TEMA 1: MODELOS DE REPRESENTACIÓN DE OBJETOS 3D

1.1. MODELOS DE SUPERFICIES

Existen varias razones para querer representar un objeto mediante un modelo de superficie:

- Cuando el objeto mismo es una superficie que podemos suponer sin grosor (por ejemplo, la chapa metálica del capó de un vehículo). Este tipo de representación nos permite visualizar superficies abiertas, mientras que los sólidos se caracterizarán por tener su superficie necesariamente cerrada sobre sí misma.
- Cuando tan sólo nos interesa visualizar su aspecto visual externo, sin detalles sobre su estructura interna, aunque el objeto ocupe un cierto volumen.
- Cuando deseamos realizar una visualización en tiempo real, y para ello utilizamos hardware o software gráfico que está sólo preparado para visualizar polígonos.

En cualquiera de estos casos es conveniente utilizar una representación de la superficie del objeto. En principio la información sobre una superficie debe dar cuenta de su geometría, de sus propiedades visuales (cómo se comporta frente a la luz [1]) y quizás también de alguna propiedad física (como la elasticidad) si se va a efectuar una simulación física sobre el objeto.

Si la totalidad del objeto consta de diferentes partes (por ejemplo, varios polígonos o varios trozos definidos por diferentes ecuaciones), entonces podemos añadir también información topológica, es decir, sobre cómo estas diferentes partes se conectan entre sí para formar la superficie. Con esta información adicional el modelo se conoce como una **representación de frontera** del objeto *(b-rep: boundary representation*). Esta representación de frontera se utiliza frecuentemente en combinación con un modelo sólido, ya que una de ellas se puede reconstruir a partir de la otra.

En general, las diferentes representaciones no se excluyen entre sí, y muchos programas suelen combinarlas, pasando de una a otra según convenga.

Una característica que suele exigirse a las superficies representadas es que sean **variedades bidimensionales** (en inglés: superficies *2-manifold*), es decir, que no existan puntos singulares donde la superficie se intersecte consigo misma o se abra en varias *hojas*. Ver figura 1.1

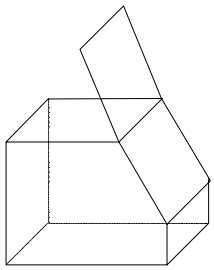


Figura 1.1.: Ejemplo de una superficie que NO es 2-mainfold

Además de la información geométrica y topológica, es frecuente añadir a la representación de los objetos datos adicionales requeridos para su visualización (como el color, las propiedades visuales del material, textura, etc.) o para efectuar procesos de simulación física (distribución de densidad, temperatura, composición).

[1] Ver el modelo Luz-Superficie

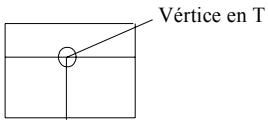
1.1.1. MODELO POLIÉDRICO

Consiste en definir el objeto a través de una superficie formada por polígonos que comparten sus aristas y vértices, es decir, de un poliedro, que puede ser abierto o cerrado. La exigencia de que sea una variedad bidimensional se traduce, en este caso, en la restricción de que los polígonos no se intersecten entre sí excepto en las aristas, y que en una arista no puedan confluir más de dos polígonos. En un vértice pueden coincidir cualquier número de polígonos.

Además, una buena representación poligonal debe tener en cuenta:

- Evitar la degeneración de los polígonos ,que se produce por ejemplo, si uno de sus lados tiene longitud cero. Los triángulos degenerados son especialmente problemáticos, ya que resulta un objeto de área nula, normalmente invisible, pero que puede provocar errores en ciertos algoritmos.

- Evitar los vértices en T ya que puede provocar problemas a la hora de rellenar el color por interpolación entre los vértices y otras singularidades en ciertos algoritmos.



- Utilizar en lo posible primitivas poligonales que permitan especificar el objeto sin repetir varias veces los vértices que pertenezcan a varios polígonos[2].

La representación basada en polígonos tiene la ventaja de que permite hacer una visualización muy rápida, porque únicamente se deben realizar una serie de operaciones lineales muy eficientes, tanto para la proyección sobre la pantalla como para el posterior rellenado de los polígonos, gracias a lo cual es muy adecuada para representaciones en tiempo real[3]. Como veremos después, su visualización mediante trazado de rayos, en concreto el cálculo de intersecciones[4] también resulta especialmente simple.

Sin embargo, hay también desventajas: los objetos complejos o de origen natural raras veces están formados por polígonos exactos, por lo que en su representación se deben realizar aproximaciones, haciendo que la superficie aparezca formada por 'trozos' no suaves cuyo aspecto artificial es visible.

1.1.1.1. Representación por Polígonos Aislados

En este caso un poliedro se representa mediante una lista de polígonos, siendo cada uno de éstos una lista de vértices o de aristas (ver el anexo 1 para una caracterización matemática más completa). Hay que tener cuidado a la hora de definir polígonos con más de tres puntos, ya que cuatro puntos no tienen porqué ser coplanarios en el espacio de tres dimensiones, y al proyectarlos para formar una imagen bidimensional los resultados pueden no estar bien definidos (ver un ejemplo en la figura 1.1.1.1.A).

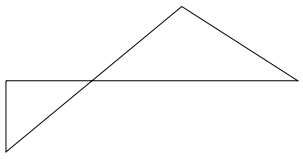
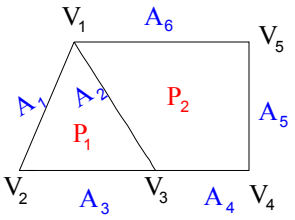


Figura 1.1.1.1.A.: Ejemplo de *bowtie* (bow=arco, tie=pajarita), una figura formada por cuatro puntos no coplanarios cuya proyección en 2D puede tener diferentes aspectos ('pajarita' en este caso)

[2] Ver apartado 1.1.1.2.: Primitivas poligonales
[3] Ver Tema 3: Sistemas de Visualización en Tiempo Real
[4] Ver Tema 4: Modelos de Iluminación Global. Trazado de Rayos

Para poder incorporar información topológica, de relación de vecindad entre los polígonos, hay que construir una estructura de datos en la que se especifiquen las referencias adecuadas.

Vamos a ver un par de formas de representar un conjunto de polígonos aislados en forma de tabla, de manera que sea posible extraer información topológica. La primera forma contendría:



- Una lista de vértices, una de aristas y una de polígonos.

Tabla de Vértices

V_1: X_1 Y_1 Z_1
⋮
V_5: X_5 Y_5 Z_5

Tabla de Aristas

A_1: V_1 V_2
A_2: V_1 V_3
⋮

Tabla de Polígonos

P_1: A_1 A_2 A_3
P_2: A_2 A_4 A_5 A_6
⋮

- Para cada vértice, definido por tres coordenadas, tendríamos referencias a las aristas que lo utilizan y (opcionalmente) a los polígonos que lo contienen.
- Para cada arista, tendríamos los dos vértices que la forman y, si se desea, los dos polígonos que la comparten.
- Para cada polígono guardaríamos las aristas que lo forman

.

A esto se le suele denominar representación doblemente indexada. Con esta estructura será muy sencillo, por ejemplo, comprobar si dos polígonos se tocan, pues deberán compartir al menos una arista.

O bien podríamos simplificar la representación eliminando las aristas:

Tabla de Polígonos

P_1: V_1 V_2 V_3
P_2: V_1 V_3 V_4 V_5
⋮

Tabla de Vértices

V_1: X_1 Y_1 Z_1
⋮
V_5: X_5 Y_5 Z_5

(INDEXADA)

- Una lista de vértices y una de polígonos.
- Para cada polígono los vértices que lo definen.

O también podríamos directamente poner las coordenadas de los vértices que forman cada polígono, aunque estas se repitan en los vértices compartidos.

Tabla de Polígonos

P_1: X_1 Y_1 Z_1
X_2 Y_2 Z_2
X_3 Y_3 Z_3
P_2: ----------

(NO INDEXADA)

Estas representaciones se pueden modificar de diversas formas, eliminando información redundante, o añadiendo más información para facilitar la búsqueda de relaciones entre los diferentes elementos. Aparece por tanto, la necesidad de un compromiso entre la cantidad de almacenamiento requerida, la complejidad de la estructura de datos y la rapidez de acceso a la información geométrica y topológica.

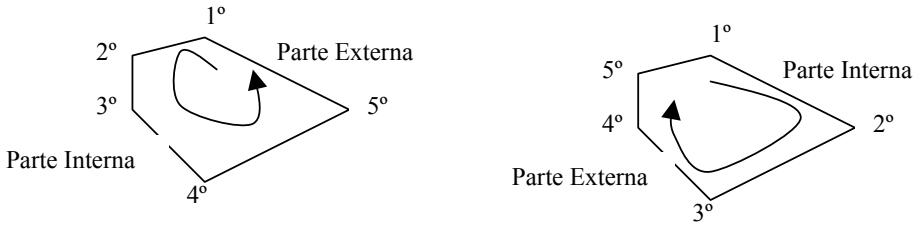
Un mismo vértice puede formar parte de diferentes polígonos. Si guardamos la información de cada vértice (posición, vector normal, color, etc.) una sola vez y luego, cuando describamos los polígonos de los que forma parte, hacemos referencia a él mediante su índice en la tabla de vértices entonces se dice que estamos describiendo el objeto de forma **indexada**. Si, por el contrario, cada vez que un vértice aparece en la descripción de algún polígono repetimos toda la información correspondiente al vértice, entonces estamos trabajando con una estructura **no indexada o explicita**.

La estructura indexada tiene la ventaja de ahorrar espacio de almacenamiento. Sus desventajas son que no permite la discontinuidad de color u otra magnitud asociada a los vértices, ya que dos polígonos que comparten el mismo vértice tienen necesariamente el mismo color en ese punto. Esto produce un efecto de suavizado del color de la superficie que a veces resulta irreal (ver Figura 1.1.1.1.B). Otra desventaja es que se produce un ligero aumento en el tiempo necesario para una consulta de datos sobre los vértices, ya que primero debe averiguarse el índice y luego buscar la información deseada en la tabla.

En caso de emplear una lista no indexada, se puede representar un poliedro con una única lista de polígonos, definidos por las coordenadas de cada uno de los vértices que lo componen. Al no ser necesaria la consulta de una tabla de vértices, se consigue acceder de forma mucho más rápida a las coordenadas, lo que puede ser un ahorro de coste importante en aplicaciones de tiempo real. Sin embargo, tiene el inconveniente de aumentar considerablemente el espacio de almacenamiento necesario, ya que las coordenadas de los vértices aparecen repetidas cada vez que un mismo vértice pertenece a varios polígonos distintos.

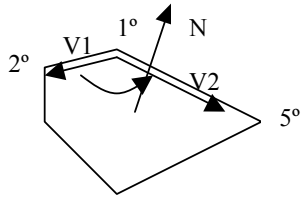
1.1.1.2 Consideraciones geométricas de los modelos Poligonales.

Una característica importante que tenemos que tener en cuenta a la hora de trabajar con modelos poligonales será el **orden en el cual se especifican los vértices** que lo componen. Este orden influye en algunos algoritmos que se emplean para eliminación de superficies ocultas. El criterio de la regla del sacacorchos es el que define el criterio de especificación de vértices, o sea la parte externa de un polígono es aquella obtenida recorriendo sus vértices en sentido antihorario:



Otro punto importante a tener en cuenta a la hora de trabajar con polígonos la obtención del plano sobre el cual cae el polígono (ya que hemos comentado que los todos los vértices de un polígono deben ser coplanarios). Para el cálculo de la ecuación del plano que ocupa un polígono y comprobar que todos los

vértices son coplanarios bastará con elegir tres de sus vértices (que no estén alineados) y calcular el vector normal al plano como el producto vectorial de las dos aristas definidas por dichos vértices:



$$\vec{N} = \vec{v}_1 \times \vec{v}_2$$

Y la ecuación del Plano será de la Forma:

$$A \cdot x + B \cdot y + C \cdot z + D = 0$$

Donde A=Nx, B=Ny y C=Nz y la D puede obtenerse al sustituir cualquiera de los vertices en la ecuación del plano y despejar. El orden del producto vectorial también afecta a la dirección de la normal y es necesario tenerlo en cuenta. La regla que se aplica es la misma que la vista para el orden de los vertices.

Teniendo la ecuación del plano y según el criterio de que la normal apunta hacia fuera del polígono ahora para un punto cualquiera P (x,y,z), si sustituimos sus coordenadas en la ecuación del plano que cotiene al polígono y el resultado es mayor que cero diremos que el punto esta fuera del plano (o delante) y si el valor es negativo diremos que esta dentro (o detrás). Este tipo de comprobaciones nos servirán en métodos de partición especial basados en superficies planas.

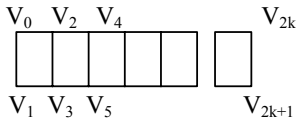
Comentar finalmente que el cálculo de normales a los polígonos es importante para los modelos de iluminación que trataremos en el tema 2 y en el tema 4.

1.1.1.3. Primitivas Poligonales

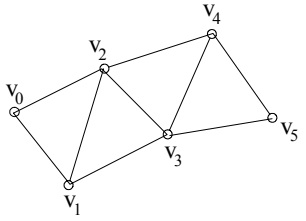
Las primitivas poligonales son grupos de polígonos que se describen de forma conjunta para ahorrar espacio de almacenamiento y coste de visualización en tiempo real, razón por la cual son ampliamente utilizadas.

Algunas de las primitivas más utilizadas son:

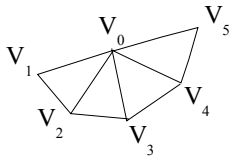
- **Tira de cuadriláteros**: los primeros cuatro vértices definen el primer cuadrilátero, y cada nuevo par define otro cuadrilátero formado por éste par de vértices y el anterior. Podemos ver que este tipo de primitiva ahorra, respecto de la especificación de polígonos aislados, casi la mitad de espacio. Sin embargo, tiene el inconveniente de que no se garantiza que en cada cuadrilátero los vértices sean coplanarios.



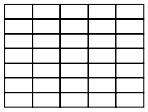
- **Tira de triángulos**: con los tres primeros puntos se construye un triángulo y los demás se forman añadiendo sucesivos puntos. Cada nuevo triángulo se forma por los tres últimos vértices añadidos, de tal forma que con N puntos se obtienen N-2 triángulos.



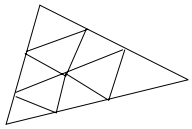
- **Abanico** (Fan): se da un primer punto y luego el resto siguiendo un abanico. Aparecen N-2 triángulos formados por el primer vértice y los vértices i, i+1 (i diferente de 1). Se emplea para conseguir formas imposibles de lograr con la tira de triángulos. Esta primitiva, al igual que la anterior, tiene la ventaja de que no es necesario comprobar la coplanariedad.



- **Malla rectangular**. Se utiliza directamente una matriz de n por m vértices.



- **Malla triangular**. Apenas se emplea.



En realidad, la mayor parte de aplicaciones y librerías gráficas descomponen posteriormente todas estas primitivas en triángulos durante el proceso de visualización.

1.1.1.4. Modelo de Representación Polígonal sobre OpenGL.

La librería gráfica OpenGL es por naturaleza una librería orientada al trabajo con modelos poliédricos, por tanto nos será fácil realizar representación de esta naturaleza. Puntualizaremos que se trata de una librería de funciones orientada principalmente a modelos interactivos, por ello se premia la rapidez frente al espacio, el tipo de representación poligonal que empleara será por tanto explícita.

Las definiciones de primitivas poligonales en OpenGL se encierran entre las llamadas a las funciones: ***glBegin***(GLEnum tipo_primitiva) y ***glEnd***(void). Entre dichas funciones deberemos especificar la lista de vértices que componen nuestro polígono. La función para pasar las coordenadas de cada vértice es ***glVertex3fv***(GLFloat *coor), donde 'coor' es un vector que contiene las tres coordenadas del vértice.
Los valores normales para el tipo de primitiva son las constantes:

Valor de la Cte GL	Tipo de Primitiva Poligonal
GL_POINTS	Puntos aislados
GL_LINES	Líneas de dos vértices
GL_LINE_STRIP	Línea de cualquier numero de vértices
GL_LINE_LOOP	Línea Cerrada.
GL_POLYGON	Polígono de Cualquier tipo
GL_TRIANGLES	Polígonos de tres lados
GL_TRIANGLE_STRIP	Tira de Triangulos
GL_QUADS	Polígonos de cuatro vertices
GL_QUAD_STRIP	Tira de Cuadrilateros.
GL_TRIANGLE_FAN	Abanico de triangulos.

Tabla de Primitivas Gráficas OpenGL

Por ejemplo la definición de un triangulo en OpenGL seria:

```
float mitrian[3][3]={{0,0,0},{1,1,0},{2,0,0}};

/* ….*/
            glBegin(GL_TRIANGLES);
                  glVertex3fv(mitrian[0]);
                  glVertex3fv(mitrian[1]);
                  glVertex3fv(mitrian[2]);
            glEnd();

/* ….*/
```

1.1.2. SUPERFICIES CURVAS

La ventaja de las representaciones poliédricas es que son más adecuadas para la visualización en tiempo real, cuyas técnicas se basan la proyección y rellenado de triángulos. El problema que plantean es que la mayoría de los objetos no son realmente poliedros, sino que están compuestos por superficies continuas y curvadas.

Para representar un objeto complejo se puede utilizar una descomposición de la superficie en trozos o parches (*patches*) triangulares o cuadrangulares, cada uno de los cuales se suele modelar con superficies paramétricas (superficies de Bezier, NURBS...). Esta representación permite modificar la forma de la superficie con mucha facilidad: en lugar de cambiar las coordenadas de un gran número de vértices, solo necesitamos cambiar unos cuantos parámetros en las ecuaciones. Una vez terminado el modelado si queremos hacer una visualización en tiempo real siempre resultará posible convertir estas superficies en una aproximación poligonal. Existen otros métodos de visualización, como el trazado de rayos, que sí pueden trabajar directamente con una representación exacta.

Los problemas que plantea la representación contínua a trozos son, por una parte, la dificultad de controlar que la forma de la superficie sea exactamente la que queremos y, por otro, la complejidad de mantener las restricciones (continuidad, derivabilidad) en las intersecciones de los trozos.

Para definir una superficie curvada y suave de forma exacta debemos recurrir a una representación analítica, es decir, mediante ecuaciones. Dependiendo de la forma de las ecuaciones habrá que evaluar la superficie (hallar sus puntos) utilizando distintos métodos. En general tenemos los siguientes tipos:

- Ec. implícita : $f(x, y, z) = 0$
- Ec. explícita : p.ej. $y = f(x, z)$
- Ec. paramétrica : p.ej. una superficie $(x, y, z) = f(u, v)$

En tres dimensiones una ecuación explícita siempre puede transformarse en paramétrica, como podemos ver en el siguiente ejemplo:

Si $z = f(x, y)$ es la ecuación en forma explícita, entonces la ecuación paramétrica podemos definirla como :

$$\left.\begin{array}{l} x = u \\ y = v \end{array}\right\} z = f_z(u, v) \quad \Rightarrow$$

$$\Rightarrow (x, y, z) = (u, v, f_z(u, v)) = f(u, v)$$

1.1.2.2. Superficies Implícitas : Cuádricas, Isosuperficies y Superficies Equipotenciales.

Cuadricas:

Comenzaremos por revisar el modelado de formas descritas en base a funciones matemáticas de tipo cuadrático que se expresan de forma implicitas.

En este caso las expresión general que identificaría a dichas superficies sería:

$$ax^2 + by^2 + cz^2 + 2dxy + 2eyz + 2fxz + 2gx + 2hx + 2jz + k = 0$$

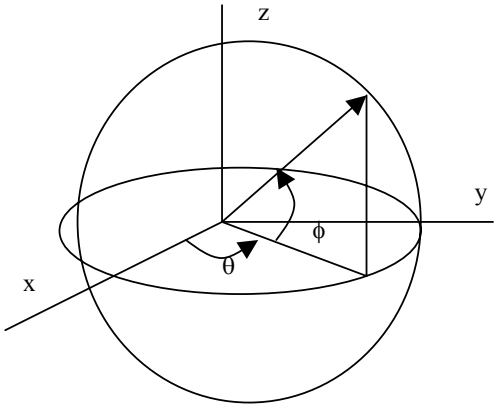
Ahora en función de los valores de estos parámetros tenemos distintos de cuádricas:

- **Esfera:** En este caso los coeficiente no nulos son a,b,c y el termino independiente representa el cuadrado del radio de las esfera. En algunos caso se suele hacer un cambio en la representación de la cuádrica para facilitar su evaluación, elegiendo representación paramétricas,en el caso de la esfera se suele representar en coordenadas polares, siendo los ángulos de barrido los parámetros.

$$ax^2 + by^2 + cz^2 - r^2 = 0$$

Ec. Implicita de la Esfera.

$$x = r\cos\phi\cos\theta, -\frac{\pi}{2} \le \phi \le \frac{\pi}{2}$$
$$y = r\cos\phi\sin\theta, -\pi \le \theta \le \pi$$
$$z = r\sin\phi$$



Representación polar de la esfera

Ejemplo OpenGL.

Veamos en el siguiente ejemplo como construir la representación de una cuadrica de tipo esférico basada en una descomponsición polígonal que utilice las tiras de triangulos de OpenGL. La función ejemplo recibirá como parámetros el radio de la esfera y las divisiones en latitud (ϕ) y longitud(θ)

```
void Esfera_T_Strip(float radio, int nlatitud , int nlongitud)
{
 float inct,incf;
int i,j;
float vertice[3];

      inct=2*PI/nlongitud;
      incf=PI/nlatitud;

      for(i=0;i<nlatitud;i++)
      {
            glBegin(GL_TRIANGLE_STRIP);

            vertice[0]=vertice[1]=0;
            vertice[2]=-radio;
            glVertex3fv(vertice);
            for(j=1;j<nlongitud-1;j++)
            {
                  vertice[0]=radio*cos(i*inct)*cos(j*incf-0.5*PI);
                  vertice[1]=radio*cos(i*inct)*cos(j*incf-0.5*PI);
                  vertice[2]=radio*sin(j*incf-0.5*PI);
                  glVertex3fv(vertice);

                  vertice[0]=radio*cos((i+1)*inct)*cos(j*incf-0.5*PI);
                  vertice[1]=radio*sin((i+1)*inct)*cos(j*incf-0.5*PI);
                  glVertex3fv(vertice);

            }
            vertice[0]=vertice[1]=0;
            vertice[2]=radio;
            glVertex3fv(vertice);
      }
}
```

- **Ellipsoide:** En este caso la forma de expresión de los parametros es algo más compleja y se puede obtener al desarrollar la siguiente expresión implicita:

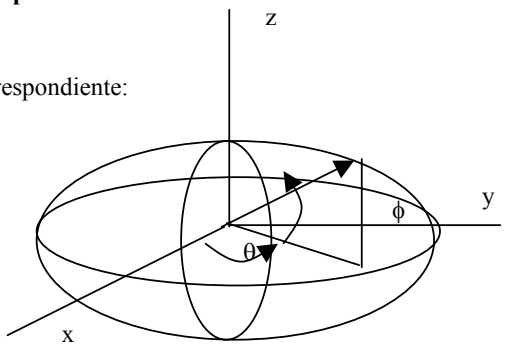
$$\left(\frac{x}{r_x}\right)^2+\left(\frac{y}{r_y}\right)^2+\left(\frac{z}{r_z}\right)^2-1=0$$ **Ec. Implicita de un Elipsoide**

En este caso también tenemos la expresión paramétrica correspondiente:

$$x = r_x \cos\phi\cos\theta, -\frac{\pi}{2} \le \phi \le \frac{\pi}{2}$$

$$y = r_y sin\phi sin\theta, -\pi \le \theta \le \pi$$

$$z = r_z sin\phi$$



Ec. Paramétrica del Elipsoide

Otro ejemplo común de cuadríca es el toro.

- **Supercuadricas:** Se trata de una generalización de las cuádricas donde se añaden parámetros adicionales que afectan al grado de la ecuación implícita, con ello se obtienen variaciones intersantes de la forma. Por ejemplo el Superellipsoide sería una variante del elipsoide anterior donde se añade un parámetro s1 y s2 que afectan al grado de la ecuación. En la figura se pueden observar los efectos de los cambios en s1 y s2.

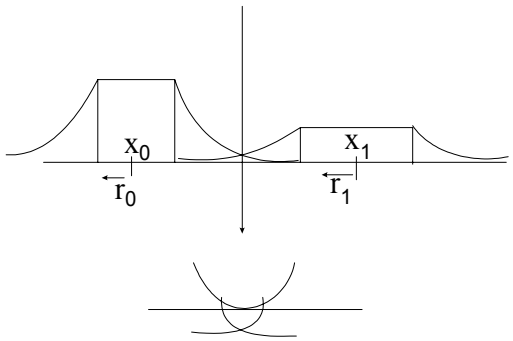
$$\left[\left(\frac{x}{r_x}\right)^{2/s_2}+\left(\frac{y}{r_y}\right)^{2/s_2}\right]^{s_2/s_1}+\left(\frac{z}{r_z}\right)^{2/s_1}-1=0$$ **Ec. Implicita de un Superelipsoide.**

Isosuperficies y Superficies Equipotenciales.

Otro método de modelado especial de superficies consiste en definirlas como funciones en forma implícita, es decir, en la forma F(x,y,z)=0 y luego buscar alguna manera de visualizarlas. Esto es equivalente a definir un campo escalar en el espacio y luego tomar todos los puntos x,y,z en los que esa magnitud toma un valor constante.

De esta forma, a partir de una función $g(x,y,z)=T\ \ (umbral\ o\ threshold)$, podemos definir otra función en forma implícita equivalente, sin más que hacer : $f(x,y,z)=g(x,y,z)-T=0$

Este tipo de objeto se denomina también *isosuperficie*. Si el campo escalar varía de forma suave (existe su gradiente o derivada respecto a la posición) entonces se denomina *potencial* (como el generado por un campo de fuerza en física). En ese caso la isosuperficie se denomina *superficie equipotencial*.

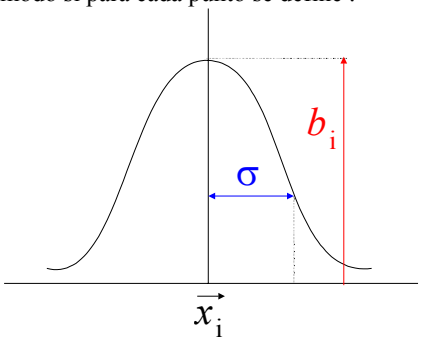


Ejemplo: un campo escalar (las tres dimensiones han sido reducidas a una) creado por dos objetos diferentes de radios r_0 y r_1. En este caso la 'superficie equipotencial' serían puntos aislados, que se hallarían sumando las dos contribuciones y buscando aquellos puntos de altura umbral T.

Una forma de poder controlar la forma de esas superficies para utilizarlas en el modelado de objetos es hacer que la función F(x,y,z) ó g(x,y,z) sea la suma de varias contribuciones generadas por diferentes objetos básicos instanciados en la escena, normalmente esferas o elipsoides. El símil físico sería que cada uno de esos objetos genera un cierto campo de fuerza (gravitatorio, eléctrico...), un potencial a su alrededor, en cada punto del espacio. La superficie equipotencial envolverá a los objetos básicos como una película elástica de aspecto suave. Es por ello que estos 'objetos globulares' (*blobs* o *blobby objects*) que se utilizan ampliamente para representar formas orgánicas, en las que la piel o la concha recubren las estructuras internas (esqueleto, músculos).

Un ejemplo popular de esta aproximación es la *combinación de funciones gaussianas* en las que el potencial en cada punto producido por cada una de las componentes es una curva normal o gaussiana.

De este modo si para cada punto se define :



$\vec{x}_i \rightarrow posicion$

$b_i \rightarrow fuerza = altura$

$r_i(\sigma) \rightarrow desviacion\ tipica$

Entonces podemos definir el potencial en cada punto como :

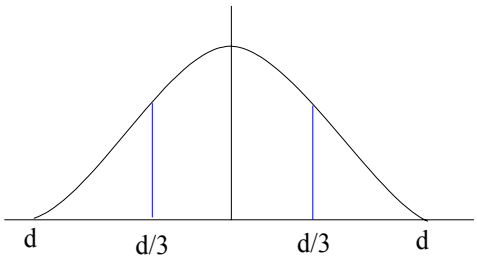
$$f_i(\vec{x}_i, b_i, r_i) = b_i e^{-\frac{1}{r_i^2}(\vec{x}-\vec{x}_i)^2}$$

De forma que la superficie equipotencial quedaría como sigue :

$$f(x,y,z) = \sum_i b_i e^{-\frac{1}{r_i^2}(\vec{x}-\vec{x}_i)^2} = T$$

Este tipo de objetos se conoce también como *blobs* en algunos paquetes de animación.

Otro tipo son las *metaballs*, definidas mediante polinomios, y por tanto más fáciles de evaluar, como podemos ver en su ecuación general (ver figura 1.1.2.2).

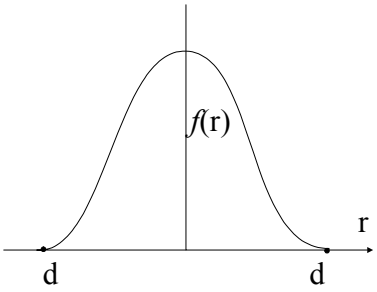


$$f(r)=\begin{cases} b\left(1-3r^2/d^2\right) & si\ r\in\left[0,\mathrm{d}/3\right] \\ \frac{3}{2}b\left(1-r/d\right)^2 & si\ r\in\left[\mathrm{d}/3,\mathrm{d}\right] \\ 0 & si\ r>\mathrm{d} \end{cases}$$

Con $r_i=\left(\vec{x}-\vec{x}_i\right)$ y **b** la fuerza, **r** la distancia entre un punto y el centro y **d** el radio, es decir, el tamaño de la componente.

Otra posibilidad son los *soft objects*. En los que la ecuación general es :

$$f(r)=\begin{cases} 1-\frac{22r^2}{9\mathrm{d}^2}+\frac{17r^4}{9\mathrm{d}^4}-\frac{4r^6}{9\mathrm{d}^6} & si\ r\in\left[0,\mathrm{d}\right] \\ 0 & si\ r>\mathrm{d} \end{cases}$$



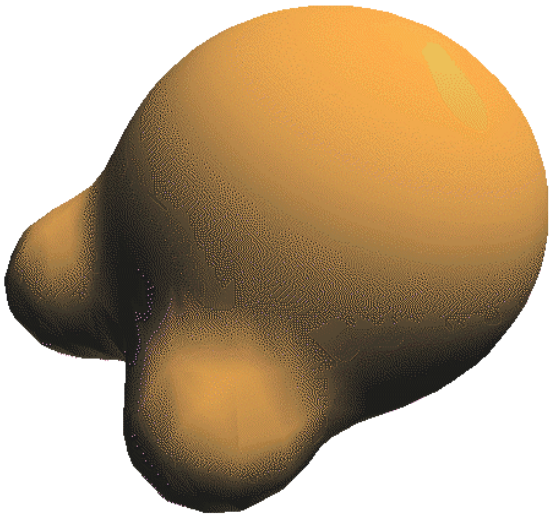


Figura 1.1.2.2.: Ejemplo de superficie equipotencial generada con tres esferas

Visualización de Isosuperficies: Marching Cubes

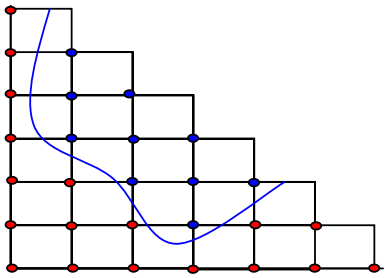
La visualización de cualquier isosuperficie puede realizarse básicamente por dos métodos diferentes, bien por generación de polígonos **Marching Cubes** o por trazado de rayos[5]. También pueden dibujarse ciertas aproximaciones durante la fase de modelado, que requiere una visualización rápida **Aproximaciones durante la edición**[6].

El algoritmo del *marching cubes* nos describirá la superficie en forma de polígonos descritos por las intersecciones de la superficie a evaluar con una descomposición del espacio en unas formas geométricas base. El algoritmo clásico comienza realizando una descomposición del espacio afectado por la superficie en cubos de un tamaño determinado (geometrías base), estos cubos se suele denominar voxel por analogía a los pixel bidimensionales (en otros puntos de este tema los volveremos a nombrar).

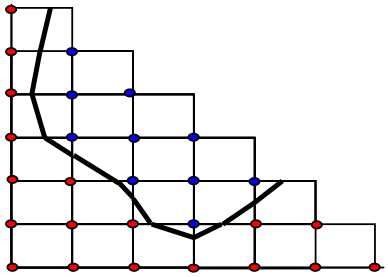
Dada la descomposición en voxels, cada uno de estos voxel puede encontrarse en una de estas tres situaciones:

- Completamente fuera de la superficie.
- Completamente dentro de la superficie
- Parcialmente dentro o sea intersectado por la superficie.

El algoritmo después de realizar esta clasificación se dedicara a la determinar la forma del polígono de intersección entre la ecuación de la superficie y el cubo intersectado. El numero de posibles formas de intersección esta limitado. Veamos un ejemplo sencillo en 2D y luego lo generalizaremos a 3D. En la imagen se observa una curva sobre un plano dividido en una retícula. La curva limita un espacio dentro de cuadrados interiores otro de exteriores y un espacio de cuadrados intersectados. En este caso lo que vamos a hacer es segmentar la curva en tramos rectos, que definen los puntos de intersección de la curva con los cuadrados:

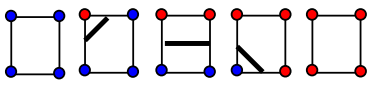


Curva real sobre la muestra tomadas de la misma.



Curva poligonal generada de la muestras.

El tipo de intersecciones esta limitado como hemos comentado anteriormente a unas cuantas posibilidades, en el caso 2D las formas de intersección curva-celdas se limita a cinco posibilidades (dos de ellas no implican intersección). El problema será detectar el punto de corte exacto, esto se puede hacer por un método de interpolación entre el valor de la función que define la curva en el punto interior (azul y en exterior (rojo) para cada segmento de celta intersectado.



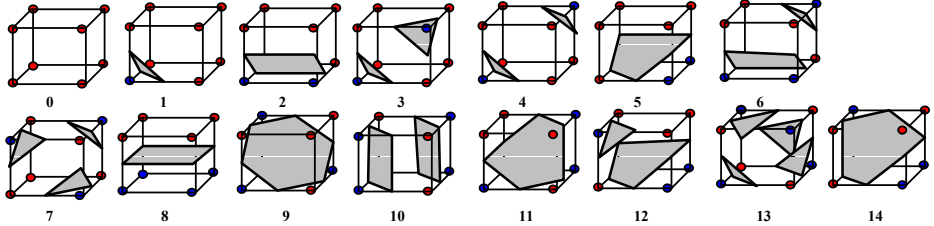
El resultado de esta representación como en toda discretización dependera de la precisión elegida, que en esta caso es función del tamaño de las celdas.

La generalización a tres dimensiones es relativamente sencilla de comprender, aunque la implementación es bastante más laboriosa.

En este caso las intersecciones en lugar de ser segmentos rectos son polígonos y la casuística de intersección se complica existiendo 16 posibilidades distintas:

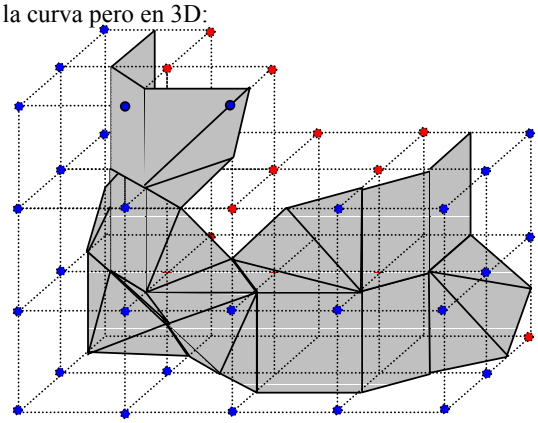
[5] Ver Tema 3: Modelos de Iluminación Global. Trazado de Rayos

[6] Ver anexo 8



Posibilidades Topologicamente distintas de intersección enter uan superficie y un cubo.

Aplicado a un ejemplo similar a la curva pero en 3D:



Superficie interpolada.

Existen codificaciones especiales en la numeración de los vértices, etc, para optimizar las clasificaciones de las intersecciones, etc, no obstante suele tratarse de un algoritmo lento y para realizarlo off-line y muchas veces los resultados no son excesivamente buenos pudiendo surgir algunos problemas de desconexión en la malla poligonal que ya se pueden intuir de las formas de intersección presentadas.

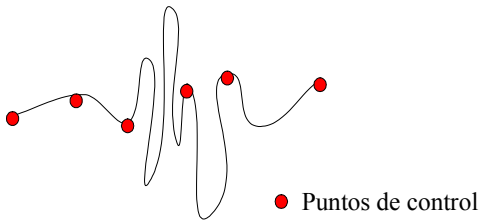
1.1.2.1. Superficies Paramétricas : Superficies de Control.

Vamos a examinar alguna de las formas de proceder con una superficie paramétrica, que mostrarán lo simple que resulta su utilización en ciertos algoritmos y su visualización. Si se emplean las ecuaciones paramétricas para definir una superficie, es posible hallar puntos que pertenezcan a ella (por ejemplo, para poder dibujarla) sin más que ir dando valores a los parámetros *u* y *v*, de tal forma que se recorre la superficie de forma exhaustiva. En muchas representaciones se normaliza la ecuación de manera que los parámetros *u* y *v* solamente tomen valores en el rango [0,1].

Además se pueden calcular muy fácilmente algunos parámetros interesantes (ver Anexo 2), como:

- Los vectores tangentes para los parámetros u, v y el vector normal a la superficie.
- Calcular la distancia recorrida en un camino a lo largo de la superficie.
- Calcular la curvatura (relacionada con las segundas derivadas) en un punto cualquiera de la superficie.
- Calcular los parámetros u,v de un punto desconocido (por ejemplo, el punto donde una recta intersecta a la superficie) mediante aproximaciones sucesivas dando valores a u y v.

El problema de esta aproximación es que la complejidad de las ecuaciones suele aumentar enormemente cuando imponemos muchas restricciones (por ejemplo, que la superficie pase por más y más puntos que nos interesan). También aparecen efectos indeseados; las superficies resultantes pueden tener una forma extraña aunque les forcemos a pasar por ciertos puntos de control, como puede verse en el siguiente dibujo.



Al aparecer ordenadores capaces de representar gráficos, y comenzar su aplicación en el Diseño Asistido por Ordenador (CAD), surgieron grupos de investigación en empresas de automóviles que desarrollaron la teoría y la aplicación de ciertas superficies paramétricas cuya forma se podía definir de manera sencilla a través de **puntos de control**, garantizándose que su forma no sufra variaciones incontrolables. Las curvas más interesantes son construidas a partir de polinomios y cumplen con la siguiente propiedad: si definimos el *cierre convexo* de un conjunto de puntos como el mínimo polígono convexo (en 2D) o poliedro convexo (en 3D) que los contiene, entonces se cumple que la curva o superficie determinada por un conjunto de puntos de control no se sale del cierre convexo de estos puntos.

No vamos a especificar aquí en detalle los diferentes tipos de superficies paramétricas polinómicas con puntos de control. Las más empleadas son las superficies de Bèzier[7], las B_Splines y las NURBS (ver Figura 1.1.2.1.). Su forma general sería:

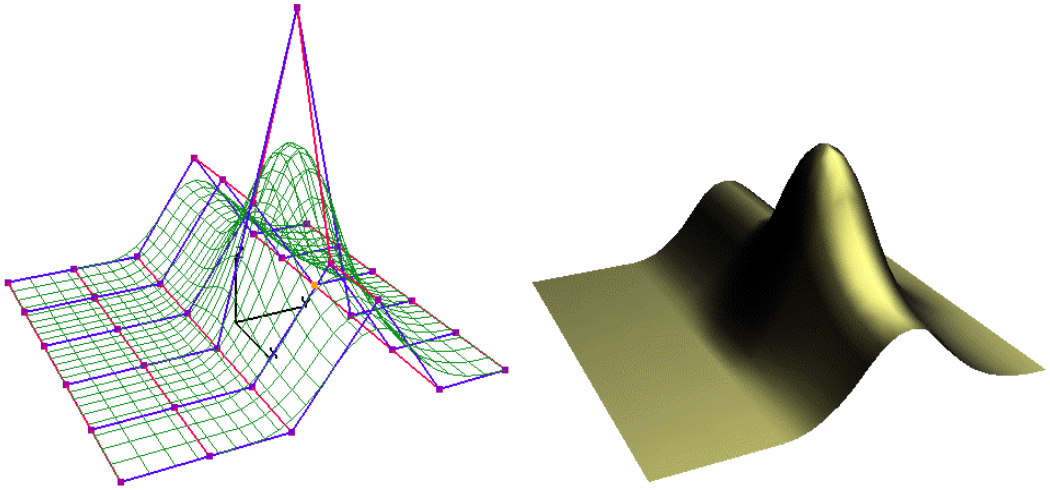


Figura 1.1.2.1.: Ejemplo de superficie diseñada con NURBS

Dentro de estas nociones generales de representación paramétrica existen dos aproximaciones principales a la hora de abordar el modelado de estas superficies o curvas:

- Curvas o superficies de interpolación
- Curvas o superficies de aproximación.

Paramétricas de Interpolación.

Las primeras ofrecen un control más inmediato del modelado y nos permiten asegurar que las curvas o superficies pasan por el conjunto de puntos de control. Esto puede parecer una ventaja pero para el modelado geométrico supone que las restricciones e información que hemos de ofrecer hagan que las superficies de interpolación sean complejas de utilizar. Se emplean sobre todo para interpolación de trayectorias donde si que necesitamos asegurar unos puntos de paso.

El ejemplo más común de las paramétricas de aproximación lo constituyen las **splines naturales,** estas curvas pertenecen a la familia de las splines las cuales se basan en definir la curva a partir de curvas polinomiales a trozos, normalmente el grado de los trozos se selección de grado cúbico. Las curvas (y por extensión superficies) tienen el aspecto de continuidad y suavidad por que se les exigen condiciones de continuidad y derivabilidad en los puntos de unión.

En el caso de la splines naturales se exige continuidad C^2 lo que significa primeras y segundas derivadas continuas. Estas condiciones nos servirán para determinar los parámetros que definen las curva.

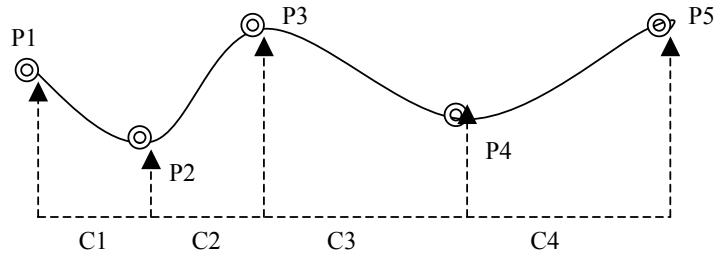
La forma de cada uno de estos trozos de curva es:

[7] Para más detalles sobre las superficies de Bèzier ver el anexo 3

$$C_1(u) = \sum_{i=0}^{3} a_{1i} u^i \qquad 0 \le u \le 1$$

$$C_1(u) = a_{10} + a_{11}u + a_{12}u^2 + a_{12}u^3$$

Es importante puntualizar que esta es la forma general para cada una de las tres dimensiones, siendo el parámetro la u, por tanto tendríamos una ecuación de este tipo para *x*, otra para *y* y otra para *z*
El numero de trozos de curva que necesitamos para definir una spline natural depende del numero de puntos por los que queramos forzar el paso, el numero de trozos es equivalente al numero de puntos de paso menos 1:



Por ejemplo en el caso de la figura el número de cúbicas que tenemos que combinar es cuatro.
Cada una de las cúbicas viene definida por cuatro parámetros (del a0 al a3), por tanto para definir completamente las curva necesitamos saber el valor de *4*(n-1)* parámetros. Para determinar esos parámetros se establece un sistema de *4n –4* ecuaciones (siempre *n* es el numero de puntos) donde las incógnitas son los parámetros.
De las condiciones de continuidad de las derivadas primera y segunda en los puntos medios y del paso por los puntos control podemos obtener 4n-6 ecuaciones y las otras 2 ecuaciones se consiguen asignando unos valores de tangencia en los extremos, estos valores de tangencia determinan también en buena medida la forma final de la *spline* natural.
Veamos esto en un ejemplo simple con solo tres puntos de paso (n=3) P1,P2, y P3. El numero de curvas será de dos y por tanto necesito 4*3-4 parámetros o sea 8, 4 por curva como hemos indicado.
Las ecuaciones se obtienen de la siguiente forma:

Sabemos que en el punto inicial la curva C1 pasa por P1 por tanto:

1. $C_1(0) = P1$

En el Punto medio la curva C1 pasa por P2, esto corresponde al valor 1 de su parámetro de evaluación. por continuidad C2 en su valor de parámetro 0 tambien pasa por P2 por tanto tenemos dos nuevas ecuaciones:

2. $C_1(1)=P2$
3. $C_2(0)=P2$

Otra ecuación la obtenemos de la condición de paso de C2 por el punto P3 en su valor de parámetro 1.

4. $C_2(1)=P3$

*Otras 2*n-2 ecuaciones salen de las condiciones de continuidad de la primera derivada y segunda derivada en los puntos de unión, en este caso serán dos ecuaciones:*

5. $C_1'(1)=C_2'(0)$
6. $C_1''(1)=C_2''(0)$

Las dos ecuaciones que nos faltan las sacamos como hemos comentado anteriormente de asignar un valor de tangencia en los extremos o sea un P1' y un P3'

7. $C_1'(0)=P1'$
8. $C_2'(1)=P3'$

La forma final de las ecuaciones segun la forma de las curvas vista anteriormente sería:

1. $a_{10} = P1$
2. $a_{10}+a_{11}+a_{12}+a_{13}=P2$
3. $a_{20}=P2$
4. $a_{20}+a_{21}+a_{22}+a_{23}=P3$
5. $a_{11}+2a_{12}+3a_{13}=a_{21}$
6. $2a_{12}+6a_{13} = 2\ a_{22}$

7. $a_{11}=P1'$
8. $a_{21}+2a_{22}+3a_{23}=P3'$

Este desarrollo ser deberá hacer para cada una de las coordenas ya que P1,P2 y P3 tienen una componente x, y, z, por tanto las ecuaciones que resolvemos realmente son el triple de las comentadas.

Las *splines* naturales tienen problemas de control local ya que las ecuaciones que escribimos hacen que existan dependencias que se trasmiten de unos puntos de control a otros. Por otro lado existe un problemas importante conforme aumenta el numero de puntos ya que el sistema de ecuaciones al resolver crece de una forma bastante importante.
Existen algunas mejoras de estas curvas de interpolación como por ejemplo las *splines cardinales*, o las *interpolaciones de hermite* que se basan en ideas paracidas a las funciones base del tipo que comentaremos en el siguiente apartado de splines de aproximación, en todos los casos se trata de minimizar los problemas apuntados en el párrafo anterior.

Curvas y Superficies Parámetricas de Aproximación

En este caso como hemos comentado el polinomio de control no forma parte de la curva o superficie si no que es un modo de producir las deformaciones en la misma. Otra característica que es habitual en estas curvas el utilizar unas funciones polinómicas base que se combinan para formar la curva que se visualiza. El tipo y naturaleza de estas funciones base se emplea normalmente para clasificar distintas categorías de superficies y curvas de aproximación. Vamos a repasar brevemente un par de tipos las curvas de Bezier y las B-Splines.

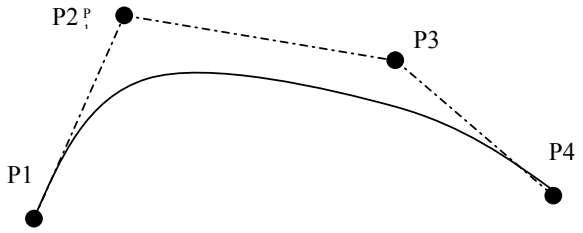
Curvas y Superficies de Bezier.

En la curvas de Bezier se cumple que la curva pasa por los puntos extremos del polinomio del control y se aproxima al resto. En este caso las funciones base son los polinomios de Berstein y la forma que toma la curva es:

$$C(u)=\sum_{i=0}^{n} B_{i,n}(u)P_i \qquad 0\le u\le 1$$

con

$$B_{i,n}(u)=\frac{n!}{i!(n-1)!}u^i(1-u)^{n-i}\ con \quad \sum_{i=0}^{n} B_{i,n}(u)=1\forall u$$



Donde la *B* son los polinomios de Berstein y las P_i son los puntos de control, n es el numero de puntos. No obstante esta forma de las curvas de Bezier tiene el problema que el grado del polinomio base depende del numero de puntos. Para evitar esto se suelen emplear para el modelado de curvas trozos de curvas de Bezier con hasta 4 puntos en el polinomio de control y a los cuales se les exigen condiciones de continuidad para generar curvas con más puntos de control.
La principal característica que deben cumplir estas curvas es que el último punto de la primera curva $\alpha(t)$ sea el primer punto de la segunda curva $\beta(t)$, es decir, los trozos deben definirse de tal modo que las curvas resultantes sean contínuas.

Sean dos curvas polinómicas de Bèzier:

$$\alpha:[0,1]\Rightarrow \Re^3 \parallel \alpha(0)=P_0\ ;\ \alpha(1)=P_n$$
$$\beta:[0,1]\Rightarrow \Re^3 \parallel \beta(0)=Q_0\ ;\ \beta(1)=Q_n$$

La curva resultante se define como:

$$\alpha \circ \beta : [0,2] \Rightarrow \Re^3$$

$$t \Rightarrow \begin{cases} \alpha(t) & 0 \le t \le 1 \\ \beta(t-1) & 1 \le t \le 2 \end{cases}$$

que estará bien definida siempre que $P_n = Q_0$

Curvas de bezier mediante el algoritmo de Casteljau

La idea de Casteljau es aplicar el método de interpolación lineal recursivamente para definir la curva paramétrica. Dados dos puntos cualesquiera P_0 ,P_1 se considera la curva:

$$\alpha = [0,1] \to R^3$$

Supongamos que tenemos los siguientes puntos P_0 ,P_1 ,P_2 ,P_3,.....,P_n se definiría la curva del siguiente modo:

con $t \in [0,1]$:

$$\left.\begin{array}{l} P_0^0(t) = P_0 \\ P_1^0(t) = P_1 \\ \cdots\cdots\cdots \\ \cdots\cdots\cdots \\ P_n^0(t) = P_n \end{array}\right\} \Leftrightarrow \left\{\begin{array}{l} P_0^1(t) = (1-t)\,P_0^0(t) + tP_1^0(t) \\ P_1^1(t) = (1-t)P_1^0(t) + tP_2^0(t) \\ \cdots\cdots\cdots\cdots\cdots\cdots \\ \cdots\cdots\cdots\cdots\cdots\cdots \\ P_{n-1}^1(t) = (1-t)P_{n-1}^0(t) + tP_n^0(t) \end{array}\right\} \Leftrightarrow \left\{\begin{array}{l} P_0^2(t) = (1-t)P_0^1(t) + tP_1^1(t) \\ P_1^2(t) = (1-t)P_1^1(t) + tP_2^1(t) \\ \cdots\cdots\cdots\cdots\cdots\cdots \\ \cdots\cdots\cdots\cdots\cdots\cdots \\ P_{n-2}^2(t) = (1-t)P_{n-2}^1(t) + tP_{n-1}^1(t) \end{array}\right\}$$

Así sucesivamente hasta llegar:

$$P_0^n(t) = (1-t)\,P_0^{n-1}(t) + t\,P_1^{n-1}(t)$$

Este útimo polimonio se representa por $\alpha(t)$ y se denomina "CURVA DE BEZIER" asociada a los "PUNTOS DE CONTROL" { P_0 ,P_1 ,P_2 ,P_3,.....,P_n } . Matemáticamente tenemos.

$$P(t) = \left\{\begin{array}{l} P_i^0(t) = P_i \\ P_i^r(t) = (1-t)\,P_i^{r-1}(t) + t\,P_{i+1}^{r-1}(t) \end{array}\right\} \left\{\begin{array}{l} r = 1,..,n \\ i = 0,...,n-r \end{array}\right\}$$

Finalmente Indicaremos que para pasar de curvas a superficies de Bezier es necesario añadir un segundo parámetro a la descripción de la curva, el polígono de control estará formado ahora por una matriz bidimensional de puntos. La forma de la ecuación es la siguiente:

$$S_{bezier}(u,v) = \sum_{i=0}^{n}\sum_{j=0}^{m} B_{i,n}(u)B_{j,m}(v)P_{ij} \qquad 0 \le u \le 1$$

Curvas y Superficies B-Splines.

Se trata de una generalización de las curvas de aproximación definidas a trozos, en este caso se han buscado unos polinomios base que permitan un control local todavía mas adecuado que en el caso de los polinomios de Bezier, para separar y agregar este control local se añaden una serie de puntos de ruptura en el parámetro base el cual es dividido en un conjunto de intervalos. Este conjunto de intervalos recibe el nombre de vector de nodos o knots.

La expresión que define a una B-Spline es la siguiente:

$$C(u)=\sum_{i=0}^{n} N_{i,d}(u)P_i \qquad u_{\min}<u<u_{\max}, 2\le d\le n+1$$

con

$$N_{i,1}(u)=\begin{cases}1 si u_i\le u\le u_{i+1}\\ 0 en otro caso\end{cases} \qquad N_{i,d}(u)=\frac{u-u_i}{u_{i+d-1}-u_i}N_{i,d-1}(u)+\frac{u_{i+d}-u}{u_{i+d}-u_{i+1}}N_{i+1,d-1}(u)$$

El termino d que aparece en la expresión de la B-Spline hace referencia al grado de los polinomios base *N* que vamos a utilizar, mientras que P_i son los puntos de control. Como vemos una primera diferencia importante con las Bezier es que el grado del polinomio base no depende del numero de puntos de control y que independientemente de ese numero podemos seleccionarlo en cada caso.
Otro aspecto interesante es la definición de los polinomios base *N*, los cuales se definen de forma recursiva y como hemos comentado se evalúan en diferentes intervalos del parámetro *u*, los intervalos vienen definidos por el vector de knots. El numero de elementos del vector de knots por la expresión recursiva debe ser del número de puntos mas el *d* seleccionado.
La selección del vector de knots es importante ya que con él se gestiona en gran medida el control local de la curva. En función de la distribución del vector de knots se tienen distintas clasificaciones de ls B-Splines:

- Uniformes: el vector de knots se distribuye uniformemente entre el mínimo y el máximo.
- No Uniformes: Esta distribució no es uniforme,etc.

Un ejemplo de la influencia del vector de knots puede verse en el hecho de que por ejemplo si queremos forzar a que la curva pase por los puntos extremos del polígono de control debemos repetir el valor inicial y final de los knots el mismo numero de veces que el grado de los polinomios base, esto es, *d* veces. Por ejemplo un vector de knots tipico para una B-Spline con cinco puntos de control, grado 3 y que pase por los puntos extremos podría ser: {0,0,0,1,2,3,3,3}. Como vemos tiene 5+d valores y se repiten 3 veces los de los extremos.

B-Splines Racionales.
Se trata de una variante de las B-Splines donde las curvas se definen a partir de un cociente o razón entre una dos expresiones basada en los polinomios base, además se añada asociado a cada punto de control una cantidad o peso que representa el poder de atracción de ese punto sobre la curva. Esto permite mejorar el grado de control local de la curva. La expresión de las Bsplines racionales es:

$$C(u)=\frac{\sum_{i=0}^{n} w_i N_{i,d}(u)P_i}{\sum_{i=0}^{n} w_i N_{i,d}(u)_i}$$

En esta expresión w_i representan los pesos asociadas a cada punto de control el resto de términos tiene el mismo significado que en el caso de la Bsplines.
Dentro de las Bsplines Racionales también podemos tener una distribución uniforme o no uniforme del vector de knots. Cuando la distribución es no uniforme estamos dentro de un conjunto muy conocido de curvas y superficies se trata de las NURBS (Non Uniform Rational B-Splines).
Las principales ventajas de las B-Splines Racionales sobre las normales es que permiten representar de forma exacta a las formas cuadricas, esto permite utilizar en los paquetes de modelado un único tipo de primitiva de modelado de superficie.
Como sucedía en el caso de las curvas de Bezier pasara **superficies B-Spline** requiere generalizar las expresiones anteriores añadiendo un segundo parámetro.

$$S_{BSpline}(u,v)=\sum_{i=0}^{n}\sum_{j=0}^{m} N_{j,l}(v)N_{i,d}(u)P_{ij}$$

Visualización de curvas y superficies paramétricas.

Para realizar la **visualización** de estas superficies primero habrá que evaluar los polinomios que aparecen en su expresión analítica. Ya hemos visto que una forma de dibujar la superficie sería ir dando valores a los parámetros u y v. Otras formas de optimizar estos cálculos se basan en el calculo de

incrementos a partir de un primer punto **cálculo de incrementos diferenciales**[8] o en la realización de sucesivas subdivisiones de forma adaptativa **métodos de subdivisión**[9].

Curvas y Superficies Parámetricas en OpenGL

Para trabajar con curvas y superficies en OpenGL podemos utilizar una descomposición de la curva o superficie en polígonos y representarlos siguiendo los métodos vistos en puntos anteriores, la otra forma es utilizar las primitivas de este tipo que posee la OpenGL.

Curvas y Superficies de Bezier en OpenGL:

En este caso tenemos un conjunto de funciones que nos permiten generar los valores de la curva de Bezier.

La primera funcion necesaria es **glMap** se trata de una funcion que define evaluaciones paramétricas de una dimension (curvas) o dos dimensiones (superficies). Los parámetros e la función harán referencia al tipo de vértice a generar, valores de los extremos de los parámetros, numero de puntos de control, valores y valores de los puntos de control**.**

Por ejemplo:

```
glMap1f(GL_MAP1_VERTEX_3, 0.0,1.0, 3, 10, puntos_control);
```

Para que esta función se habilite es necesario emplear el glEnable(GL_MAP1_VERTEX_3).

Además de hacer esta habilitación y el mapeo anterior ahora será necesario añadir las función de recogida de las salidas y de dibuja. El siguiente ejemplo dibuja una curva de bezier, el contorno del polígono de contol y los puntos de control

```
void dibujar_curvas_bezier()
{
int i,j;

      /** Definicion del Mapeo */
      glMap1f(GL_MAP1_VERTEX_3,0.0,1.0,3,n_pc_c_bezier,&pc_c_bezier[i]
);
      /*Habilitacion del mapeo a vertices 3D */
      glEnable(GL_MAP1_VERTEX_3);
/* Especificación del numero de evaluaciones del mapeo (100 en los
valores 0 y 1 del parámetro*/
      glMapGrid1d(100,0,1.0);
/*Dibujado de los puntos de la curva de bezier */
            glEvalMesh1(GL_LINE,0,1);

/* Dibujado de la polilinea de control*/
            glBegin(GL_LINE_STRIP);
            for(j=0;j<n_pc_c_bezier;j++)
                  glVertex3fv(pc_c_bezier[j]);
            glEnd();
/* Dibujado de los puntos de control */
            glPointSize(5.0);
            glBegin(GL_POINTS);
            for(j=0;j<n_pc_c_bezier;j++)
                  glVertex3fv(pc_c_bezier[j]);
            glEnd();


}
```

Para el caso de las superficies de Bezier es necesario emplear la funcion glMap2f y evlaur la mesh en dos dimensiones glEvalMesh2. En el siguiente ejemplo se dibuja la superficie de bezier y el polinomio de control y los puntos de control.

```
void dibuja_s_bezier()
{
int i,j,k;
      glDisable(GL_LIGHTING);
```

[8] Para ver como funciona este método ver el anexo 5

[9] Este método se explica en el anexo 6

```
            glLineWidth(1.0);
        glMap2f(GL_MAP2_VERTEX_3,0,25,3,n_pc_s_bezier[0],0,25,30,
      n_pc_s_bezier[1],&pc_s_bezier[i]);
            glEnable(GL_MAP2_VERTEX_3);

    //      glEnable(GL_AUTO_NORMAL);
      //    glShadeModel(GL_SMOOTH);
/** Evaluacion de la Mesh con 25 divisiones en cada parametro de 0 a
25 */
            glMapGrid2f(25,0,25,25,0,25);
/*Dibujado de la superficie de bezier */
            glEvalMesh2(GL_LINE,0,25,0,25);

/*Dibujado del poligono de control y de los puntos de control*/
            glLineWidth(2.0);
            glColor3f(0,1,0);
            for(j=0;j<n_pc_s_bezier[0];j++)
            {
                  glBegin(GL_LINE_STRIP);
                  for(k=0;k<n_pc_s_bezier[1];k++)
                        glVertex3fv(pc_s_bezier[j][k]);
                  glEnd();
            }
        glColor3f(0,0,1);
            for(j=0;j<n_pc_s_bezier[1];j++)
            {
                  glBegin(GL_LINE_STRIP);
                  for(k=0;k<n_pc_s_bezier[i][0];k++)
                        glVertex3fv(pc_s_bezier[k][j]);
                  glEnd();
            }
            glPointSize(5.0);
            glColor3f(1,1,0);
            glBegin(GL_POINTS);
            for(j=0;j<n_pc_s_bezier[0];j++)
                  for(k=0;k<n_pc_s_bezier[1];k++)
                        glVertex3fv(pc_s_bezier[j][k]);
            glEnd();
            }
      }
```

NURBS en OpenGL

En este caso la cosa es algo más complicada debiéndose definir un objeto de descripción de la Bspline o NURB. En el siguiente ejemplo se muestran los pasos necesarios para una superficie NURB

```
void dibujanurbs()
{
int i,j;
float ctrlptos[4][4][3];
float knots[8]={0,0,0,0,1,1,1,1};
GLUnurbsObj *pnurb=NULL; // Declaración de la variable de objeto NURB
/* Rellenado de unos puntos de control simples*/
      for(i=0;i<4;i++)
            for(j=0;j<4;j++)
            {
            ctrlptos[i][j][0]=i;
                  ctrlptos[i][j][1]=j;
             ctrlptos[i][j][2]=0;
            }
    for(i=0;i<4;i++)
            ctrlptos[2][i][2]=1;
    glDisable(GL_LIGHTING);
/* Creacion del Objeto NURB */
      pnurb=gluNewNurbsRenderer();
      glEnable(GL_AUTO_NORMAL);
      //glShadeModel(GL_SMOOTH);
```

```
/* Cambio de las propiedaddes del objeto, en este caso la toleracia de
error se pone en 20 pixel, esto implica mas o menos poligonización en
la visualización*/
      gluNurbsProperty(pnurb,GLU_SAMPLING_TOLERANCE, 20.0);
/* Establecemos que quermos ver la NURB en modo alambre*/
      gluNurbsProperty(pnurb, GLU_DISPLAY_MODE,GLU_OUTLINE_POLYGON);
/* Iniciamos el dibujado de la superficie NURB*/
    gluBeginSurface(pnurb);

/*Damos los valores de descripcion de la NURB, en este caso 8 knots y
grado de los polinomios 4*/
      gluNurbsSurface(pnurb,8,knots,8,knots,4*3,3,&ctrlptos,4,4,GL_MAP
2_VERTEX_3);

      gluEndSurface(pnurb);

/* Eliminamos el objeto nurb que creamos*/
      gluDeleteNurbsRenderer(pnurb);

}
```

1.2. MODELO DE SÓLIDOS

Deberemos representar nuestros objetos con un modelo de sólido cuando:

- No sólo nos interesa la apariencia externa del objeto sino también discernir entre el espacio ocupado por el objeto y el exterior.
- Debemos guardar información referente a alguna magnitud que varía en el interior de objeto para posteriormente representarla gráficamente. Por ejemplo, esto sucede con los datos médicos obtenidos con algun sistema de scanner tridimensional, como el TAC.
- Debemos efectuar determinadas operaciones (por ejemplo la unión o la intersección) entre objetos o aplicar ciertos algoritmos que resultan más eficientes representándolos como sólidos.

Existen dos tipos básicos de representaciones:

- Representaciones de frontera (*boundary rep* o *b-rep*). Se describe la frontera del objeto utilizando un modelo de superficie con información topológica, tal como vimos en el apartado anterior , teniendo en cuenta que esta superficie deberá siempre ser cerrada.
- Representaciones sólidas o de volumen. No se emplea la superficie sino que se efectúa una descomposición del espacio mismo, indicando qué partes caen dentro o fuera del objeto. Estos métodos permiten además asignar propiedades no solo a la superficie (por ejemplo, cómo responde a la iluminación), sino también al interior del objeto. Son los que vamos a estudiar en este apartado.

Al igual que en las superficies formadas por polígonos podían plantear situaciones no regulares, también las representaciones de volumen pueden dar lugar a singularidades (p. ej. objetos de volumen nulo) o incoherencias (p. ej. autointersecciones en las que una misma parte del objeto se representa dos veces).

Sea cual sea el modelo que elijamos para representar los sólidos, éste debe permitirnos realizar las llamadas *operaciones booleanas* (unión, intersección, resta o negación). La resta y la negación son interdefinibles, es decir, que podemos definir una a partir de la otra del siguiente modo: $A - B = A \cap (\neg B)$.

Estas operaciones deben definirse de forma que sean regulares: su resultado debe ser siempre un sólido real (deben tratarse con cuidado los casos en que la intersección es sólo un plano, recta o punto, o

cuando el resultado puede ser nulo, por ejemplo al restar un objeto a sí mismo). Estas operaciones son muy útiles para el modelado geométrico 3D, ya que podemos crear nuevos objetos añadiendo o quitando partes sólidas manteniendo la unidad del objeto como entidad.

Debemos tener siempre presente que cada método de representación tiene ciertas ventajas e inconvenientes, por lo que se suelen combinar en las aplicaciones prácticas. Por ejemplo, se puede usar uno durante la fase de modelado, otro para una simulación física y otro para la visualización realista, efectuándose las conversiones apropiadas.

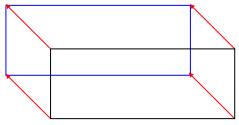
Los objetos de partida pueden generarse por diferentes métodos, sea cual sea el método de representación interna elegido. Los más sencillos son la instanciación de primitivas, es decir, de objetos predefinidos, y la generación por desplazamiento de superficies o sólidos. El barrido o desplazamiento puede ser utilizado también como una representación de ciertos objetos. Por ejemplo, cuando trasladamos una figura plana a lo largo de un eje definimos una forma tridimensional por extrusión (ver figura 1.2.1.) y cuando giramos una superficie o línea en el espacio alrededor de un eje definimos un sólido de revolución.

1.2.1. MODELOS DE BARRIDO

- **Tipos**

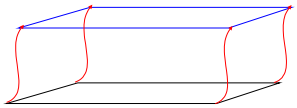
a) Extrusión de una superficie (genera un volumen)

Ex. paralela recta

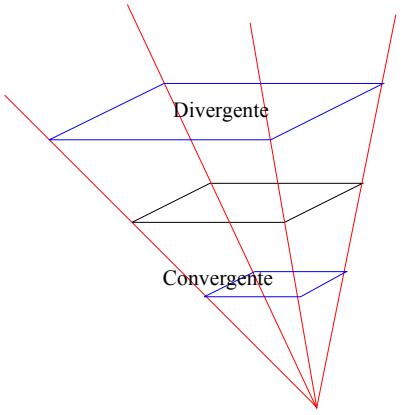


Paralelepípedo

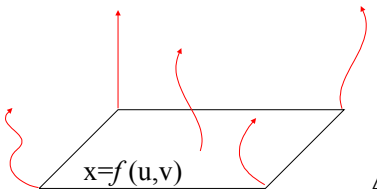
Ex. paralela curva



Ex. de proyección: convergente o divergente



Ex. Generalizada



$\Delta x = f'(u, v)$

b) Extrusión de una línea

Existen los mismos tipos que en el caso anterior.

La línea puede ser abierta o cerrada dependiendo de lo cual se generará una superficie abierta o cerrada (que envuelve un volumen).

c) Rotación de una línea sobre su eje

Genera una superficie o un sólido de revolución dependiendo de si la línea se cierra sobre su eje (ver figura 1.2.1.)

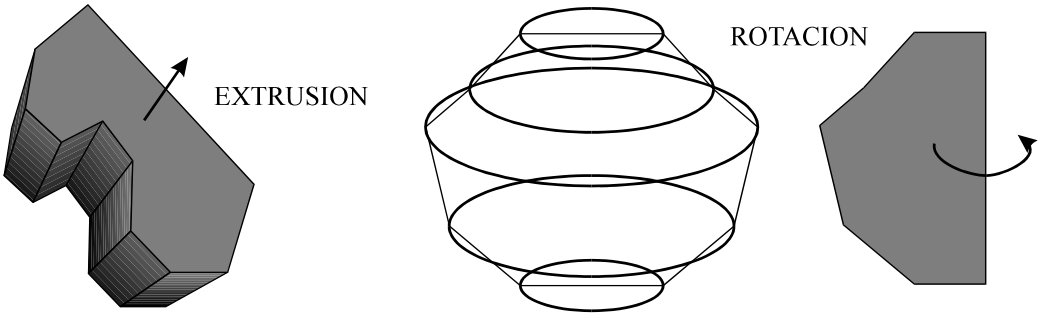


Figura 1.2.1.: Generación de un objeto por extrusión y por rotación

- **Visualización**

El mecanismo de barrido se suele emplear para construir los objetos, pero a la hora de visualizarlos lo habitual es pasar a otro tipo de representación (paramétrica o poligonal). Sin embargo, también es posible hacer un trazado de rayos con bastante facilidad.

1.1.2.2. Diseño de Sólidos. Geometría Sólida Constructiva (*Constructive Solid Geometry : CSG*)

Este es uno de los paradigmas dominantes en el modelado de sólidos. Se trata de instanciar objetos primitivos y combinarlos mediante operaciones booleanas (unión, intersección, resta, ...). La representación de un objeto complejo es un árbol en el que figuran las operaciones y objetos primitivos que definen la forma del objeto (ver figura 1.2.2.). De esta forma tan sencilla se representa un objeto en la memoria o en una base de datos.

En el árbol CSG se incluyen distintos tipos de nodos :

- Nodos de primitivas : serán las hojas del árbol.
- Nodos de operaciones : unión, intersección y diferencia (o negación).
- Nodos de transformación de coordenadas (pueden incluirse en los nodos de primitivas).

La cuestión siguiente es cómo visualizarlo. Existen algoritmos que sirven para visualizar directamente el objeto a partir del árbol CSG. mediante trazado de rayos[10] o procesamiento de tipo ráster. Otros métodos se basan en el paso de la representación CSG a otro tipo (superficie o partición espacial) que facilite el uso de algoritmos de visualización.

[10] Ver Tema 4: Modelos de Iluminación Global. Trazado de rayos

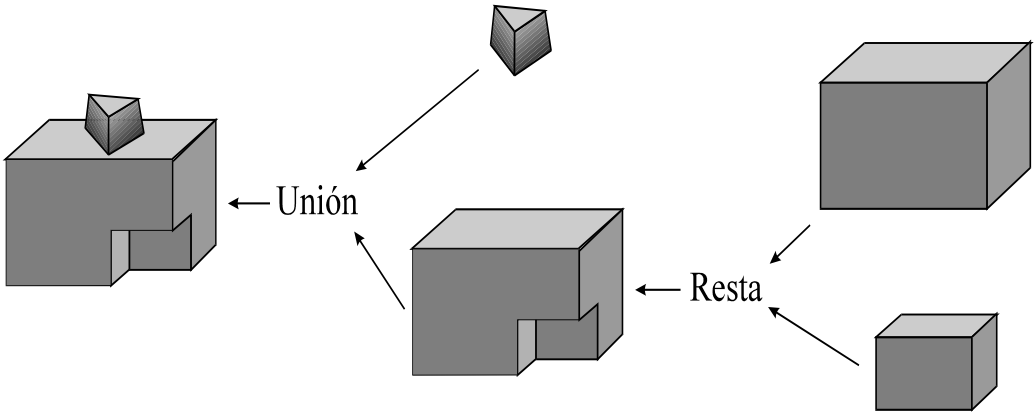


Figura 1.2.2.: Construcción de un sólido mediante un árbol CSG

Como otros métodos de representación del volumen, el sistema CSG resulta también adecuado para añadir a los objetos propiedades físicas, ya que se está representando directamente su volumen, y permite realizar la comprobación de pertenencia al objeto de los puntos en el espacio, como vamos a ver ahora.

El modelo CSG resulta muy adecuado cuando los objetos a representar se forman por combinación de formas básicas sencillas (paralelepípedos, esferas, cilindros, etc.), pero puede resultar muy ineficiente para otro tipo de objetos.

C.S.G. como método de partición espacial

Los métodos de partición espacial distinguen entre aquellas regiones del espacio que contienen puntos del objeto y aquellas que no lo contienen. La evaluación de puntos resulta muy sencilla en un árbol CSG por medio de un proceso recursivo.

El algoritmo siguiente muestra una posible implementación para este algoritmo recursivo suponiendo que la estructura NODO tiene una serie de campos específicos como son, *nodo_tipo* que indica el tipo de operación booleana , si es un nodo de operación, o el tipo de objeto en caso de que se trate de un nodo de primitivas. Además contiene dos campos adicionales entendidos como dos punteros a cada uno de sus posibles hijos, que serán de nuevo una estructura nodo.

```
esta_dentro (nodo, punto)
{
      switch (nodo_tipo) {
            caso UNION :     esta_dentro ← esta_dentro (nodo.hijo1) OR esta_dentro(nodo.hijo2)
            caso INTERSEC :esta_dentro ← esta_dentro(nodo.hijo1) AND esta_dentro(nodo.hijo2)
            caso RESTA:    esta_dentro ← esta_dentro (nodo.hijo1) AND NO esta_dentro(nodo.hijo2)
              .
              .
            caso CUBO :       esta_dentro ←  comprobar_cubo(punto)
              .
              .
            }
}
```

1.2.3. MODELOS DE PARTICIÓN ESPACIAL

Aunque la geometría sólida constructiva induce una división del espacio, no puede hablarse propiamente de una partición, ya que por definición ésta estaría formada por elementos que no tienen intersección entre sí.

Definición de partición espacial: Conjunto de volúmenes { Vi } , que permite caracterizar qué parte del espacio está dentro del objeto y qué parte está fuera. Debe cumplir:

$$\forall_i \quad V_i \neq \varnothing$$
$$\forall_{i,j} \quad V_i \cap V_j = \varnothing \qquad \bigcup_i \{V_i\} = \text{Espacio } (R^3)$$

Algunas de las utilidades del empleo de este método de representación son:

- Clasificación de puntos : para discernir cuáles son los puntos del espacio ocupados por el objeto y cuáles no
$$\{V_i\} : V_k \in \text{Objeto} \ , \ V_j \notin \text{Objeto}$$
- Información sobre propiedades de puntos en el espacio. Cada volumen de la partición puede además contener información adicional sobre alguna propiedad del objeto en esa parte del espacio (densidad, color, presión, composición…).
- Clasificación de objetos dentro de la escena : en este caso no se trataría de describir la forma de un objeto determinado, sino de agrupar los objetos de una escena compleja en partes para optimizar los métodos de visualización[11].

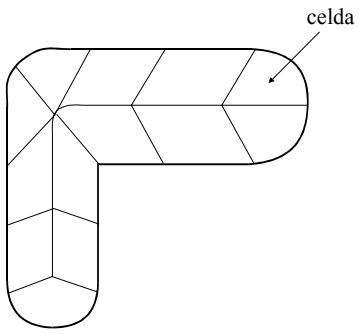
Las particiones pueden dividirse en *no jerárquicas*, cuando existe un único conjunto de volúmenes que forman la partición, y no tiene relaciones internas de inclusión, o bien particiones jerárquica y que consistirán en sucesivas particiones de más y más detalle, cuyos volúmenes están incluidos en una partición de nivel inferior, y que se forma mediante operaciones sucesivas de división.

1.2.3.1. Particiones No Jerárquicas

En este caso la partición no se forma por operaciones sucesivas de división, sino que se da directamente todo el conjunto de volúmenes en los que se ha dividido el espacio de partida. No existen relaciones de inclusión, entre los volúmenes .

- **Descomposición en celdas**

En este caso los volúmenes de la partición son celdas contiguas delimitadas por caras que pueden ser planas o curvas, y que pueden tener asociados datos escalares o vectoriales sobre la parte del objeto que representan. Las diferentes celdas se encuentran pegadas entre sí sin intersectarse, y pueden tener diferente forma y tamaño. Frecuentemente esta representación se usa para efectuar el llamado *análisis por elementos finitos* (*FEA: Finite Element Analysis*), que simula efectos físicos en un objeto sólido o fluido (cambios de temperatura, presión, velocidad de flujo, etc.) utilizando una descomposición en celdas que interactúan entre sí.

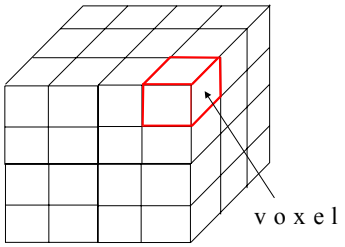


Aunque se trata de una estructura de datos sencilla, tiene el inconveniente de requerir gran volumen de almacenamiento.

[11] Ver Tema 3: Simulación en Tiempo Real y el Tema 4: Modelos de Iluminación Global.

- **Enumeración de la ocupación espacial: voxels**

Es un caso especial de descomposición por celdas en el que éstas son todas idénticas y se organizan en un entramado regular o rejilla. Es frecuente que las celdas sean pequeños cubos alineados en una trama ortogonal en cada uno de los ejes de coordenadas. Estas celdas idénticas se llaman voxels (el equivalente de un pixel en tres dimensiones). Se trata de una representación muy utilizada para almacenar datos de dispositivos tipo escáner. El problema es la cantidad de espacio que se requiere para almacenar un objeto complejo y el coste de su representación visual.



1.2.3.2. Particiones Jerárquicas

Este tipo de particiones se caracteriza por irse realizando por niveles sucesivos. La estructura de datos que se emplea es un árbol, lo que disminuye el espacio de almacenamiento requerido y hace posible definir a la vez varios niveles de detalle. Podremos acceder a la información o efectuar la visualización recorriendo la estructura solamente hasta el nivel que nos interese.

- **Particiones Jerárquicas no ortogonales: árboles binarios**

Se puede efectuar una división sucesiva del espacio en semiespacios utilizando planos de orientación arbitraria. Estos planos se organizan en un *árbol de particiones espaciales binarias o BSP-tree* (*Binary Space Partition tree*), en el que cada nodo representa un plano, y sus dos hijos son los dos semiespacios que define. En los nodos terminales del árbol del árbol se nos indica si cada región definida por los sucesivos cortes cae dentro o fuera del objeto (ver un ejemplo en 2D en la figura 1.2.3.2).

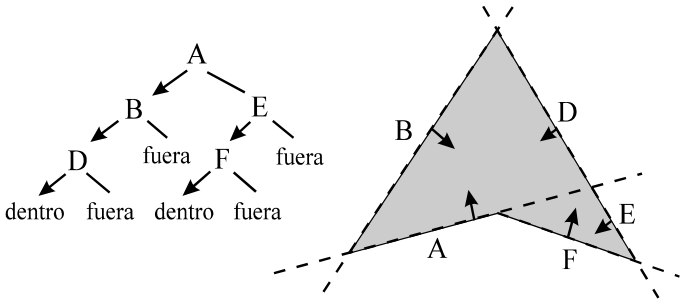


Figura 1.2.3.2: Descomposición de una figura bidimensional en un BSP-tree

Utilización de un BSP_trees para clasificar puntos

Como los demás métodos de partición espacial, estos árboles binarios nos permiten efectuar una clasificación de la pertenencia de cualquier punto del espacio al objeto definido. Para ello hay que comenzar por el plano que está en el nodo raíz del árbol. Calculando el signo de la distancia del punto al plano se sabe si hay que pasar a la rama derecha o la izquierda del árbol. Se sigue este proceso recursivamente hasta que se llega a un nodo terminal, que nos indicará si el punto está dentro o fuera.

```
esta_dentro (nodo, punto)
{
      si ( nodo = SI )              esta_dentro ← VERDAD /* Nodo terminal "dentro" = SI */
      si ( nodo = NO )esta_dentro ← FALSO   /* Nodo terminal "fuera" = NO */
      si ( comprobar_punto_plano ( punto, nodo.plano) = DENTRO )
            esta_dentro ← esta_dentro (nodo.nodo_dentro, punto)  /* Pasar a hijo izquierdo */
      sino
            esta_dentro ← esta_dentro (nodo.nodo_fuera, punto)   /* Pasar a hijo derecho */
}
```

El problema es que la representación de un objeto con un BSP-tree no tiene porqué ser única, y por lo tanto las hay más sencillas y más complejas. Por esta razón habrá que tener cuidado con el orden que se emplea al efectuar las particiones. Las mejores elecciones serán posiblemente aquellas que descarten más espacio que no pertenecen al objeto.

Además de las utilidades comunes a todos los métodos de partición, los árboles binarios han sido frecuentemente utilizados para ordenar los polígonos de una escena según la distancia al observador, y poder de esa manera aplicar el algoritmo del pintor (dibujar los polígonos de mayor a menor distancia) para calcular la ocultación de superficies en una escena 3D.

Para visualizar este tipo de objetos podemos emplear el trazado de rayos o bien convertirlos primero en polígonos.

Como puede verse fácilmente, los árboles binarios de particiones resultan especialmente eficientes para representar objetos formados por facetas planas, en los que la representación resulta, además, exacta.

- **Particiones jerárquicas ortogonales (según los ejes del espacio)**

En este caso las sucesivas particiones se realizan según planos que están orientados siguiendo los ejes ortogonales del espacio. El espacio suele, en este caso, delimitarse mediante una caja ortogonal que contiene completamente al objeto o escena. Si cada partición divide al espacio en dos semiespacios iguales, entonces estaríamos de nuevo ante un árbol binario. Si cada partición divide la caja en cuatro partes iguales de forma recursiva, entonces tenemos un *árbol de cuadrantes* o *quadtree* (ver figura 1.2.3.2.1.), que se suele usar para figuras planas. En el espacio resulta conveniente la división de cada caja en ocho cajas de tamaño mitad, formándose un *árbol de octantes* u *octree*.

Las particiones también pueden ser no homogéneas, cuando las partes resultantes no son iguales.

a) Partición jerárquica ortogonal no homogénea.

En este caso la descomposición no se realiza por mitades iguales sino desplazando los planos de corte al punto donde se considere óptimo. De esta forma se puede aproximar más rápidamente la forma del objeto, aunque la estructura de datos se complica ligeramente y resulta un poco más difícil de manejar.

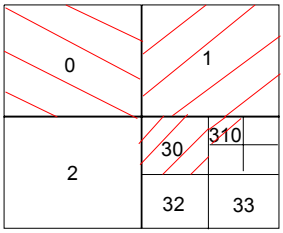
Los métodos jerárquicos ortogonales tienen la ventaja de presentar una estructura de datos muy compacta, que se presta a la utilización de algoritmos recursivos muy sencillos. Al ser jerárquicas, permiten una representación multiresolución del objeto y resultan mucho más compactas que la enumeración espacial (voxels).

Por contra, resulta difícil llegar a representar a un objeto de forma exacta, ya que éstos raramente tienen todas sus facetas orientadas ortogonalmente. La visualización de un objeto representado por un octree también es difícil, puesto que la transformación en polígonos no es inmediata, y el trazado de rayos tampoco es fácil si queremos calcular el valor del vector normal en la superficie del objeto para efectuar la iluminación.

b) Partición jerárquica ortogonal homogénea: Octrees

Si queremos describir la forma de un objeto con un octree o árbol de octantes tenemos que crear una estructura arbórea en la que la que el volumen ocupado por el objeto se va aproximando recursivamente por medio de cubos, o paralelepípedos en general. Se trata de combinar celdas ortogonales cuya longitud de arista se va dividiendo sucesivamente por 2, intentando rellenar lo más aproximadamente posible el espacio del objeto deseado (ver figura 1.2.3.2.A.). Los nodos que no caen totalmente dentro o fuera del objeto se seguirán subdividiendo de forma recursiva. En cada nodo del árbol se puede indicar únicamente si la celda correspondiente cae dentro o fuera del objeto, o se puede considerar algún tipo de gradación (parcialmente dentro, 25, 50 y 75 % dentro, etc.). La recursión deberá detenerse según algún criterio (por ejemplo, las celdas no pueden tener una arista menor de un cierto valor) se limita directamente la profundidad del árbol a un valor máximo.

Existen métodos para asociar un código a cada posible celda, de manera que exista una notación para enumerar las que componen un cierto objeto.



Objeto = 0+1+30+310

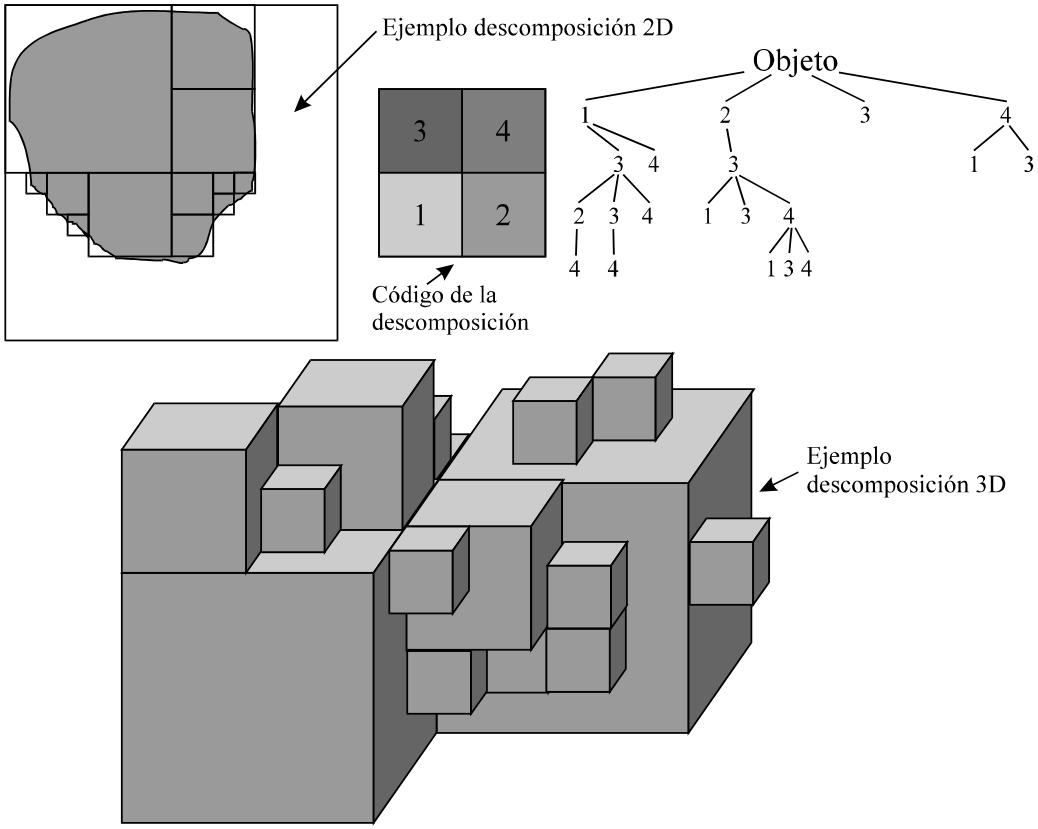


Figura 1.2.3.2.A.: Descomposición de superficies en quadtrees y sólidos en octrees

La primera llamada al función tomaría como parámetros el cubo que se va a ir dividiendo y el valor del umbral para el cual ya no se deberá seguir dividiendo. La función conectar_nodo añade a la estructura arborescente el nuevo nodo, de tal forma que ya se conoce el camino desde la raíz hasta este nodo.

```
poner_nodo (nodo_padre, real lado, real lado_minimo){
    si lado > lado_minimo{
        para cada octante o cuadrante
        si (octante ∈ objeto){
            hijo ← crear_nodo (SI)
            conectar_nodo (padre, hijo)
        }
        sino  si ( octante ∉ objeto){
            hijo ← crear_nodo (NO)
            conectar_nodo (padre, hijo)
            }
            sino{
                hijo ← crear_nodo (INDETERMINADO)
                conectar_nodo (padre, hijo)
                poner_nodo ( hijo, lado/2, lado_minimo)
            }
        }
}
```

En el caso de los octrees también resulta fácil utilizar esta estructura para clasificar puntos, comprobando recursivamente la pertenencia del punto a los nodos del árbol, comenzando por el nodo raíz.