funciones pueden ser de cualquier tipo; pero en CG se prefieren las polinómicas por la simplicidad de su evaluación (algoritmo de Horner[2]) y sus bondades matemáticas (derivadas sencillas y continuas).

Una **curva en serie de potencias** se define mediante un polinomio vectorial de grado n:

$$\mathbf{P}(u) = \sum_{0}^{n} \mathbf{a}_i u^i \qquad \text{(aquí, el superíndice del parámetro indicará siempre potenciación)}$$

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

[2] Que no es de Horner. Citando una nota de Jeff Erickson, en su libro *"Algorithms"*: La ley de Stigler de los epónimos dice que "Ningún descubrimiento científico recibe el nombre de su descubridor original". El estadístico Stephen Stigler, del cual recibe su nombre esta ley, la presentó en un paper de 1980, afirmando que fue descubierta por el sociólogo Robert K. Merton. Pero resulta que en realidad hay formas similares anteriores: Vladimir Arnold en los 70 ("Los descubrimientos raramente se atribuyen a la persona correcta."), Carl Boyer en 1968 ("Clio, la musa de la historia, suele ser muy flexible al asignar nombres a los teoremas."), Alfred North Whitehead en 1917 ("Todo lo importante ha sido dicho antes por alguien que no lo descubrió."), y hasta el padre de Stephen, George Stigler en 1966 ("Sería notable encontrar un caso en el que una teoría reciba el nombre de la persona correcta").

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

Para el usuario que diseña la curva en un programa (CAD), no resulta nada sencillo definir el conjunto de vectores $\mathbf{a}_i$ que haga 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 (limitado).

1. Curvas de Bézier

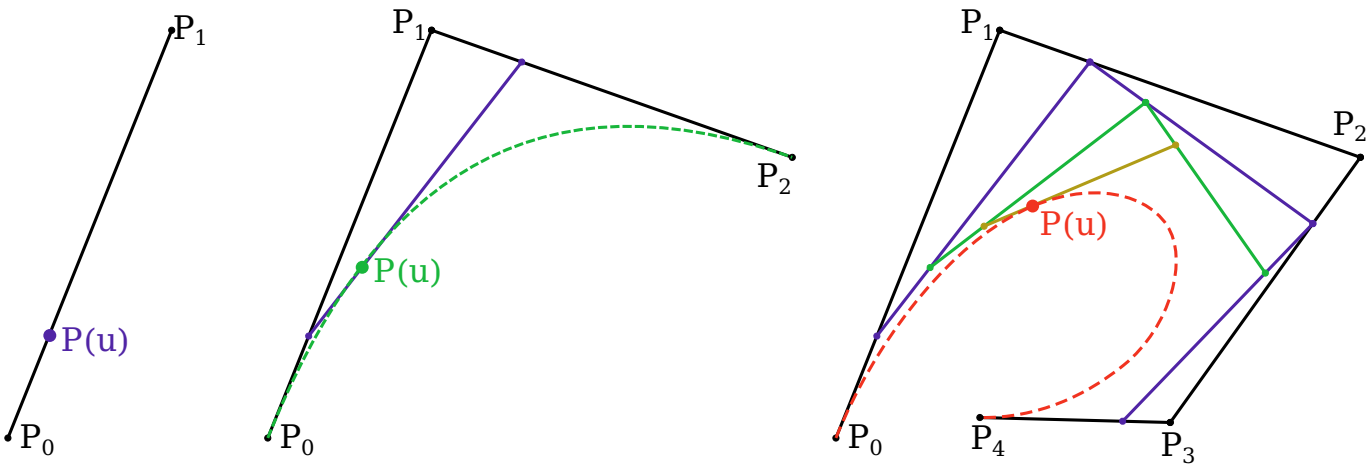
Se denomina curva de Bézier a **un método de definición de una curva en serie de potencias**. Consiste en definir algunos **puntos de control**, a partir de los cuales se calculan los puntos de la curva.

Esta página está repleta de curvas de Bézier, suele haber más de una en cada letra *true-type*. Haciendo mucho zoom en esta página o agrandando el tamaño de letra, nunca verá los caracteres pixelados, pues están definidos vectorialmente.

1.1. Algoritmo de De Casteljau

Describiremos el método de construcción recursiva 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: $P(u) = (1–u)P_0 + uP_1$. Donde los P_i son los puntos de control y $u \in [0, 1]$ el parámetro.



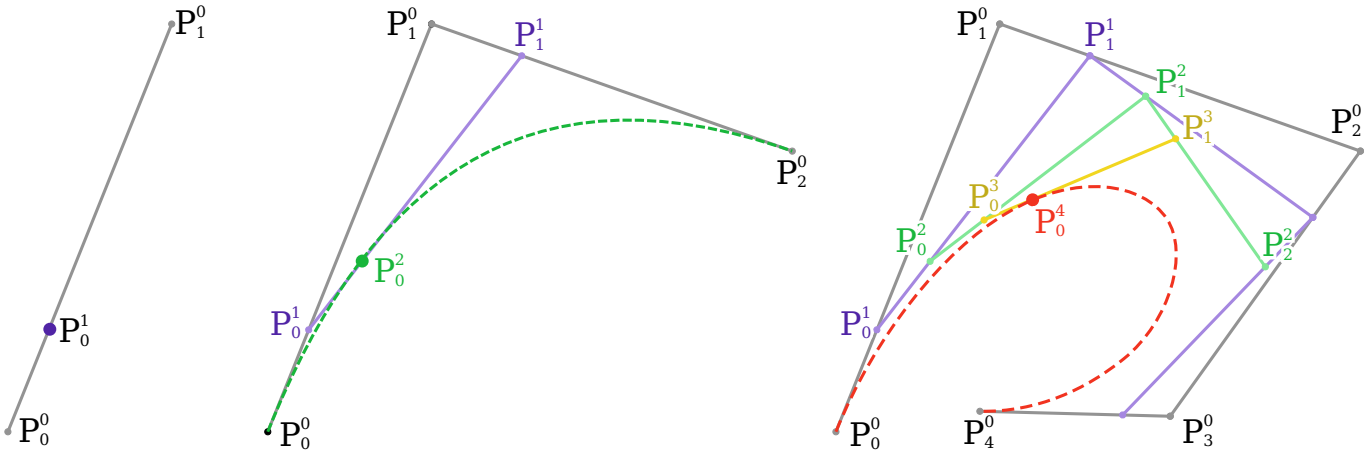
Agregando un punto de control más, la curva adquiere más interés: se interpola linealmente en cada uno de los segmentos consecutivos y luego entre los puntos resultantes, siempre en la misma dirección y con el mismo valor del parámetro. Para más puntos de control, el proceso se repite en forma iterativa.

La poligonal abierta que se forma con los puntos de control ordenados, es el **polígono de control**.

Cambiemos la notación para interpretar mejor las propiedades y el código fuente para programarlas. Usemos un superíndice en los puntos, para indicar el nivel de iteración. Llamando P_0^i a los puntos de control dados (nivel 0, de partida); en cada iteración se obtiene un punto nuevo, interpolado entre cada par sucesivo, que se denomina igual que el primero del par, pero aumentando uno el superíndice:

$$P_i^{j+1} = (1-u)P_i^j + uP_{i+1}^j$$

Con esta notación, si hay $n+1$ puntos de control (0 a n), en cada iteración hay un punto menos; al final sólo queda uno, el punto de la curva, que es $P_0^n(u)$. El dibujo anterior queda como sigue:



Por razones históricas se denomina **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 agregado, 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.

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

Hagamos el desarrollo analítico del punto variable en función de los puntos de control dados, para una curva de 3er grado; que es la que casi siempre se utiliza. El superíndice de u o $(1-u)$ indica potenciación.

$$\begin{aligned}
P_0^3 &= \underbrace{(1-u)}_{B_0^1} P_0^2 + \underbrace{u}_{B_1^1} P_1^2 \\
&= (1-u)\underbrace{[(1-u)P_0^1 + uP_1^1]}_{P_0^2} + u\underbrace{[(1-u)P_1^1 + uP_2^1]}_{P_1^2} = \underbrace{(1-u)^2}_{B_0^2} P_0^1 + \underbrace{2(1-u)u}_{B_1^2} P_1^1 + \underbrace{u^2}_{B_2^2} P_2^1 \\
&= (1-u)^2\underbrace{[(1-u)P_0^0 + uP_1^0]}_{P_0^1} + 2(1-u)u\underbrace{[(1-u)P_1^0 + uP_2^0]}_{P_1^1} + u^2\underbrace{[(1-u)P_2^0 + uP_3^0]}_{P_2^1} \\
&= \underbrace{(1-u)^3}_{B_0^3} P_0^0 + \underbrace{3(1-u)2u}_{B_1^3} P_1^0 + \underbrace{3(1-u)u^2}_{B_2^3} P_2^0 + \underbrace{u^3}_{B_3^3} P_3^0
\end{aligned}$$

El último resultado se puede escribir como suele encontrarse en la bibliografía:

$$P(u) = \sum_{0}^{n} B_i^n(u) P_i$$

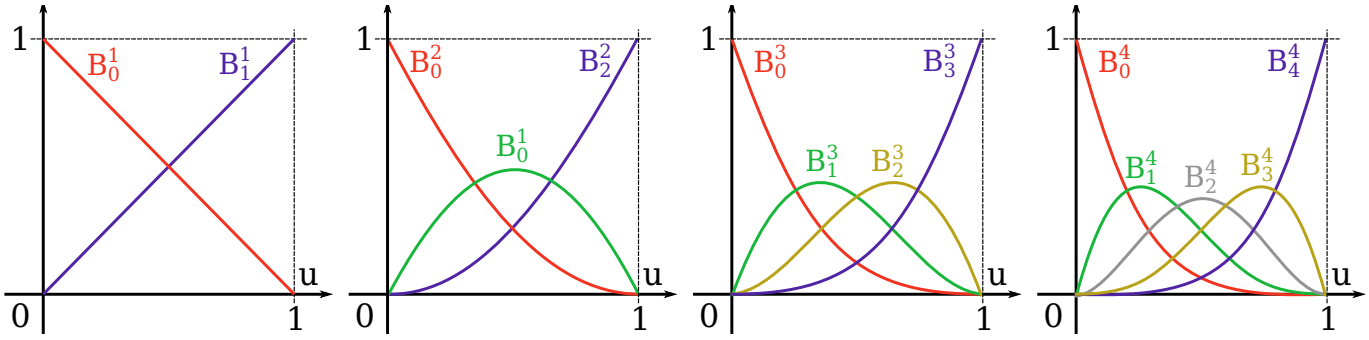
Los $B_i^n(u)$ son los **polinomios de Bernstein** y forman la base de un espacio funcional (n+1)-dimensional. Un punto de la curva de Bézier se identifica mediante sus $n+1$ coordenadas vectoriales, que son los puntos de control P_i. La base de polinomios de Bernstein es:

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

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:

$$C_i^n = \frac{n!}{i!(n-1)!} = \frac{n(n-1)...(n-i+1)}{12...i} \qquad \text{(i términos numerador y denominador)}$$

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



1.2. Propiedades

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. Esa simetría implica que **se puede revertir la lista de puntos de control y la curva resultará igual**.

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 (que varían entre cero y uno) suman uno.

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

Por lo tanto, **cualquier punto (toda la curva) estará en el *convex-hull* de los puntos de control.**

Cada punto de control se corresponde con un polinomio de Bernstein que actúa como su peso variable en función de u para la combinación. En este contexto las funciones de peso suelen denominarse ***blending functions*** (funciones de mezcla) y miden la influencia de cada punto de control en cada punto de la curva. El polinomio B_0^n toma el valor 1 en $u = 0$; mientras que $B_n^n = 1$ en $u = 1$; eso implica que **la curva interpola (pasa por) los puntos de control inicial y final**. Ningún otro B toma el valor unitario, por lo que P_0 y P_n son los únicos puntos de control interpolados siempre (aunque la curva puede pasar, por casualidad, sobre un punto de control intermedio).

Al mover un punto de control, cambia la curva entera; pues cada punto de la curva es afectado por todos los puntos de control. El cambio es mayor en las cercanías del punto movido, pero toda la curva cambia. Esto se llama **control global** y **es una propiedad indeseable** en CAGD, donde siempre se pretende el **control local** de la curva, es decir: que la curva varíe solamente en las cercanías del punto de control movido. Más adelante veremos cómo se arregla esto con las *splines*.

Las curvas de Bézier tienen **invariancia afín** porque son combinación afín de los puntos de control. Como ya hemos visto, la transformación afín mantiene las combinaciones afines, **para hacer una transformación afín de la curva solo hay que transformar sus puntos de control y construirla en el espacio transformado**, en lugar de tener que transformar una miríada de puntos de la curva. Esto no se cumple en una perspectiva, que no es afín.

1.3. Elevación de grado

A veces se cuenta con curvas de distinto grado y hay que representar todas según el mayor grado, elevando los grados menores. Se logra utilizando la identidad trivial: $P = (1{-}u)P + uP$; que eleva el grado en una unidad. Hagamos un ejemplo de 2do a 3er grado:

$$
\begin{aligned}
P &= \underbrace{(1-u)^2 P_0}_{B_0^2} + \underbrace{2(1-u)u}_{B_1^2} P_1 + \underbrace{u^2}_{B_2^2} P_2 = \\
&= \underbrace{(1-u)[(1-u)^2 P_0 + 2(1-u)uP_1 + u^2 P_2}_{(1-u)P} + \underbrace{u[(1-u)^2 P_0 + 2(1-u)uP_1 + u^2 P_2}_{uP} = \\
&= \underbrace{(1-u)^3 P_0 + 2(1-u)^2 uP_1 + (1-u)u^2 P_2}_{(1-u)P} + \underbrace{(1-u)^2 uP_0 + 2(1-u)u^2 P_1 + u^3 P_2}_{uP} = \\
&= \underbrace{(1-u)^3}_{B_0^3} P_0 + \underbrace{(1-u)u^2}_{\frac{1}{3}B_1^3}(2P_1 + P_0) + 2\underbrace{(1-u)u^2}_{\frac{1}{3}B_2^3}(P_2 + 2P_1) + \underbrace{u^3}_{B_3^3} P_3 = \\
&= \underbrace{(1-u)^3}_{B_0^3}\underbrace{P_0}_{\hat{P}_0} + \underbrace{3(1-u)u^2}_{B_1^3}\underbrace{\frac{1}{3}(2P_1 + P_0)}_{\hat{P}_1} + \underbrace{3(1-u)u^2}_{B_2^3}\underbrace{\frac{1}{3}(P_2 + 2P_1)}_{\hat{P}_2} + \underbrace{u^3}_{B_3^3}\underbrace{P_3}_{\hat{P}_3}
\end{aligned}
$$

Los extremos son (obviamente) los originales, los intermedios (uno más que antes) se interpolan entre pares sucesivos de originales. Dividiendo en tres partes cada segmento del polígono de control (hacerlo)

se puede observar la evidente regla gráfica. Por ejemplo, de 5^o a 6^o grado se divide cada segmento en 6 y serán 1, 2, 3, 4 y 5 sextos para el punto anterior y viceversa: (5 a 1) sextos para el posterior.

También se puede ver que, elevando indefinidamente el grado, el polígono de control se transforma en la curva. Pero la convergencia es lineal y hay un método más rápido para la aproximación y rasterización.

1.4. Derivadas

La derivada $\dot{P}(u) = dP(u)/du = \lim \Delta P/\Delta u$, es la velocidad de movimiento del punto al variar el parámetro. Todas las derivadas son vectoriales, pues la diferencia de puntos o de vectores, da vectores. Si el parámetro se identifica como tiempo, la primera derivada es la velocidad, tangente a la curva en el punto en que fue calculada; la segunda es la aceleración, con una componente tangente (cambio de magnitud) y una centrípeta (cambio de dirección), perpendicular a la curva y hacia la concavidad; la tercera derivada puede tener una componente perpendicular a las dos anteriores, que indica el cambio del plano de la curva. El interesado en la geometría diferencial puede leer sobre el triedro de Frenet-Serré.

Continuidad C^k significa que las primeras k derivadas existen y son continuas, si k es cero, la función es continua. Siendo polinómicas, **las curvas de Bézier tienen continuidad** C^∞; todas sus derivadas existen y son continuas. Eso no implica que sean visualmente suaves, sino que el trazado es suave: la derivada primera existe, pero puede ser nula en un punto interior; en ese punto, no existe *versor* tangente y la curva puede cambiar abruptamente de dirección.

Para calcular las derivadas basta con derivar los polinomios de Bernstein. Usando la definición factorial del número combinatorio, se llega a:

$$\frac{dB_i^n}{du} = n(B_{i-1}^{n-1} - B_i^{n-1}) \qquad \text{(Se utilizó: } C_i^n = \frac{n}{i}C_{i-1}^{n-1} = \frac{n}{(n-i)}C_i^{n-1}\text{)}$$

Los dos casos especiales se presentan con $i = 0$ e $i = n$; que darían índice -1 y mayor que el grado:

$$\frac{dB_0^n}{du} = -nB_0^{n-1} \qquad \frac{dB_n^n}{du} = nB_{n-1}^{n-1}$$

Pero puede considerarse: $B_{-1}^k = B_n^{n-1} = 0$.

Reemplazando, distribuyendo y alterando subíndices:

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

Que **puede verse como una curva de Bézier grado $n-1$, definida mediante puntos de control que son n veces las diferencias entre pares de puntos sucesivos**:

$$\frac{dP}{du} = \sum_{0}^{n-1} B_i^{n-1}(n(P_{i+1} - P_i)) = \sum_{0}^{n-1} B_i^{n-1}(n\Delta_i)$$

Resulta ilustrativo graficar las derivadas como una curva de Bézier, usando los segmentos Δ_i del polígono de control como vectores posición de los nuevos puntos de control. El vector posición de un punto de esta curva (que se denomina **hodógrafa**) es, para un dado u, la derivada dividida por el grado. Recursivamente se pueden obtener las derivadas sucesivas. El Apéndice I muestra el método. El *convex-hull* de las hodógrafas acota las derivadas (variaciones) de la curva, que oscilan cada vez menos.

Se conoce como ***variation diminishing*** o variación disminuida al hecho de que la curva no tiene más oscilaciones (*wiggles*) que su polígono de control. **En 2D, cualquier línea que corte al polígono de control en k puntos no puede cortar a la curva más de k veces (y k será siempre $\leq n$). Lo mismo sucede en 3D con un plano que corta la curva y su polígono de control.**

En la fórmula anterior tenemos las derivadas como funciones paramétricas de los puntos de control (de sus diferencias); pero podemos también derivar, en forma recursiva, en el algoritmo de De Casteljau; en función de los puntos interpolados en los distintos pasos:

Para una curva de 1er grado:

$$\frac{dP_0^1}{du} = \frac{d[(1–u)P_0^0 + uP_1^0]}{du} = P_1^0 - P_0^0 = \Delta_0^0$$

Para una de segundo grado:

$$\begin{aligned}\frac{dP_0^2}{du} &= \frac{d[(1–u)P_0^1 + uP_1^1]}{du} = \Delta_0^1 + (1–u)\frac{dP_0^1}{du} + u\frac{dP_1^1}{du} = \\ &= \Delta_0^1 + (1–u)(P_1^0 - P_0^0) + u(P_2^0 - P_1^0) = \\ &= \Delta_0^1 + [(1-u)P_1^0 + uP_2^0] - [(1-u)P_0^0 + uP_1^0] = \Delta_0^1 + P_1^1–P_0^1 = \\ &= 2\Delta_0^1\end{aligned}$$

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

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

De donde podemos ver que **el último segmento de De Casteljau es tangente a la curva y su longitud, multiplicada por el grado, es el módulo de la derivada**. En particular, en $u = 0$ o $u = 1$ puede verse que **la curva comienza y termina tangente al polígono de control**.

1.5. Reparametrización

Si el parámetro variable $u \in [0,1]$ se expresa en función de otro parámetro $t \in [t_0, t_1]$, la curva es exactamente la misma, pero se dice que está reparametrizada: $P(u(t)) = P(t)$. Cuando observamos una curva, a menos que tenga marcas o cambios de color, no podemos decir con qué velocidad ha sido dibujada. Por ejemplo: para un segmento recto, la velocidad $v = dP/du = \Delta P/\Delta u = \Delta P$ es constante; el punto a medio parámetro es el punto medio: $P(\frac{1}{2}) = \frac{1}{2}P_0 + \frac{1}{2}P_1$. Pero si $u = t^2$, el punto es $(1-t^2)P_0 + t^2 P_1$ y la velocidad crece: $dP/dt = 2t\Delta P$; el punto de parámetro medio: $P(\frac{1}{2}) = \frac{3}{4}P_0 + \frac{1}{4}P_1)$ está antes de la mitad, porque al principio va más lento.

La reparametrización permite diseñar la velocidad de recorrido de la curva; por ejemplo, para definir el movimiento de una cámara. En tales casos importan la posición y la velocidad en cada ”momento”.

En una **reparametrización afín**, las derivadas cambian de forma muy sencilla:

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

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

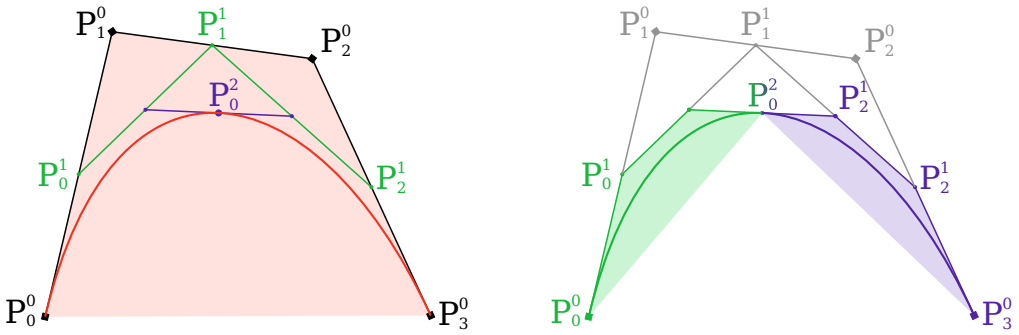
Una **reparametrización por longitud de arco** s haría que la velocidad del trazado, es decir: el vector tangente, tenga módulo constante. Para una curva $P(u)$ la longitud de arco$s$ se obtiene integrando:

$$ds^2 = dP \cdot dP = dx^2 + dy^2 + dz^2 \quad \Rightarrow \quad (\frac{ds}{du})^2 = (\frac{dP}{du})^2 = \dot{P}^2 \quad \Rightarrow \quad s(u) = \int_0^u \sqrt{\dot{P}^2} du$$

Obteniendo $s(u)$, podemos saber las longitudes de las porciones de curva y la función inversa $u(s)$ daría la curva en función de la longitud de arco; pero en general, esa integral no se puede calcular analíticamente; ni siquiera para polinomios. Esa reparametrización permitiría, por ejemplo, dividir la curva en tramos de igual longitud. Esto sólo puede realizarse mediante métodos numéricos aproximados.

1.6. Subdivisión

Cualquier tramo de curva de Bézier es, a su vez, una curva de Bézier del mismo grado; cualquier punto de la curva sirve para dividirla en dos. Dado un valor fijo u^* entre cero y uno, la secuencia de puntos $P_0^i(u^*)$ (todos con subíndice cero), calculados con el algoritmo de De Casteljau, define los nuevos puntos de control para el tramo de la curva entre 0 y u^*.



Para demostrar que el tramo coincide con la curva original, se realiza una reparametrización afín del tramo $[0, u^*]$ con $t = u/u^* \in [0, 1]$; desarrollando las ecuaciones para 1D y luego usando inducción, se demuestra que $P(t(u)) = P(u)$. Por simetría, queda también demostrado para el tramo final.

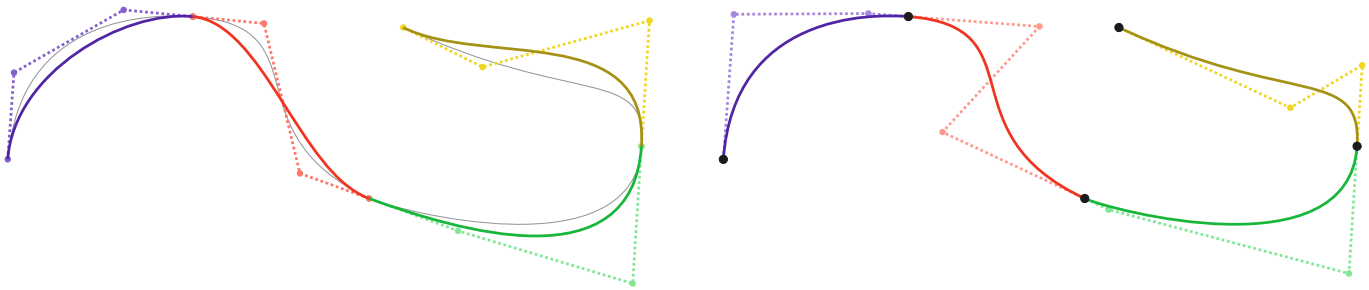
En la figura de la derecha se observa que el "ancho" del *convex-hull* de cada tramo se reduce; podemos definirlo como la distancia entre dos paralelas al segmento (P^0, P^n) y que encierren el *convex-hull*. Tampoco lo demostraremos aquí, pero **el ancho del *convex-hull* se reduce cuadráticamente con cada subdivisión**, proveyéndonos de un algoritmo de aproximación que converge más rápido que la elevación del grado. **La subdivisión recurrente es el método natural de aproximación**, sirve para encontrar rápidamente la intersección de la curva con una línea u otra curva, para rasterizar la curva o, también, para encontrar el punto de la curva más cercano a un punto dado (ej: cursor). Dado que la curva está dentro del c*onvex-hull*, el ancho acota el error y se utiliza para decidir que un tramo ya es suficientemente recto, detener el algoritmo y tratar al tramo como un segmento rectilíneo, ya sea para rasterizar o interceptar. Las subdivisiones se realizan mientras el ancho sea mayor que un determinado valor, en un caso el error admisible y en el otro un píxel.

2. *Splines* por unión de curvas de Bézier.

2.1. Continuidad Geométrica y Paramétrica

Si se quiere trazar una curva de Bézier compleja, se necesitan muchos puntos de control y, por lo tanto, un grado muy alto; que, sumado al control global, hace que resulte muy engorroso. Para aproximar una curva cualquiera, se puede hacer una unión "suave" de varias curvas de Bézier de grado bajo (dos o tres). Tal unión suave se denomina ***spline*** (o ***path***, en algunos programas).

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 límites, se conoce como ***knot vector*** o vector de nudos.



Cuando analizamos la continuidad o suavidad de las curvas definidas por tramos, estamos analizando en realidad la continuidad de las derivadas en los puntos de unión, ya que en el interior de cada tramo 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 cada par sucesivo tenga una 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 las derivadas sean proporcionales y en el mismo sentido (proporción positiva). Esto 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 debe variar suavemente el vector derivada intrínseca $\dot{P}/||\dot{P}||$, que no es más que el versor tangente a la curva original y sería la derivada si el parámetro fuese la longitud de arco. Por extensión, para las derivadas sucesivas se pide que el vector derivada, pero normalizado (cuando el módulo no es nulo) sea continuo; si el módulo se anula, porque se anula una derivada paramétrica en un punto, se pide que la discontinuidad sea evitable, es decir que coincidan los límites a derecha e izquierda.

En el caso particular de G^1, bastará entonces 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. No es exactamente lo mismo que pedir continuidad de la recta tangente, porque implica también un sentido de recorrido: imagine una curva que termina tangente a una recta y una curva simétrica también tangente a la recta; al unir los tramos hay continuidad de la recta tangente, pero los versores tangentes tienen sentidos opuestos. Es más, como ya se mostró, en una curva C^∞ puede haber un punto interior de quiebre si la velocidad (derivada paramétrica) se anula. En cambio, si una curva es G^1, también será G^2 si los dos tramos tienen la misma circunferencia osculatriz (o círculo osculador)[3], porque eso implica que tienen el mismo centro y radio de curvatura.

En general, la continuidad visual es G^1, pero cuando lo que importa o lo que se ve es la ”velocidad” con la que varía 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,

[3] Recordar que la tangente se define mediante dos puntos acercándose al límite. ”Ósculo” significa ”beso”; osculatriz es la circunferencia que besa a la curva y se define igual en dos o más dimensiones, como la circunferencia que pasa por tres puntos de la curva que se aproximan hasta el límite de distancia nula. Queda definida por la segunda derivada, del mismo modo que la tangente queda definida por la primera derivada.

se requiere continuidad G^2 para que no haya cambios bruscos de la curvatura y por lo tanto del giro del volante y de la fuerza centrífuga.

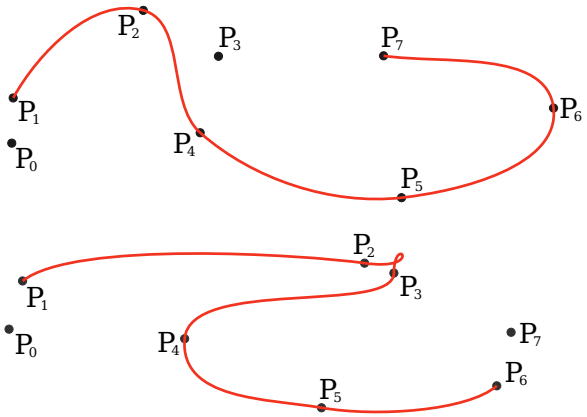
Volviendo a la unión de curvas de Bézier: Para lograr la continuidad G^1 sólo se requiere que el segmento inicial del polígono de control de cada tramo sea colineal, contiguo y opuesto al segmento final de la curva anterior, independientemente del grado de cada tramo. La continuidad C^1 (idénticas derivadas paramétricas) requiere que los segmentos, multiplicados por el grado, sean iguales y opuestos a cada lado de la unión. La relación entre la curvatura y la posición de los puntos de control es más complicada, esperemos hasta las *B-splines* para lograr continuidad C^2 en forma sencilla.

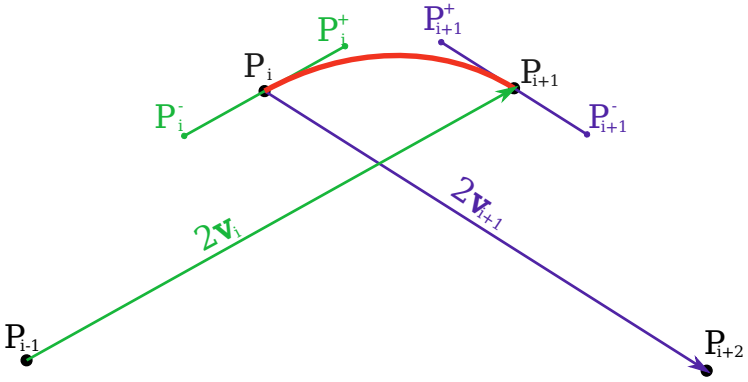
2.2. Interpolación de puntos con Splines

Para hacer pasar una curva por un conjunto dado de puntos ("interpolar") hay que hacer algunas elecciones arbitrarias, pues hay infinitas curvas posibles. Aquí analizaremos dos posibles soluciones: las *splines de Camtull-Rom*, y el *método de Overhausser*

2.2.1. Splines de Catmull-Rom

Dado un conjunto ordenado de puntos P_i, con $i \in [0, m]$, una *spline* de Catmull-Rom es una sucesión de curvas de Bézier de tercer grado, que interpola los $m-1$ puntos intermedios con continuidad C^1.





Entre cada par de puntos P_i y P_{i+1}, se define una curva de Bézier de tercer grado; para ello se agregan (inventan) dos puntos de control intermedios que garanticen la continuidad paramétrica. El parámetro u se hace variar 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 se define la velocidad local como la velocidad media de los tramos que se unen:

$$\mathbf{v}_i = \frac{dP(i)}{du} = \frac{\Delta P}{\Delta u} = \frac{P_{i+1} - P_{i-1}}{(i+1) - (i-2)} = \frac{P_{i+1} - P_{i-1}}{2}$$

Conociendo el vector derivada $\mathbf{v}_i$ en el punto i, podemos inventar los dos puntos extra: P_i^- para la curva de Bézier que llega al punto y P_i^+ para la que sale de ahí. Como se explicó al desarrollar las derivadas,

cada uno de esos puntos dista de Pi un tercio (grado tres) de la derivada recién calculada:

$$P_i^- = P_i - \mathbf{v}_i/3 \qquad P_i^+ = P_i + \mathbf{v}_i$$

Los puntos originales proveen **control local** a la curva, el movimiento de uno de ellos afecta a lo sumo a cuatro tramos de la curva; esto simplifica mucho el diseño por tramos, para una curva extensa.

Supóngase un tramo largo seguido de uno corto; habiendo asignado la misma velocidad media a ambos tramos, es posible que la curva "no tenga tiempo de frenar" (*overshoot*) y genere un rulo. Se ve un ejemplo entre P_4 y P_5 en una figura.

2.2.2. El método de Overhausser

Cuando hay tramos de distinta longitud (y, en general, los hay) 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 elegir las velocidades y también hay muchas formas de definir la dirección de la recta tangente en los puntos de unión, la elección del método depende del uso que se le quiera dar.

En el método de Overhausser, el parámetro en el punto i no es i sino la suma de las longitudes de los segmentos hasta ese punto, de modo que la velocidad media en cada tramo es de módulo unitario:

$$\mathbf{v}_i^- = \frac{P_i - P_{i-1}}{|P_i - P_{i-1}|} \qquad \mathbf{v}_i^+ = \frac{P_{i+1} - P_i}{|P_{i+1} - P_i|}$$

Para definir la dirección común a los dos tramos (G^1), se realiza un promedio ponderado de las velocidades medias anterior y posterior, donde pese más el punto más cercano. Pesar con la cercanía es pesar con la inversa de la distancia, algebraicamente resulta lo mismo que pesar cada velocidad con la distancia al otro punto, el cociente es la suma de longitudes, pues los pesos deben sumar uno:

$$\mathbf{v}_i = \frac{|P_{i+1} - P_i|\mathbf{v}_i^- + |P_i - P_{i-1}|\mathbf{v}_i^+}{|P_i - P_{i-1}| + |P_{i+1} - P_i|}$$

En este caso la variación del parámetro no es unitaria sino la longitud del segmento; por lo tanto, las distancias a los puntos agregados no son $\mathbf{v}_i/3$ sino que deben multiplicarse por la respectiva longitud que es el factor de reparametrización:

$$P_i^- = P_i - |P_i - P_{i-1}|\mathbf{v}_i/3 \qquad P_i^+ = P_i + |P_{i+1} - P_i|\mathbf{v}_i/3$$

Con esto se pierde la continuidad paramétrica, el dibujo de la curva es G^1, pero la velocidad paramétrica pega saltos al pasar por los puntos originales. La curva se ve suave pero no se recorre suavemente.

Con otros criterios se pueden definir distintos métodos de acuerdo a las necesidades y aplicaciones, pero siempre se basan en definir la dirección tangente y las derivadas a ambos lados.

2.2.3. Tratamiento de los extremos

En los casos recién mencionados, para *inventar* puntos de control alrededor de P_i se utilizan los puntos a interpolar P_{i-1} y P_{i+1}. Sin embargo, ¿qué hacer en los extremos inicial y final, ya que no dispone de ambos puntos?

Una posible respuesta es ”no interpolar los puntos inicial y final”. En este caso, esos puntos solo servirán para definir las derivadas al comienzo y al final de la curva, pero la verdadera curva irá desde el 2do hasta el ante-último.

Otra opción algo mejor en muchos casos es ”inventar” los tramos iniciales y finales de la curva, en base a un solo punto adicional. Por ej, para el 1er tramo, que va desde P_0 a P_1 necesitaríamos P_0^+ y P_1^-. Como no podemos obtener P_0^+ con los métodos propuestos anteriormente, podemos adaptar el tramo para definirse a partir de solamente P_0, P_1^- y P_1. Se puede lograr esto haciendo que ese tramo sea de grado 2 en lugar de 3, y adaptando el cálculo de P_1^- al nuevo grado; por ejemplo, en el método de Overhausser tendríamos:

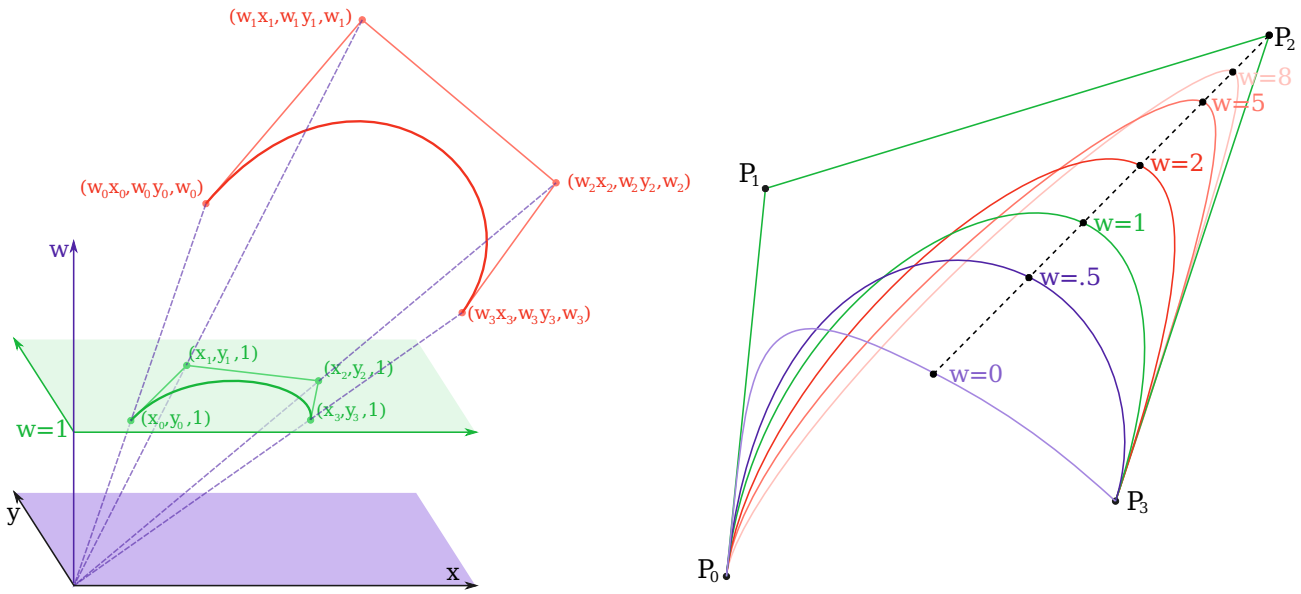
$$P_1^- = P_1 - |P_1 - P_0|\mathbf{v}_1/\boxed{2} \qquad P_{n-1}^+ = P_{n-i} + |P_n - P_{n-1}|\mathbf{v}_{n-1}/\boxed{2}$$

De esta forma se definen los tramos inicial y final como curvas de Bezier de 2do grado. Se puede además obtener las curvas de 3er grado equivalentes (aplicando el algoritmo de elevación de grado que se describió más arriba), para almacenar todos los tramos como curvas de grado 3, y así simplificar la implementación (utilizar una estructura de datos uniforme para todos los tramos, y evitar condicionales en las evaluaciones de la spline).

3. Curvas de Bézier racionales

Una curva de Bézier racional está definida igual que antes, pero mediante puntos de control con coordenadas homogéneas R^{d+1} $\{x(u), y(u), z(u), w(u)\}$. Allí, en R^{d+1}, es una simple curva polinómica con una dimensión más. En el espacio proyectivo P^d, donde la vemos, la traza se obtiene dividiendo por el polinomio $w(u)$ y de ahí la denominación de ”racional” por ser un cociente de polinomios.

Una curva cambia de forma cuando se realiza una transformación, si la curva está definida por medio de una combinación afín de puntos, mantiene la misma formulación ante una transformación afín, pero no en una proyección central u otra proyectiva. Las transformaciones proyectivas son lineales en una dimensión más; por lo tanto, la transformación proyectiva de una curva de Bézier se hace por medio de la transformación lineal de sus puntos de control en R^{d+1} y luego se realiza la división por $w(u)$ al final.



Las curvas racionales se pueden definir sin dificultad (y como antes) en coordenadas homogéneas:

$$\hat{P} = \sum B_i^n \hat{P}_i \quad \text{o:} \quad \{wP, w\} = \sum B_i^n \{w_i P_i, w_i\} \qquad (\hat{P} \text{ es 4D, } P \text{ es 3D})$$

El punto variable de la curva, proyectada en tres dimensiones, se obtiene dividiendo por el polinomio *w*:

$$O = \frac{\sum B_i^n w_i P_i}{\sum B_j^n w_j} = \sum \frac{w_i}{w} B_i^n P_i \qquad (w = \sum B_j^n w_j)$$

Se puede ver el sentido de ”peso” adicional que adquiere la coordenada w y se justifica la denominación w por *weight*. Los pesos de los puntos suman uno y por eso las curvas de Bézier racionales son combinación afín (y convexa, si ningún w_i es negativo) 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:

$$\hat{P}_i^j = (1-u)\hat{P}_i^{j-1} + uP_{i+1}^{j-1} \quad \Rightarrow \quad P_i^j = \frac{(1-u)w_i^{j-1}P_i^{j-1} + uw_{i+1}^{j-1}P_{i+1}^{j-1}}{(1-u)w_i^{j-1} + uw_{i+1}^{j-1}}$$

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

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

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

Resulta muy beneficioso poder alterar la forma de una curva de grado bajo, cambiando el peso de un punto. El problema es no imponer al usuario una notación y conceptos complicados, los puntos de control

tienen cuatro dimensiones que el usuario no necesita entender. Una interfaz gráfica adecuada debe pedir al usuario que defina los puntos de control que ve: $\{x_i, y_i, z_i\}$ y esconder el carácter homogéneo mediante un número adicional w_i que funciona como peso, que "atrae" la curva hacia el punto. En OpenGL, por ejemplo, si el usuario quiere un punto "en $\{x, y, z\}$, con peso w", el punto debe definirse mediante `glVertex4d(wx,wy,wz,w)` en el espacio homogéneo; en cambio, si quiere definir un vector o un punto en el infinito $\{\Delta x, \Delta y, \Delta z\}$ (o $\{x, y, z\}$, pero en la dirección que va desde $\{0, 0, 0\}$ hacia $\{x, y, z\}$, debe ser definido mediante `glVertex4d(x,y,z,0)`.

Las derivadas de las curvas racionales se dificultan por la regla del cociente:

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

La expresión en sí no interesa, pero nos muestra que dependen en forma sencilla del último segmento del algoritmo de De Casteljau. En todo el proceso recursivo de De Casteljau, la recta tangente a la curva 3D es proyección de la tangente 4D. 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.

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

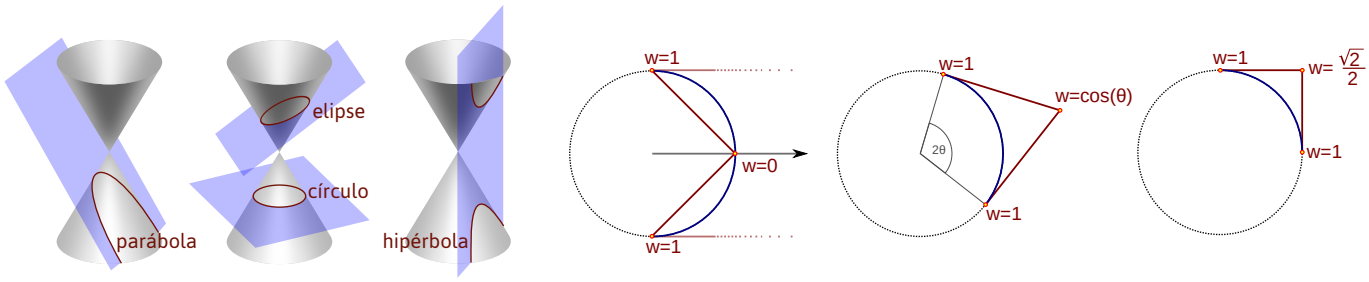
La excepción es cuando el denominador resulta nulo, donde habrá que hacer un análisis de límites.

Además de la sencilla forma de hacer una transformación proyectiva y la posibilidad de modificar la curva mediante pesos, la mayor justificación para usar estas curvas es la posibilidad de dibujar "cónicas"; la circunferencia es la más importante, aunque para distintas aplicaciones técnicas se requieren, a veces, arcos de hipérbola, parábola o elipse.

Cortando un cono mediante un plano se obtienen todas las curvas de segundo grado (si el plano contiene al vértice resultarán "degeneradas"). Un plano perpendicular al eje define una circunferencia; al inclinarse respecto del eje se obtienen las elipses, de excentricidad creciente, hasta que el plano, paralelo a la generatriz, define una parábola; para una inclinación mayor se obtienen las hipérbolas. Una transformación proyectiva permite transformar a cualquiera en cualquier otra (sombra en la pared del aro del velador).

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

Dados los puntos y las tangentes inicial y final de una curva de Bézier de segundo grado, el punto de control central estará necesariamente 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 pues escalando en x o en *y* y girando obtenemos las elipses, siendo la circunferencia un caso particular.

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. En todos los casos se puede demostrar algebraicamente que $x^2(u) + y^2(u) = r^2$; es decir que, efectivamente, son arcos de circunferencia.

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, asumiendo radio uno, las dos primeras tienen los puntos de control definidos como $\{wx, wy, w\}$; pero en la tercera, aun cuando el centro no sea el origen de coordenadas, el punto de control en el infinito sólo puede representarse como $\{1, 0, 0\}$; ($\{a, 0, 0\}$ daría una elipse de semiejes 1 y a, con paralelas que se unen en el mismo punto ideal). En general no hay que usar pesos nulos o negativos.

Obviamente no se puede definir una circunferencia completa con una sola curva de Bézier de segundo grado (hay una construcción con una de quinto), 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. Aun 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).

4. B-splines

Una *spline* es una curva continua, obtenida mediante unión suave de tramos curvos simples. 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 se utilizan las NURBS (*Non–Uniform Rational B–Spline*).

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 Bézier.

Normalmente, las *B-splines* y las NURBS se definen mediante las *blending functions* (que son como los polinomios de Bernstein para las curvas de Bézier) calculadas mediante el algoritmo recursivo de Cox-De Boor. 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.

4.1. *Blossoming* y Forma Polar

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

Definamos una función f multi-afín y totalmente simétrica, de n variables reales. Para no perdernos, fijemos en tres la cantidad de variables:

- Por "totalmente simétrica" entendemos que no cambia si se permutan libremente las variables:

$$f(a,b,c) = f(a,c,b) = f(c,a,b) = ...$$

- Por multiafín entendemos que la función se puede interpolar linealmente para cualquier variable:

$$f(a,(1-u)b+uc,d) = (1-u)f(a,b,d) + uf(a,c,d)$$

Suponiendo conocidos $f(a,b,d)$ y $f(a,c,d)$, podemos obtener la función para a, d y un valor entre b y c, mediante la interpolación lineal de los resultados conocidos.

Se puede pensar a f como una rutina que devuelve un real si le damos tres reales $\{a,b,c\}$, en cualquier orden: $f(3,1,5)$ devuelve el mismo resultado que $f(5,1,3)$. La rutina sólo puede calcular el resultado interpolando entre dos datos o resultados previos; por ejemplo: si conozco $f(3,1,3)$ y $f(3,1,9)$ y quiero calcular $f(3,1,5)$, debo obtener el valor del parámetro para 5 entre 3 y 9 e interpolar los resultados:

$$f(3,1,5) = 4/6f(3,1,3) + 2/6f(3,1,9)$$ (9-3=6, de 3 a 5 hay 2, de 5 a 9 hay 4)

La fórmula general, para interpolar en t a partir de t_0 y t_1 es una simple interpolación lineal:

$$f(a,b,c,\dots,t) = \frac{(t_1-t)f(a,b,c,\dots,t_0) + (t-t_0)f(a,b,c,\dots,t_1)}{t_1-t_0}$$

En nuestro caso, f no es un real sino un vector $\{x,y,z,w,R,G,B,\dots\}$, todas sus componentes reciben

el mismo tratamiento en simultaneo. Por simplicidad trataremos con puntos $P\ \{x, y, z[, w]\}$, pero no perdamos de vista que pueden tener cualquier cantidad de componentes. El punto resultado se obtiene mediante una sucesión de combinaciones afines de los puntos de control dados, es decir que es una combinación afín de los datos, cuyos pesos son polinomios en función de un solo parámetro real y por lo tanto define una curva unidimensional. Es una ampliación del algoritmo de De Casteljau.

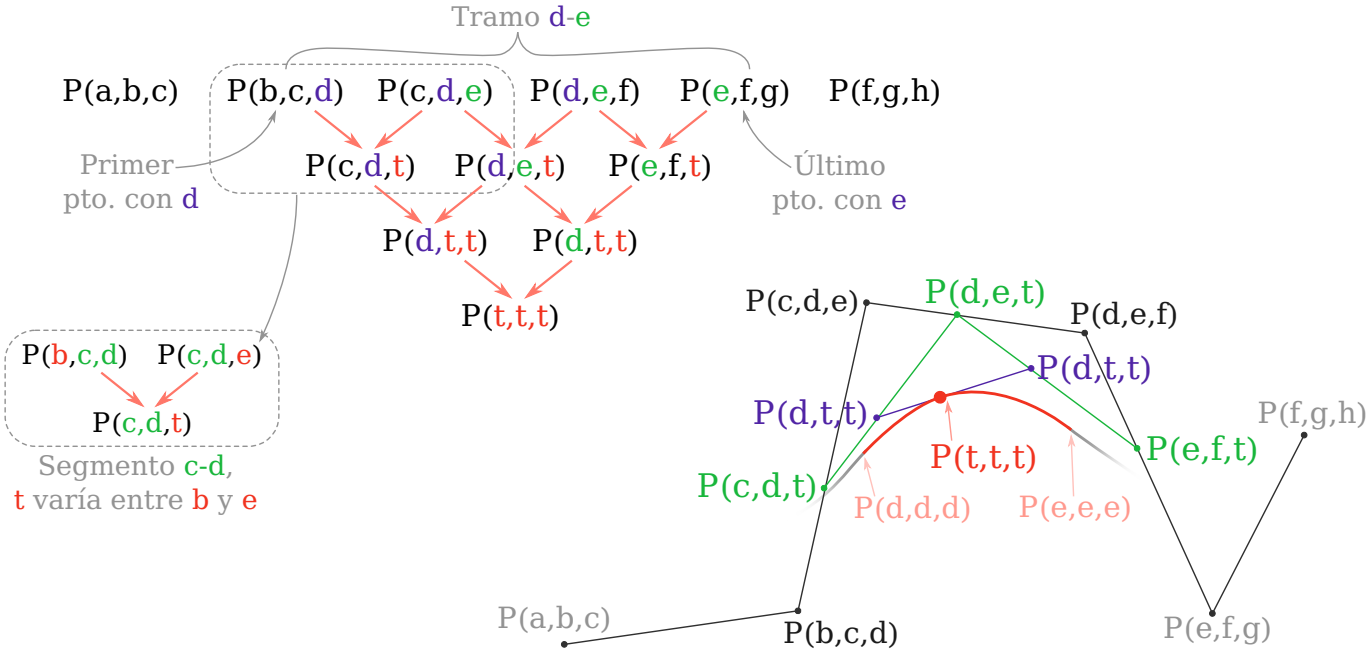
- $P(t, t, t)$, es el punto de la curva, cuando el parámetro toma el valor t.
- $\{a, b, c, d, \dots\}$ es un conjunto no-decreciente de valores reales del parámetro ($a \leq b \leq c \leq d \leq \dots$) que viene dado y que denominamos *knot-vector* o vector de nudos (notar que pueden repetirse).

- Partimos de los puntos de control conocidos que llamaremos $P(a, b, c),\ P(b, c, d),\ P(c, d, e),\ \dots$

Los puntos de control, por ejemplo: $P(0.75, 1.2, 7.5)$ vienen dados como puntos en el espacio, pero se les asignan tres *knots* sucesivos (o n, para una curva de grado n) de un vector de nudos que también viene dado, después veremos cómo y por qué.

El *Blossoming* (florecimiento) es un método de cálculo del punto de la curva, haciendo interpolaciones lineales recursivas de los puntos de control dados. El algoritmo de De Casteljau es un ejemplo.

El siguiente esquema del *blossoming* muestra la marcha de cálculos para obtener un punto en una curva de tercer grado (tres interpolaciones). En este ejemplo tenemos un vector de ocho nudos $\{a, b, c, d, e, f, g, h\}$ (valores reales del parámetro t, cada uno mayor o igual que el anterior) definiendo, para tercer grado, seis puntos de control $\{P(a, b, c),\ P(b, c, d),\ P(c, d, e),\ P(d, e, f),\ P(e, f, g),\ P(f, g, h)\}$. Queremos calcular el punto de la curva $P(t, t, t)$ para un dado valor de t, que se encuentra entre d y e del *knot-vector*.



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 $P(c, d, t)$ se obtiene interpolando los dos puntos que contienen el par c, d: $P(b, c, d)$ y $P(c, d, e)$; siendo $b \leq t \leq e$:

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

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 está 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 (curva no–interpolante).

La curva comienza en $P(c,c,c)$ y termina en $P(f,f,f)$. En el primer tramo, t varía entre c y d y 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. El intervalo total está definido por c que es el último valor del primer punto y f, que es el primer valor del último punto.
2. La curva sí depende de todos los puntos, los puntos extremos 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 $\equiv$ tercer grado).
4. Para calcular un punto en una curva de tercer grado: $P(t,t,t)$, cuando hay más 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 ($n-1$) 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, dividiendo $(e-c)$ en tramitos iguales y marcando el t interpolado).
6. Un punto de la curva correspondiente a un *knot* (ej.: $P(d,d,d)$) requiere una interpolación menos:

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

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

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

7. No habrá divisiones por cero, pues solo se interpola entre parámetros distintos.

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

La curva de Bézier es un caso particular; se caracteriza por el modo de repetición de los *knots.* En una Bézier de tercer grado el vector es $\{0,0,0,1,1,1\}$ (para grado n, hay un *knot* repetido n veces al principio y otro mayor, también n veces al final; pero pueden ser otros números, usamos 0 y 1 para no dividir por la diferencia, que es siempre 1). 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, ya son puntos de la curva. Calculemos un punto cualquiera de parámetro $u \in [0,1]$:

$$P(0,0,u) = (1–u)P(0,0,0) + uP(0,0,1)$$
$$P(0,u,1) = (1–u)P(0,0,1) + uP(0,1,1)$$
$$P(u,1,1) = (1–u)P(0,1,1) + uP(1,1,1)$$

$$P(0,u,u) = (1–u)P(0,0,u) + uP(0,u,1)$$
$$P(u,u,1) = (1–u)P(0,u,1) + uP(u,1,1)$$
$$P(u,u,u) = (1–u)P(0,u,u) + uP(u,u,1)$$

Como vemos, es exactamente el algoritmo de De Casteljau, un ejemplo de *blossoming*.

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.

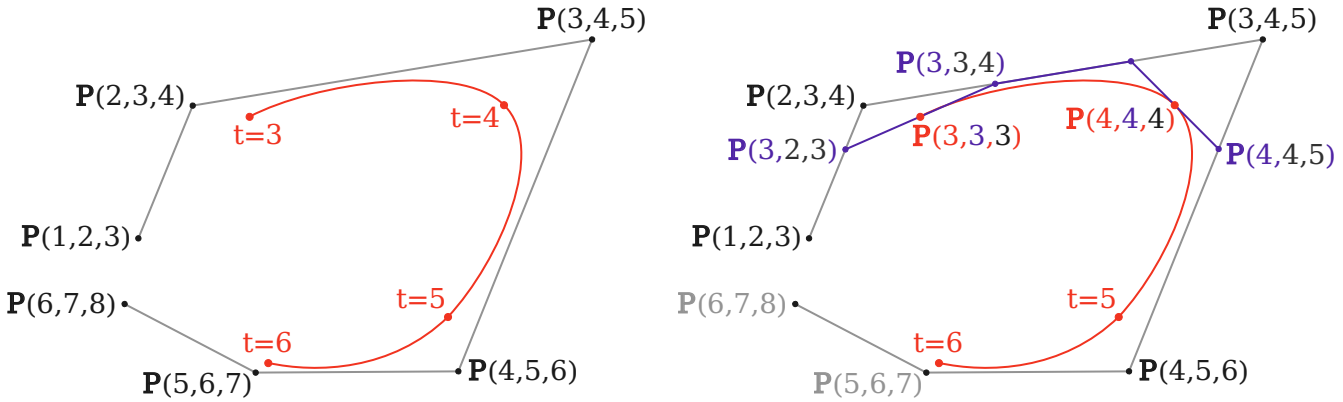
En las figuras que siguen se muestra una B-*spline* de tercer grado definida por el *knot-vector* $1,2,3,4,5,6,7,8$ y los seis puntos de control $\{P(1,2,3),\ P(2,3,4),\ P(3,4,5),\ P(4,5,6),\ P(5,6,7),\ P(6,7,8)\}$. El punto variable de la curva es $P(t,t,t)$ con $t \in [3,6]$ (el último del primer PC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

La curva consta de tres tramos 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.

Notar que en las figuras y fórmulas la notación varía, pues la simetría lo permite: $P(2,3,4) \equiv P(4,2,3)$.



Ejemplo de cálculo: Obtenemos el punto $P(3,2,3)$ interpolando entre $P(1,2,3)$ y $P(2,3,4)$. Dado que los números 2 y 3 están repetidos, el parámetro variable es $t=3$, entre $t_0=1$ y $t_1=4$:

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

Puede deducirse de la forma de interpolación, que un tramo de la curva $[t_0, t_1]$ está influenciado solamente por los puntos de control que tienen a t_0 o a t_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 $P(5,6,7)$.

5. NURBS

Las NURBS no son más que B-*splines* con pesos o proyectadas a partir de las B-*splines* en el espacio homogéneo, de una dimensión más. Como antes, se pueden hacer los cálculos del *blossoming* con pesos extra para cada punto de control, o se puede hacer todo en una dimensión más y luego dividir por w.

5.1. Knot-Vector

Para dibujar la curva se definen los puntos de control y el vector de nudos, que ya vimos que es una secuencia no decreciente con valores del parámetro. Si la secuencia está equiespaciada ($k_{i+1} - k_i = \text{cte.}$) la curva se denomina *spline* uniforme, en caso contrario será no-uniforme. La secuencia puede tener valores repetidos que se denominan nudos múltiples o *multiple knots*.

Los puntos de control libres al principio y al final imponen 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 se repiten los *knots* al principio y/o al final, tantas veces como indica el grado. Por ejemplo, en la figura de arriba, si al inicio de la curva 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: $\{3,3,3,4,5,6,7,8\}$; haciéndolo también al final nos quedará: $\{3,3,3,4,5,6,6,6\}$ y los puntos de control $\{P(3,3,3),\ P(3,3,4),\ P(3,4,5),\ P(4,5,6),\ P(5,6,6),\ P(6,6,6)\}$, quedando una *spline* no-uniforme que interpola los puntos de control extremos y llega tangente a los respectivos segmentos.

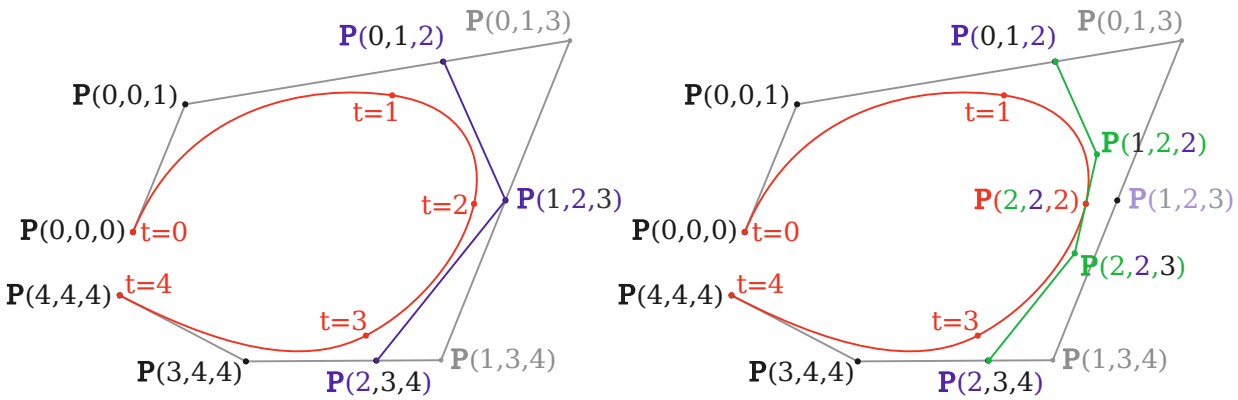
Tanto la forma de la curva como la velocidad del recorrido dependen del vector de *knots*. Hay que tener eso muy en cuenta, sobre todo cuando se definen trayectorias mediante NURBS, donde importan tanto el camino como la velocidad o la posición en cada ”momento”.

El vector de nudos se impone al definir la curva. En los programas de dibujo o CAD, donde solo importa la forma de la curva, suele ser totalmente transparente al usuario quien solamente define los puntos de control y pesos. El programa suele poner condiciones de Bézier en los extremos y puede definir una secuencia uniforme de *knots* en el interior, o bien puede espaciarlos en proporción a las distancias entre los puntos de control para mantener una velocidad más pareja (no es posible hacerla constante). Cuando el usuario quiere alterar la curva en cierta región, pica sobre la curva y el programa agrega un *knot* sin cambiar el resto ($\Rightarrow$ no-uniforme) y un punto de control más para que el usuario lo acomode.

El usuario suele ”tocar” los valores de los *knots* 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 se mira (*eye* y *target*). El vector de *knots* responde la pregunta ¿en qué momento el punto móvil está en las cercanías de tal punto de control? La función de peso que será máxima en las cercanías de un punto de control corresponde, en forma aproximada, al *knot* intermedio de ese punto, normalmente hay que experimentar hasta lograr la trayectoria y velocidad buscadas.

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 curva.

La **inserción de *knots***, como ya se mencionó, es el mecanismo que permite ”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 *spline* y se inserta un nuevo nudo intermedio de valor 2. El nuevo polígono de control, más refinado, ahora tiene el punto $P(1,2,3)$ que puede moverse libremente para reajustar la curva en las proximidades del nuevo punto.



El **algoritmo de De Boor**, que se utiliza para calcular un punto de la curva, consiste en encontrar el punto $P(t,t,t)$ y equivale a insertar tres veces (grado 3) 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. Pero cuidado: si ese punto se mueve, se mueve con continuidad C^0, esto se ve si se considera un movimiento del punto sin mover los vecinos $P(1,2,2)$ y $P(2,2,3)$, la curva pasa siempre por $P(2,2,2)$, llegando y saliendo tangente a los segmentos adyacentes, que antes eran antes colineales y ahora pueden hacer un quiebre.

El mismo truco de insertar n veces un *knot* 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. Esta subdivisión tiene las mismas propiedades y utilidades que la subdivisión de curvas de Bézier, como se puede observar en la figura, el nuevo polígono de control se aproxima mucho más a la curva; por lo tanto, es el método que se utiliza para rasterizar y para calcular intersecciones.

5.2. Derivadas

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

$$P(t,t,t) = n\frac{P(t,t,t_1) - P(t,t,t_0)}{t_1 - t_0} \qquad \text{(Recordarlo como } n\Delta P/\Delta t\text{)}$$

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. Es la técnica que se utiliza para definir puntos de quiebre de la continuidad de alguna derivada en el interior de una *spline,* cuando se quiere utilizar una sola *spline* con cambios de curvatura o incluso con puntos angulosos.

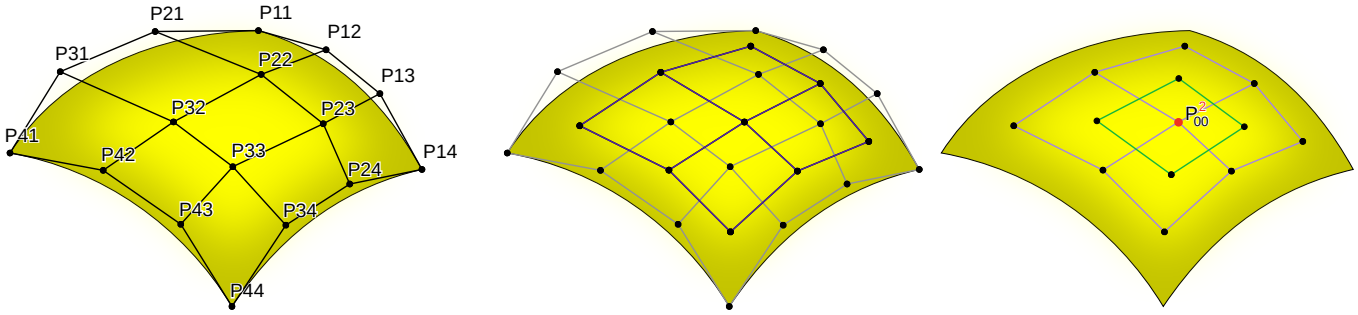
Para las NU**R**BS (racionales) las derivadas se complican por la regla del cociente; pero el método de construcción es igual al de Bézier y el resultado es similar y conceptualmente idéntico: en la medida en que no haya pesos nulos ni negativos las tangentes proyectadas son tangentes.

6. Superfices

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

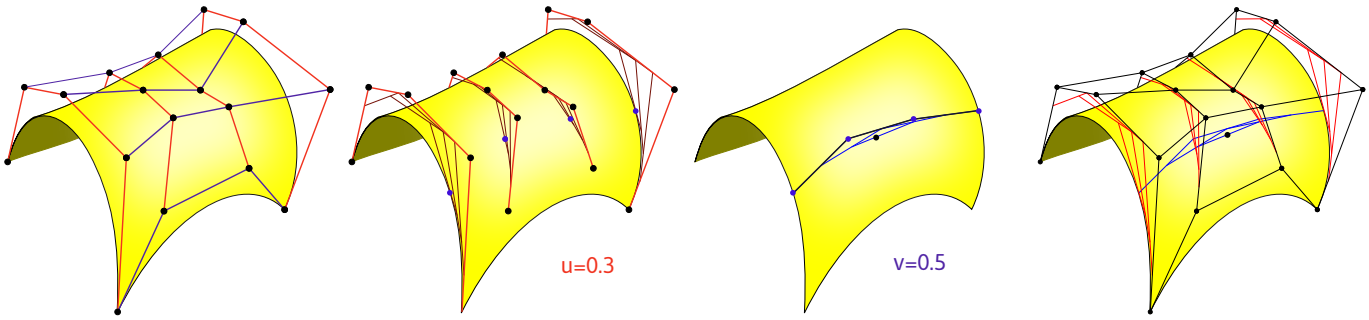
$$P(u,v) = \sum_i \sum_j B_i^n(u) B_j^m(v) P_{i,j} = \sum_i B_i^n(u) [\sum_j B_j^m(v) P_{i,j}] = \sum_j B_j^m [\sum_i B_i^n(u) P_{i,j}]$$

Los $(n+1) \times (m+1)$ puntos de control pueden ubicarse libremente en el espacio; pero están ”interconectados”, en secuencia 2D, como una **grilla regular** de cuadriláteros.



Para encontrar un punto de la curva $P(u, v)$ se puede utilizar una variante bidimensional del algoritmo de De Casteljau: en cada "cuadradito" se interpola bilinealmente con (u, v), formando nuevos "cuadraditos" donde se repite el proceso en forma recursiva hasta llegar al punto. La ultima superficie bilineal es tangente a la superficie, por lo tanto, la normal se puede calcular con el producto cruz de los segmentos $u =$ cte. y $v =$ cte. de la última interpolación bilineal.

Los dos últimos miembros, en la ecuación de arriba, muestran que la superficie también puede considerarse como una interpolación afín entre **curvas de control**. En el penúltimo miembro, por ejemplo, se puede definir $P_i(v) = \sum_j B_j^m(v) P_{i,j}$, que es la i–ésima curva de control. Para un dado valor fijo de u se obtiene una **curva isoparamétrica** $P(v)$ con v variable. Lo mismo sucede fijando j para obtener curvas de control o v para las isoparamétricas.

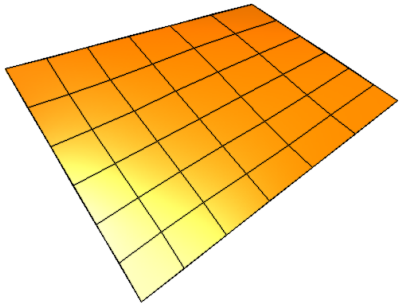


Las derivadas parciales se obtienen de las curvas isoparamétricas y su producto vectorial es la normal a la superficie en el punto.

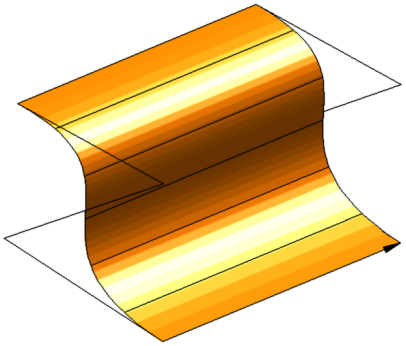
Existen variantes sencillas que son casos especiales y reciben nombres especiales:

- Cuando las dos direcciones se rigen por interpolaciones lineales tenemos las **superficies bilineales**, definidas por cuatro puntos, que son los vértices. Del mismo modo se denominan bicuadrática, bicúbica etc. cuando los polinomios (o cocientes racionales) son todos del mismo grado.
- Si los polinomios son lineales en una dirección, tendremos una **superficie reglada**. (Reglada ~ Se puede recorrer toda la superficie con una recta móvil). En el caso particular en que los segmentos son todos paralelos, la superficie es cilíndrica o extruida; si las rectas se cortan todas en un punto será un cono o un cono truncado. (En el espacio proyectivo, cilindro = cono con vértice en el infinito).
- Si una curva (generatriz o perfil) recorre la otra (directriz o trayectoria) se obtienen diversas **superficies de barrido (*swept*)** dependiendo de si se mantiene el ángulo entre las curvas (en general, 90^o; ej.: manguera) o si la generatriz se traslada paralela a sí misma (**traslacionales**; ej.: cilindros).

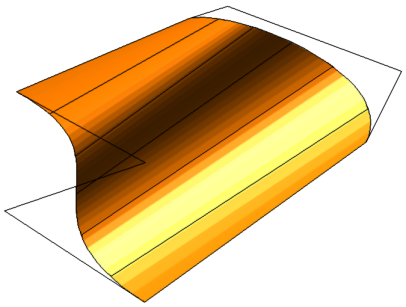
- Si un conjunto de curvas isoparamétricas (u o v) son círculos de eje común, se obtiene una **superficie de revolución**. Los puntos de control de la generatriz se copian girados sobre una circunferencia.



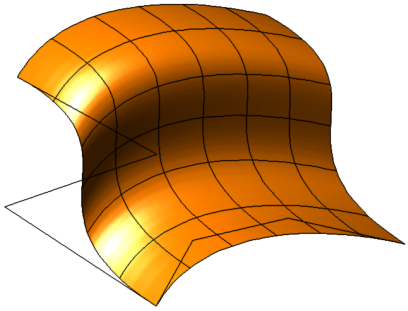
bilineal



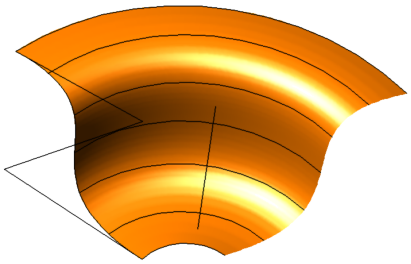
cilindrica



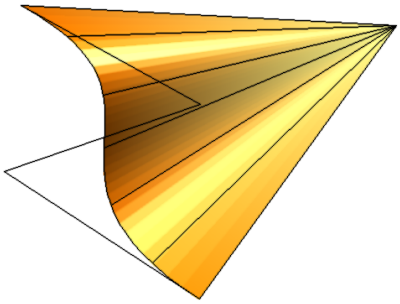
reglada



traslacional



revolución



cono

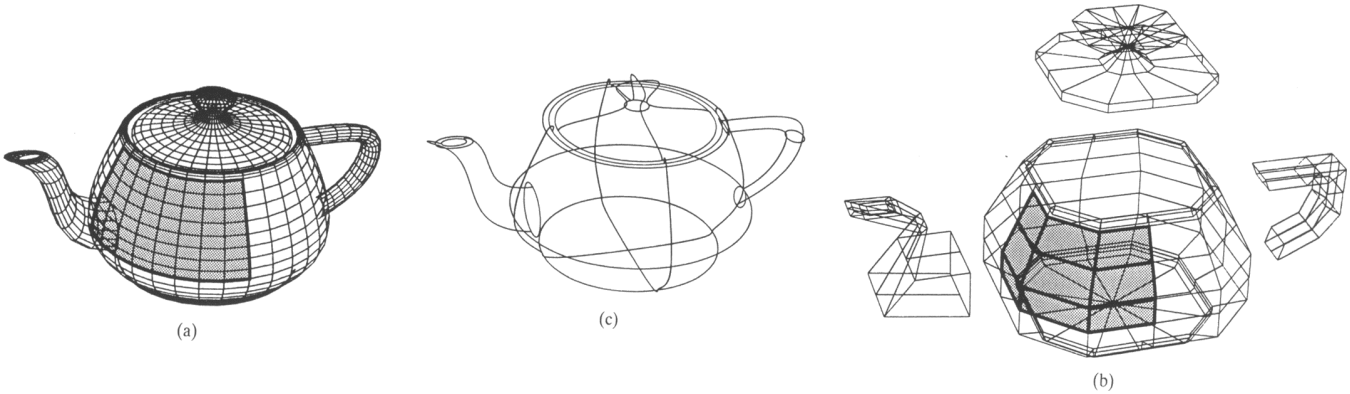
Hay un caso especial, históricamente anterior al producto tensorial y formalmente distinto, pero muy utilizado (ej.: AutoCAD), que es el de **Coons**. Se dan cuatro curvas unidas en circuito cerrado y se genera una superficie interior sin definir puntos de control internos. Para ello se realiza una combinación afín de tres superficies, se suman las regladas de cada par opuesto y se resta la bilineal de los cuatro vértices ($1 + 1 - 1$ da suma uno de los pesos). Analíticamente, llamemos $X_0(u)$ y $X_1(u)$ al par de curvas opuestas con u variando hacia un mismo lado y lo mismo para las otras dos curvas $Y_0(v)$ e $Y_1(v)$:

$$P(u,v) = [(1-u)Y_0(v) + uY_1(v)] + [(1-v)X_0(u) + vX_1(u)] - \\ -\{(1-u)[(1-v)X_0(0) + vX_1(0)] + u[(1-v)X_0(1) + vX_1(1)]\}$$

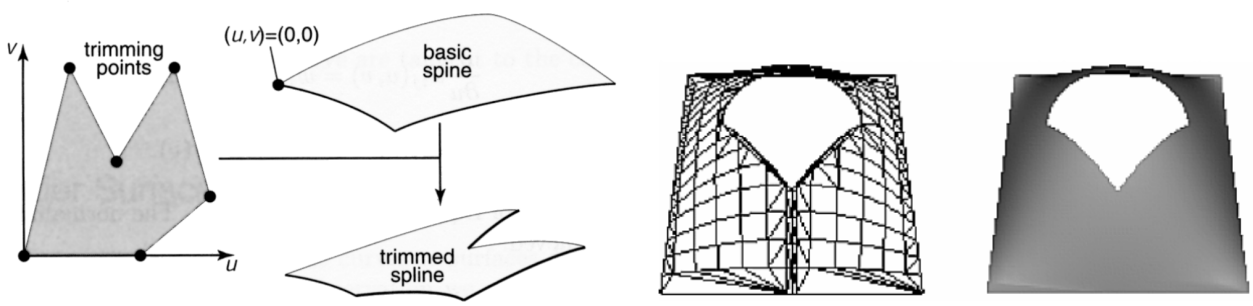
6.1. Unión y recorte de *patches*

Las superficies complejas se construyen como unión de *patches* o parches simples, del mismo modo que las *splines*. La continuidad del plano tangente y su normal (G^1) evita efectos lumínicos indeseables. Para definir el plano tangente en la unión, se necesitan dos curvas cruzadas; una de ellas es el borde común, de modo que basta asegurar que una curva atraviese al borde con continuidad G^1; y para que ello suceda en toda la unión, debe verificarse en sus puntos de control.

Un ejemplo es la tetera de Utah, construida por medio de parches de superficies de Bézier.



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 más curvas planas que recortan el exterior (CCW) y huecos (CW) 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 de OpenGL, que 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.

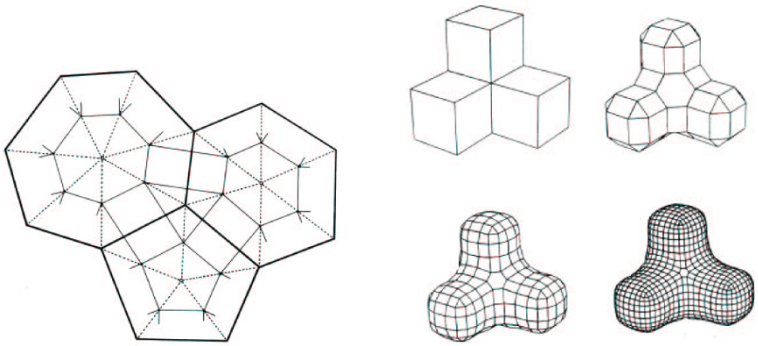
Las *trimmed* NURBS proveen una gran ventaja para realizar las uniones, pero no solucionan todo el problema, pues el borde recortado, en el espacio, tiene una formulación muy complicada como para ser usada por otra superficie NURBS de grado no muy alto, por lo cual la mayoría de los programas suelen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

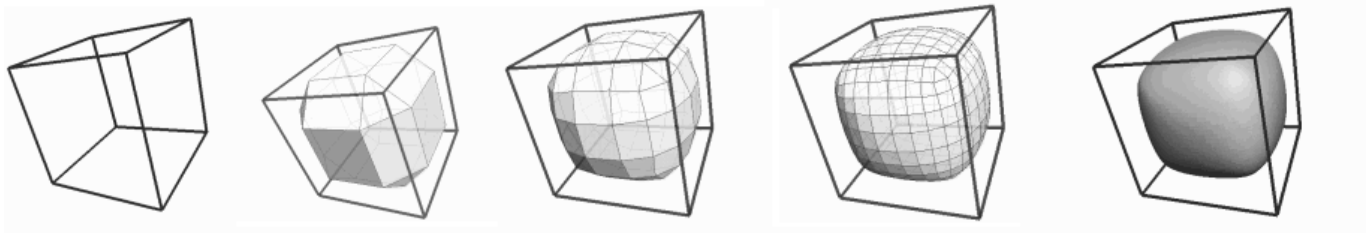
Hay muchos tipos de superficies, pero casi todos pueden traducirse o representarse como NURBS, a veces recortadas. De hecho, la mayoría de los paquetes de software (al menos los más técnicos) transforman cualquier otra posibilidad en una NURBS para el intercambio de archivos. Hay también superficies de Bézier y NURBS triangulares en vez de cuadriláteras, en ellas todo lo que hemos dicho sobre coorde-

nadas cartesianas se reemplaza por coordenadas de área o baricéntricas. 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.

6.2. Subdivision Surfaces

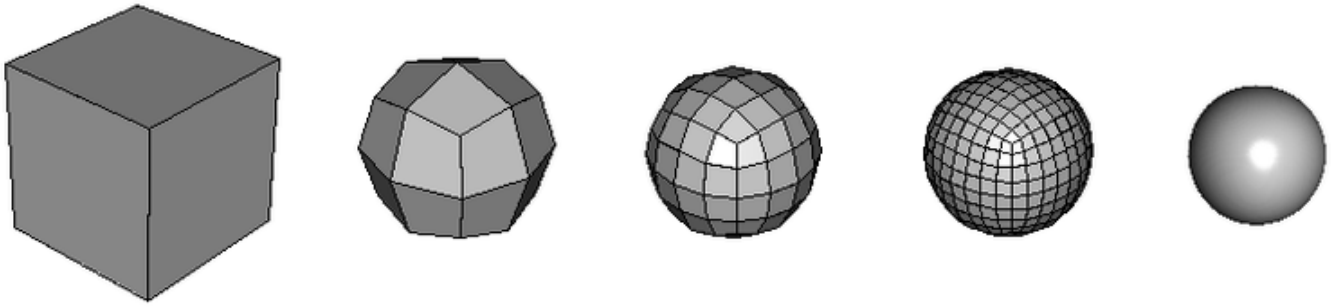
La idea general de las *subdivision surfaces* es partir de una forma muy basta para refinarla en un proceso recursivo, cuyo límite es la superficie buscada. Hay muchos métodos para construirlas, veremos dos: Doo-Sabin y Catmull-Clark.





Para el método de Doo–Sabin (que se muestra en la figura anterior) en cada paso se une el centro de cada cara (promedio de vértices) 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 2^0 grado.

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

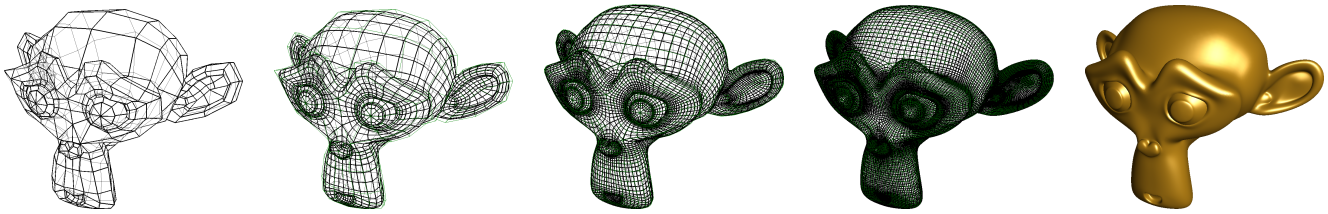


Para cada cara (de v vértices) ponemos un punto en el centro: $F_i = (\sum P_j)/v$.

Para cada arista también se agrega un punto medio: $R_i = (P_{i0} + P_{i1})/2$. El centro R_i de la arista se mueve al punto R_i' promedio entre el punto en que estaba y el promedio de los centros de las dos caras adyacentes en esa arista: $R_i' = (R_i + (F_{i0} + F_{i1})/2)/2 = (P_{i0} + P_{i1} + F_{i0} + F_{i1})/4$.

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

$$P' = \frac{F + 2R + (c-3)P}{c} = \frac{4R' - F + (c-3)P}{c}$$ (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 está 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.