DIBUJO EN INGENIERÍA

DIBUJO ASISTIDO POR ORDENADOR

Cátedra de Dibujo
E.U.I.T. Minera y Topográfica
Universidad de Oviedo

GEOMETRÍA Y GENERACIÓN DE LÍNEAS

PUNTOS Y RECTAS

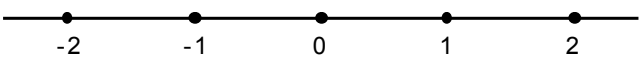
¿Qué es un punto? Para responder esto nos apoyaremos en una noción intuitiva de posición. Pensamos en un punto como una posición en el espacio. El espacio tiene un número infinito de posibles puntos. Cada una de las posiciones corresponde a un único punto.

¿Cómo es de grande un punto? Se dice que un punto es infinitesimal. Esto significa que no tiene ningún tamaño. Supongamos que tenemos un potente microscopio que puede ampliar un objeto tanto como queramos. Así, si tuviéramos que ver un objeto muy pequeño pero finito con nuestro microscopio, podríamos aumentar la ampliación hasta que fuera tan grande con quisiéramos. Sin embargo, si observamos un punto bajo el microscopio, se vería siempre con el mismo tamaño (muy pequeño). Podríamos aumentar indefinidamente la ampliación y el punto seguiría siendo un punto.

Dados dos puntos, ¿son iguales? Se dice que dos puntos son iguales si son el mismo punto. Dos puntos son dos posiciones en el espacio. Si son diferentes, entonces existe una distancia entre ellos. Esta distancia podría ser muy pequeña, pero si tuviéramos que examinar estos dos puntos con nuestro microscopio podríamos aumentar la ampliación hasta convertirla en algo apreciable. Si los dos puntos fueran iguales (es decir, ambos identifican la misma posición en el espacio), entonces no importa la ampliación que empleemos, no podremos distinguir un punto de otro. Cuando dos puntos son iguales, no son en realidad dos puntos, sino dos nombres distintos para un punto único.

¿Qué es una recta? Responderemos a esta cuestión empleando otro concepto intuitivo, e de la dirección. Dos puntos diferentes especifican una dirección (o, si lo prefiere, dos direcciones - hacia delante y hacia atrás). Consideremos ahora un tercer punto diferente de los otros dos. El primer y tercer punto también especificarán una dirección. Si la dirección definida por el primer y tercer punto es la misma que la dirección definida por el primer y segundo punto, entonces diremos que los tres puntos están situados sobre la misma recta. La recta definida por dos puntos es el conjunto de estos dos puntos y el resto de puntos que satisfacen nuestro test de dirección para pertenecer a la recta; es decir, una recta es la suma de todos los puntos que están situados sobre ella.

¿Cómo podemos especificar un punto? Consideremos primero esta cuestión en un ejemplo sencillo. ¿Cómo identificamos un punto en una recta? Supongamos que tenemos dos puntos (por ejemplo, el punto 0 y el punto 1). Estos dos puntos definen una recta. Como denominamos al punto situado a mitad de camino entre los puntos 0 y 1. Como está en el medio, parece adecuado llamarle punto ½ o 0,5. El punto simétrico al 1 respecto al 0 podría llamarse -1. El punto situado al doble de distancia respecto al 0 que el 1, en su misma dirección, sería el 2. Por lo tanto, podemos emplear la distancia desde el punto 0 para dar nombre a cualquier otro punto. Esto es en realidad lo que hace una regla. Especifica una posición como la distancia a lo largo de una recta desde un punto concreto (punto 0). Si sabemos donde están los puntos 0 y 1, entonces podemos identificar cualquier otro punto dando únicamente su distancia.

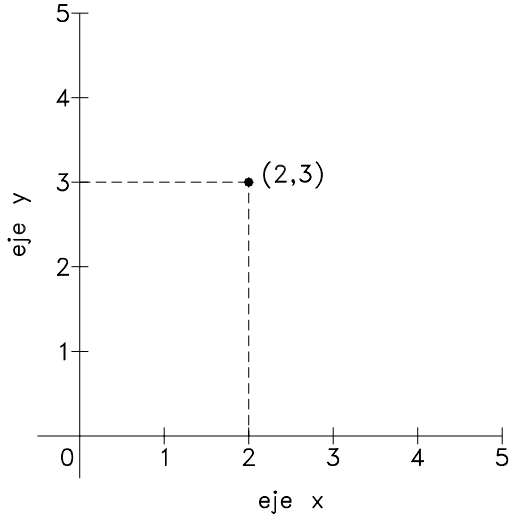


Especificación de puntos en una recta

PLANOS Y COORDENADAS

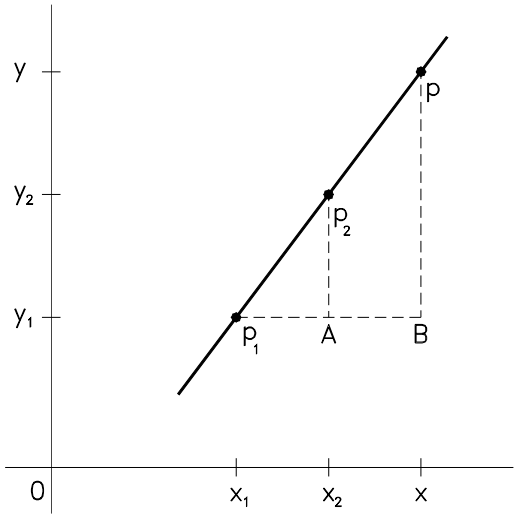
¿Qué es un plano? Las rectas son objetos que tienen una sola dimensión, su longitud. Un plano tiene dos dimensiones, largo y ancho. Podemos conceptualizar un plano como hoja de papel plana, muy delgada que se extiende indefinidamente a lo largo y a lo ancho.

¿Cómo podemos identificar un punto en un plano? Una recta es unidimensional. Sólo es necesario un número para identificar cualquier punto situado sobre ella. Pero la pantalla de una ordenador o una hoja de papel son bidimensionales. Tienen ambos largo y ancho. ¿Cómo podemos entonces identificar un punto en una superficie bidimensional? Una forma de hacerlo es emplear dos rectas diferentes de la superficie que se corten. Por convención estas rectas son perpendiculares, siendo una vertical y otra horizontal. La recta horizontal recibe el nombre de *eje* x y la recta vertical es el *eje* y. Como en el caso unidimensional, empleamos estas rectas para medir distancias. El eje x mide la distancia horizontal (a izquierda y derecha) y el eje y mide la distancia vertical (arriba y abajo). El punto donde las dos rectas se cortan es el punto 0 para las dos rectas y se llama *origen*. Con este esquema podemos identificar cualquier punto situado en una superficie bidimensional empleando dos números. El primer número (denominado *coordenada* x) mide el alejamiento hacia la derecha del punto desde el origen, sobre el eje x. El segundo número (la coordenada y) mide sobre el eje y el alejamiento en vertical del punto desde el origen. Las coordenadas se escriben como un par ordenado. Por tanto el par (2, 3) especifica un punto que está dos unidades a la derecha y 3 unidades hacia arriba desde el origen. Esta malla rectangular para identificar los puntos recibe el nombre de *Sistema Cartesiano de Coordenadas*.



Sistema Cartesiano de Coordenadas

¿Podemos especificar una recta en términos de coordenadas? La respuesta es sí. Consideremos un punto general (x, y) donde x es la coordenada x e y es la coordenada y. Podemos escribir una ecuación en x e y de tal forma que la ecuación sea válida si y sólo si (x, y) es un punto de la recta. Primero, especifiquemos una recta. Recordemos que esto requiere dos puntos diferentes. Sean $p_1=(x_1, y_1)$ y $p_2=(x_2, y_2)$.



Definición de una recta sobre un plano

Consideremos un tercer punto p=(x, y) que está situado en la recta. Podemos construir dos triángulos que nos ayuden en nuestro estudio. Extendemos una recta horizontal desde p_1. Ya que esta es una recta horizontal, todos los puntos situados en ella tienen la misma coordenada y, que será igual a la del punto p_1, es decir y_1. Si bajamos una recta vertical desde p_2 (todos los puntos de esta recta tienen a x_2 como coordenada x), se cortará con la recta horizontal en el punto A=(x_2, y_1). Una recta vertical trazada desde el punto p corta a la horizontal en el punto B=(x, y_1). Consideremos ahora los dos triángulos, el pequeño p_1p_2A y el grande p_1pB. Estos son dos triángulos semejantes y, por lo tanto, la relación entre los lados semejantes es la misma; es decir, la altura p_2A es a la altura pB como el ancho p_1A es al ancho p_1B. Pero:

Altura p_2A = y_2 - y_1
Altura pB = y - y_1
Ancho p_1A = x_2 - x_1
Ancho p_1B = x - x_1

Entonces, tenemos:

$$\frac{y_2 - y_1}{y - y_1} = \frac{x_2 - x_1}{x - x_1}$$

o de otra forma:

$$(x - x_1)(y_2 - y_1) = (y - y_1)(x_2 - x_1)$$

y esta es la ecuación de una recta que pasa por los puntos (x_1, y_1) y (x_2, y_2).

Un poco más de álgebra nos permite despejar y para tener:

$$y = y_1 + \frac{y_2 - y_1}{x_2 - x_1}(x - x_1)$$

o

$$y = mx + b$$

donde

$$m = \frac{y_2 - y_1}{x_2 - x_1}$$

y

$$b = y_1 - mx_1$$

La pendiente m es el cambio de altura dividido por la distancia en horizontal entre dos puntos de la recta. La distancia b es la altura a la que la recta corta al eje y. Esto se comprueba fácilmente observando que el punto (0, b) verifica la ecuación de la recta.

¿Podemos determinar el punto de intersección de dos rectas? Si, es bastante fácil determinar donde se cruzan dos rectas. Cuando decimos que dos rectas se cruzan queremos decir que tienen un punto en común. Este punto verifica las ecuaciones de ambas rectas. El problema reside en encontrar ese punto. Supongamos que tenemos las ecuaciones de dos rectas:

$$\text{recta 1: } y = m_1 x + b_1$$
$$\text{recta 2: } y = m_2 x + b_2$$

Si existe un punto (x_i, y_i) compartido por ambas rectas, entonces las ecuaciones:

$$y_i = m_1 x_i + b_1$$
$$y_i = m_2 x_i + b_2$$

serán ciertas. Despejando y_i tenemos:

$$m_1 x_i + b_1 = m_2 x_i + b_2$$

de donde podemos extraer el valor de x_i:

$$x_i = \frac{b_2 - b_1}{m_1 - m_2}$$

Llevando este valor a la ecuación de la recta 1 o 2 tenemos:

$$y_i = \frac{b_2 m_1 - b_1 m_2}{m_1 - m_2}$$

Por lo tanto, el punto de intersección tendrá por coordenadas:

$$\left(\frac{b_2 - b_1}{m_1 - m_2}, \frac{b_2 m_1 - b_1 m_2}{m_1 - m_2} \right)$$

Es importante señalar que dos rectas paralelas tendrán la misma pendiente. Ya que tales rectas no se intersectan, no es extraño comprobar en la expresión anterior que los denominadores se anulan, impidiendo la solución del problema.

SEGMENTOS

¿Qué es un segmento? Nuestras ecuaciones de la recta especifican todos los puntos en una determinada dirección. Las rectas se prolongan indefinidamente en un sentido y otro. Esto no es exactamente lo que nosotros necesitamos para realizar gráficos. Nos gustaría representar únicamente algunas porciones de la recta. Consideremos sólo aquellos puntos de la recta comprendidos entre dos puntos extremos p_1 y p_2.

Este conjunto de puntos recibe el nombre de segmento. Un segmento puede definirse por sus dos extremos. A partir de estos dos puntos podemos determinar la ecuación de la recta. A partir de esta ecuación y de los propios extremos podemos decidir si un punto pertenece o no al segmento. Si los extremos son p_1=(x_1, y_1) y p_2=(x_2, y_2) y estos nos dan la ecuación de la recta $y = mx + b$, entonces otro punto p_3=(x_3, y_3) pertenece al segmento si:

1) $y_3 = mx_3 + b$

2) $\min(x_1, x_2) \le x_3 \le \max(x_1, x_2)$

3) $\min(y_1, y_2) \le y_3 \le \max(y_1, y_2)$

¿Qué longitud tiene un segmento? Si conocemos los extremos p_1 y p_2 de un segmento, podemos determinar su longitud L. Construimos un triángulo rectángulo p_1p_2A trazando una recta vertical por p_2 y una recta horizontal por p_1 que se cortan en A. Si aplicamos ahora el teorema de Pitágoras tenemos:

$$L = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

¿Cuál es el punto medio de un segmento? El punto medio de un segmento es frecuentemente de utilidad y fácil de calcular. Este punto tendrá una coordenada x a medio camino entre las coordenadas x de los extremos del segmento, y una coordenada y a medio camino entre las coordenadas y de los extremos del segmento. Por lo tanto el punto medio será:

$$(x_m, y_m) = \left(\frac{x_1 + x_2}{2}, \frac{y_1 + y_2}{2} \right)$$

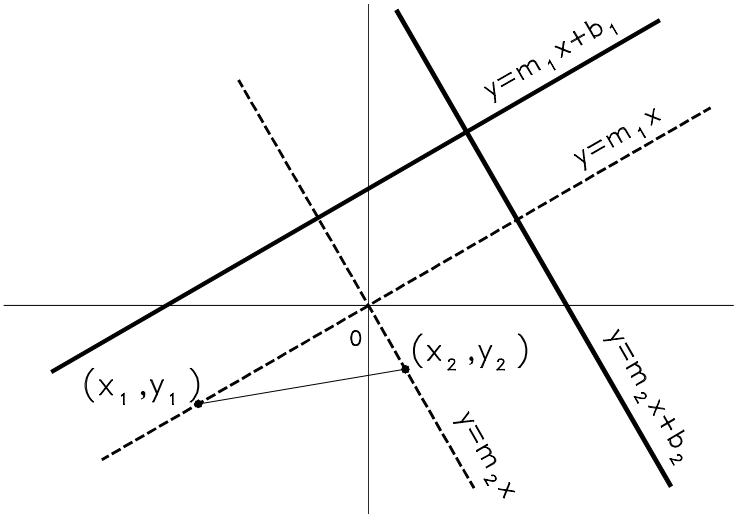
RECTAS PERPENDICULARES

¿Podemos saber cuando dos rectas son perpendiculares? Esto se consigue estudiando la pendiente de ambas rectas. Supongamos que tenemos dos rectas de ecuaciones:

recta 1: $y = m_1 x + b_1$

recta 2: $y = m_2 x + b_2$

Si la primera es perpendicular a la segunda, entonces una recta paralela a la primera (es decir, con la misma pendiente que aquella) también será perpendicular a la segunda. Por ejemplo, $y = m_1 x$ debería ser perpendicular a $y = m_2 x + b_2$. El mismo razonamiento se puede aplicar a la segunda recta: $y = m_2 x$ será perpendicular a $y = m_1 x$. Estas dos rectas se cortan en el origen.



Construcción para comprobar si dos rectas son perpendiculares

Tomemos ahora un punto (x_1, y_1) de $y = m_1 x$, que verifica $y_1 = m_1 x_1$; y un punto (x_2, y_2) de $y = m_2 x$, que cumple $y_2 = m_2 x_2$. Los puntos (x_1, y_1), (x_2, y_2) y el

origen forman un triángulo. Si las dos rectas son perpendiculares, el triángulo será recto y se cumplirá el teorema de Pitágoras:

$$x_1^2 + y_1^2 + x_2^2 + y_2^2 = (x_1 - x_2)^2 + (y_1 - y_2)^2$$

Simplificando la expresión anterior, se obtiene:

$$x_1 x_2 + y_1 y_2 = 0 \qquad \text{o} \qquad \frac{y_1}{x_1} = -\frac{x_2}{y_2}$$

pero, como $y_1 = m_1 x_1$ e $y_2 = m_2 x_2$, tenemos:

$$m_1 = -\frac{1}{m_2}$$

Por lo tanto, dos rectas son perpendiculares, cuando la pendiente de una es la inversa de la otra cambiada de signo.

VECTORES

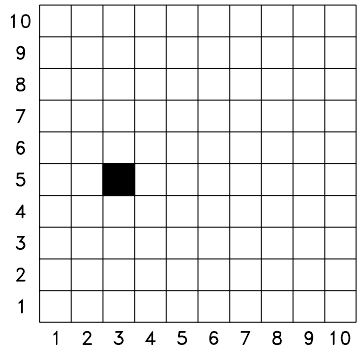
¿Qué es un vector? Un vector tiene una dirección única y un módulo. Un vector puede representarse por el par $\left[D_x, D_y\right]$ donde D_x es el desplazamiento a lo largo del eje x y D_y es el desplazamiento a lo largo del eje y.

Al contrario que los segmentos, los vectores no tienen una posición fija en el espacio. Nos indican cuanto y en que dirección nos tenemos que mover, pero no nos indican donde tenemos que empezar. La idea del vector es útil porque refleja aproximadamente el comportamiento de un lápiz que dibuja líneas sobre un papel o un rayo de electrones dibuja líneas sobre una pantalla de rayos catódicos.

PIXELS

¿Cómo se refleja todo esto en una pantalla gráfica? Para empezar, la noción matemática de un infinito número de puntos infinitesimales no es aplicable en una pantalla gráfica. No podemos representar un número infinito de puntos en un ordenador, así como tampoco podemos representar una cantidad infinita de números. Las máquinas son finitas, y estamos limitados a un número finito de puntos que componen cada una de las líneas de la pantalla, cuyo número varía desde varios cientos a varios miles. El máximo número de puntos distinguibles que una línea puede tener es una medida de la resolución de la pantalla. Cuanto mayor es el número de puntos mayor es la resolución. Esta limitación en el número de puntos no debe preocuparnos mucho, ya que el ojo humano no aprecia detalles más allá de los 1000 puntos por segmento. Ya que hemos de construir nuestras líneas a partir de un número finito de puntos, cada uno debe tener un cierto tamaño y no será por tanto un punto de verdad. Recibe el nombre de pixel (de las palabras inglesas *picture element*). Un pixel es el elemento direccionable más pequeño de la pantalla. Es la porción más pequeña que podemos controlar. Todos los pixel tienen un nombre o una dirección. Los nombres que identifican a los pixel corresponden a las coordenadas que definen a los puntos. Esto es parecido a la forma de seleccionar los elementos de una matriz mediante subíndices. Las imágenes gráficas creadas por ordenador se obtienen activando la intensidad y el color de los pixel que componen la pantalla. Se dibujan segmentos iluminando, es decir, dando brillo a un conjunto de pixel situados entre el pixel inicial y el pixel final. Podemos pensar en la pantalla como una malla o matriz de pixel. Debemos dar coordenadas enteras a cada uno de ellos. Empezando a la izquierda con 1, se numera cada una de las columnas.

Empezando abajo con 1, se numera cada una de las filas. Las coordenadas (i, j) de un pixel indican por tanto la columna y la fila en la que está situado.



Ejemplo de pixel iluminado

Se colocan los valores de intensidad de todos los pixel en una matriz en la memoria del ordenador. De esta forma la pantalla gráfica puede acceder a esta matriz para determinar la intensidad con la que debe iluminarse cada pixel. Esta matriz, que contiene una representación interna de la imagen, se denomina *frame buffer*.

GENERACIÓN DE VECTORES

El proceso de activación de los pixel para un determinado segmento se denomina *generación de un vector*. Si conocemos los extremos que definen al segmento, ¿cómo decidimos que pixel deben ser activados? Existen varios esquemas para seleccionar estos pixel. El método que presentamos aquí recibe el nombre de Analizador Diferencial Digital Simétrico (ADD). Este nombre se deriva de la utilización de un método numérico similar al empleado para la resolución de ecuaciones diferenciales. Sin embargo, la ecuación diferencial que vamos a resolver es una muy sencilla, la que corresponde a una línea recta, y en vez de imprimir la solución, la vamos a dibujar.

Empezaremos por considerar otra forma de la ecuación de una recta. La denominada forma *paramétrica* porque damos los valores de x e y en función de un parámetro u. Supongamos que queremos representar el segmento comprendido entre los puntos (x_1, y_1) y (x_2, y_2). Queremos que la coordenada x varíe uniformemente entre x_1 y x_2. Esto podría representarse mediante la ecuación:

$$x = x_1 + (x_2 - x_1)\,u$$

Cuando u vale 0, x vale x_1. A medida que u crece hasta 1, x se mueve uniformemente hacia x_1. Pero un segmento de recta, necesitamos que la coordenada y varíe también entre y_1 e y_2 de manera uniforme, al mismo tiempo que cambia x.

$$y = y_1 + (y_2 - y_1)\,u$$

Estas dos ecuaciones en conjunto describen una línea recta. Esto se demuestra despejando el parámetro u en ambas ecuaciones e igualando las expresiones que se obtienen. Estas ecuaciones sugieren un método para seleccionar los pixel que deben ser iluminados. La idea es empezar dando a u el valor 0 e ir aumentándolo en pequeñas pasos hasta alcanzar el valor 1. A medida que hacemos esto, x e y se desplazan a lo largo del segmento de recta en pequeños pasos. Este es el tipo de algoritmo que es útil para dibujar una línea. En cada paso iluminamos el pixel que contiene el punto (x, y).

GENERACIÓN DE CURVAS

Hasta ahora hemos trabajo con segmentos rectos. Ya vimos como estos segmentos son fáciles de generar. Luego aprendimos como representar y manipular estos segmentos. Las líneas rectas tienen una representación matemática sencilla, que las hace muy fáciles de manejar en operaciones como las transformaciones y el recorte. Sin embargo, el mundo real no está compuesto únicamente por figuras lineales. Muchas de las cosas que podríamos querer modelar y dibujar supone la utilización de curvas.

Existen dos planteamientos para la representación de curvas. Uno es desarrollar un algoritmo de generación de curvas como el ADD antes. Con este planteamiento se crea una curva real. El segundo método es aproximar la curva mediante un determinado número de pequeños segmentos rectos. Las técnicas de interpolación que veremos más adelante se emplean en este segundo planteamiento. Empezaremos considerando la generación de curvas reales.

Supongamos que tenemos una curva, tal como un arco de circunferencia, que queremos dibujar. Si conocemos la ecuación diferencial de esta curva, podemos escribir un algoritmo Analizador Diferencial Digital Simétrico que nos permitirá calcular las coordenadas de los puntos de la curva.

Las ecuaciones diferenciales para curvas sencillas tales como círculos y elipses son bastante fáciles de resolver, y los algoritmos para su generación se pueden implementar directamente en el hardware. Así un dispositivo gráfico puede ser capaz de dibujar tanto líneas rectas como arcos.

Veamos un ejemplo, con la creación paso a paso de un algoritmo para la creación de arcos de circunferencia. Las ecuaciones del arco se pueden escribir en función del ángulo α como sigue:

$$x = R\cos\alpha + x_0$$
$$y = R\,\mathrm{sen}\,\alpha + y_0$$

donde (x_0, y_0) es el centro del arco y R es el radio.

Si derivamos las expresiones anteriores tenemos

$$\mathrm{d}\,x = -R\,\mathrm{sen}\,\alpha\,\mathrm{d}\,A = -(y - y_0)\,\mathrm{d}\,A$$
$$\mathrm{d}\,y = R\cos\alpha\,\mathrm{d}\,A = (x - x_0)\,\mathrm{d}\,A$$

De esta forma podemos saber las cantidades que hay que añadir a las coordenadas de un punto viejo para obtener un punto nuevo.

$$x_2 = x_1 + \mathrm{d}\,x = x_1 - (y_2 - y_0)\,\mathrm{d}\,A$$
$$y_2 = y_1 + \mathrm{d}\,y = y_1 + (x_2 - x_0)\,\mathrm{d}\,A$$

Estas ecuaciones constituyen el núcleo del algoritmo de generación de arcos. El valor del incremento angular en cada paso debe ser lo suficientemente pequeño para que no haya huecos entre los pixel que definen la curva y de una buena aproximación del arco.

Existen, sin embargo, algunos problemas que surgen cuando se generan curvas a esta nivel. Para definir una curva, necesitamos más información que sólo sus extremos. También puede surgir otro problema cuando se apliquen transformaciones. Un segmento recto sigue siendo recto cuando cambia su escala, pero otros curvas podrían comportarse de forma distinta. Por ejemplo, cuando cambiamos la escala de una circunferencia sólo en una dirección se convierte en una elipse. Si únicamente tenemos un generador de arcos de

circunferencia, estaremos muy limitados a la hora de escalar nuestras imágenes. Otro problema surge de la necesidad de recortar las curvas cuando se salen del área de dibujo. Aunque todos estos inconvenientes no son desdeñables queda todavía un último problema: no todas las curvas que queramos dibujar tendrán un algoritmo de generación sencillo. Las curvas que se emplean en la representación de las alas de un avión o los alerones de los coches de carreras o las caras humanas no son simples arcos y elipses. Tendremos que aproximar tales curvas mediante trozos de arcos de circunferencia, elipses y espirales. Pero podríamos también juntar muchos segmentos rectos a los que luego sería posible aplicar las transformaciones y algoritmos que veamos aquí.

INTERPOLACIÓN

¿Cómo podemos expresar una curva que no tiene una definición matemática sencilla? Podemos dibujar una aproximación de dicha curva si tenemos un conjunto de puntos representativos. Podemos luego adivinar el comportamiento que debería tener la curva entre los puntos anteriores. Si la curva es suave y los puntos representativos están los suficientemente juntos, podemos hacer una buena aproximación de las partes que faltan de la curva. Nuestra predicción no será exacta, pero bastará para guardar las apariencias. Llenaremos las partes desconocidas de la curva con porciones de curvas conocidas que pasan por los puntos que la definen. Ya que la curva desconocida y la conocida comparten los mismo puntos localmente, suponemos que en esas zonas las dos son bastante parecidas. Ajustamos una porción de la curva que no conocemos con una curva que conocemos. Así podemos rellenar los huecos existentes entre los puntos de partida determinado las coordenadas de los puntos que pertenecen a la curva conocida o aproximante y uniendo estos puntos con segmentos de recta.

La siguiente pregunta que nos hacemos es ¿qué curva empleamos como aproximación? ¿cuál es la expresión matemática de la curva que pasa por varios puntos en una determinada región? Existen muchas funciones que se pueden hacer pasar por un determinado número de puntos mediante el ajuste de los parámetros que las caracterizan. Se han empleado las funciones trigonométricas, polinómicas, exponenciales y otros tipos. Es más, dentro de cada clase existen múltiples opciones. Nosotros utilizaremos las funciones polinómicas para realizar la aproximación de la curva.

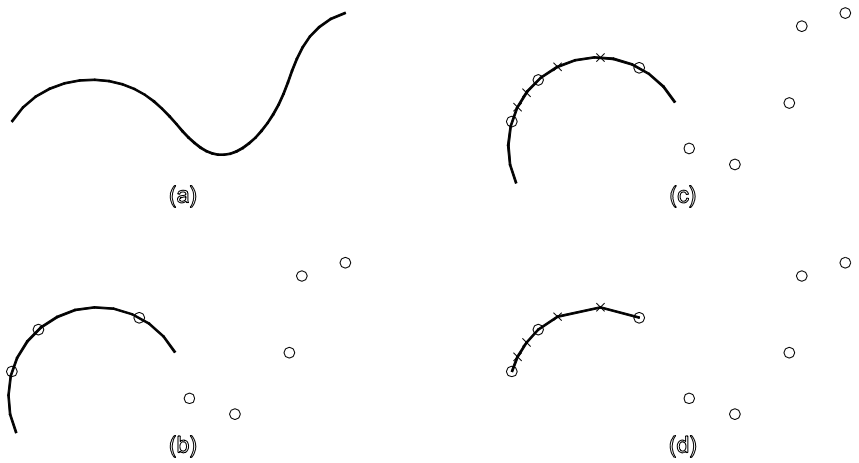


Figura 1.- Proceso de Interpolación
curva desconocida a aproximar
ajustamos una región con una curva conocida
calculamos más puntos a partir de la curva conocida
dibujamos los segmentos de recta que conecta todos los puntos

Las funciones se suelen expresar frecuentemente de la forma $y = f(x)$, pero nosotros preferimos la forma paramétrica:

$$x = fx(u)$$
$$y = fy(u)$$
$$z = fz(u)$$

Existen un par de razones por las que la forma paramétrica es preferible. Por un lado, la diferencia entre dos y tres dimensiones sólo supone añadir una tercera ecuación para z. Trata las tres direcciones de forma similar y permite valores múltiples (varios valores de y para un mismo valor de x) de tal forma que las curvas pueden volver sobre si mismas o incluso cortarse.

Inventemos ahora una función que pueda ser empleada en la interpolación. Habrá un montón de cosas mal en este primer intento, pero a pesar de ello nos servirá para comprender mejor este planteamiento.

Supongamos que necesitamos una curva polinómica que para por un determinado número n de puntos

$$(x_1, y_1, z_1), (x_2, y_2, z_2), \ldots, (x_n, y_n, z_n)$$

Construiremos la función de acuerdo a los siguientes sumatorios, un término por cada uno de los puntos de referencia.

$$fx(u) = \sum_{i=1}^{n} x_i B_i(u)$$
$$fy(u) = \sum_{i=1}^{n} y_i B_i(u)$$
$$fz(u) = \sum_{i=1}^{n} z_i B_i(u)$$

Las funciones $B_i(u)$ reciben el nombre de *funciones de curvatura*. Para cada valor del parámetro u indican cuanto afecta el punto i a la posición de la curva. Podemos imaginar que cada punto intenta atraer la curva hacia si mismo. La función $B_i(u)$ nos dice como es de fuerte esta atracción. Si para algún valor de u, $B_i(u) = 1$ y para cada $j \neq i$, $B_j(u) = 0$, entonces el punto i tiene el control completo sobre la curva. La curva pasará por el punto i. Si, para un valor diferente del parámetro, otro punto tiene el control de la curva, también pasará la curva por él. Lo que necesitamos entonces son funciones de curvatura que para diferentes valores de u den sucesivamente el control de la curva a cada uno de los puntos de referencia o *puntos de control*.

Los valores particulares del parámetro u para los que la curva pasa por los puntos de control no tienen demasiada importancia, siempre que los puntos de referencia tomen el control de la curva en el orden correcto. Debemos crear funciones de curvatura para las que el primer punto de referencia (x_1, y_1, z_1) tiene el control total cuando $u = -1$, el segundo cuando $u = 0$, el tercero cuando $u = 1$ y así sucesivamente. La razón para esta elección de los valores de u es hacer más fácil la implementación del programa. Para esto necesitamos que $B_1(u)$ valga 1 cuando $u = -1$, y 0 para $u = 0, 1, 2, \ldots, n-2$. Una expresión que es nula en los puntos correctos es:

$$u(u-1)(u-2)\ldots(u-(n-2))$$

Para $u=-1$ esta expresión vale

$$(-1)(-2)(-3)\ldots(1-n)$$

por tanto, si dividimos por esta constante tenemos un valor unitario para $u=-1$, que es precisamente el comportamiento que buscamos

$$B_1(u)=\frac{u(u-1)(u-2)\ldots(u-(n-2))}{(-1)(-2)(-3)\ldots(1-n)}$$

La función de curvatura *i*-ésima se puede construir de la misma forma para que tome el valor 1 en $u=i-2$ y sea nula para el resto de los enteros.

$$B_i(u)=\frac{(u+1)u(u-1)\ldots(u-(i-3))(u-(i-1))\ldots(u-(i-2))}{(i-1)(i-2)(i-3)\ldots(1)(-1)\ldots(i-n)}$$

Consideremos el caso donde existen cuatro puntos de control. Serán necesarias por tanto cuatro funciones de curvatura. La expresión anterior nos da las siguientes funciones:

$$B_1(u)=\frac{u(u-1)(u-2)}{(-1)(-2)(-3)}$$
$$B_2(u)=\frac{(u+1)(u-1)(u-2)}{(1)(-1)(-2)}$$
$$B_3(u)=\frac{(u+1)u(u-2)}{(2)(1)(-1)}$$
$$B_4(u)=\frac{(u+1)u(u-1)}{(3)(2)(1)}$$

Utilizando estas funciones y los cuatro puntos de control se puede construir una curva que pase por los cuatro.

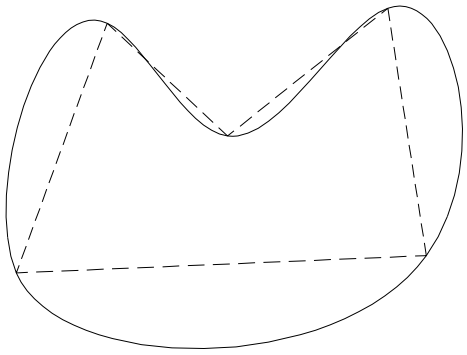
$$x=x_1B_1(u)+x_2B_2(u)+x_3B_3(u)+x_4B_4(u)$$
$$y=y_1B_1(u)+y_2B_2(u)+y_3B_3(u)+y_4B_4(u)$$
$$z=z_1B_1(u)+z_2B_2(u)+z_3B_3(u)+z_4B_4(u)$$

Ahora se espera que esta curva se acerque bastante a la curva real en la región correspondiente a estos cuatro puntos, particularmente en el medio, entre el segundo y el tercero. Esta es la zona de la curva que se obtiene para valores de u comprendidos entre 0 y 1. Podemos encontrar los puntos de curva aproximada dando valores a u dentro de ese rango. Los resultados de estos cálculos nos permiten obtener las coordenadas de los segmentos que luego podrán ser dibujados como representación de dicha curva aproximada. Por ejemplo, supongamos que deseamos aproximar la curva entre el segundo y el tercer punto de referencia mediante tres segmentos rectos. Tenemos como puntos válidos los obtenidos para $u=0$ y $u=1$. Si utilizamos nuestras ecuaciones para calcular los puntos en $u=1/3$ y $u=2/3$, tendremos cuatro puntos que nos sirven como extremos de los tres segmentos aproximados. Siguiendo este mismo proceso podemos aproximar toda la curva. Tomamos cuatro puntos de control consecutivos (1, 2, 3, 4) y aproximamos el tramo comprendido entre los dos del medio (2, 3). Damos ahora un paso más, y nos desplazamos al siguiente punto de control en un extremo y descartamos el último por el otro lado (2, 3, 4, 5). Podemos aproximar entonces la siguiente porción de la curva (3, 4). De esta forma nos movemos por todos los puntos de control hasta dibujar la curva completa. Las regiones inicial y final de la curva requieren un tratamiento

especial. Para los primeros cuatro puntos (1, 2, 3, 4) necesitaremos dibujar la región comprendida entre los puntos 1 y 2 dando valores a u entre -1 y 0. De igual forma, la función de curvatura para la última etapa del proceso debe ser evaluada para valores de u entre - y 2. No existe nada particularmente difícil en estos casos especiales, sólo debemos tenerlos en cuenta.

POLÍGONOS INTERPOLANTES

Los lados de un polígono pueden ser redondeados también mediante la utilización de las funciones de curvatura que hemos visto anteriormente. De hecho, un polígono es mucho más fácil de manejar, ya que no es necesario procesar por separado las regiones inicial y final. Lo único que hacemos es avanzar alrededor del polígono, suavizando cada uno de los lados mediante varios segmentos pequeños. Se empieza con un polígono que tiene unos pocos lados y se termina un nuevo polígono que tiene muchos más lados y aparece más redondeado.



Suavizado de un polígono

B-SPLINES

Ahora que hemos presentado un método de interpolación utilizando nuestras funciones de curvatura caseras, examinemos sus inconvenientes. Algo que debemos tener en cuenta es que la suma de las funciones de curvatura no es 1 para todos los valores de u. Las funciones de curvatura han sido diseñadas para sumar 1 cuando el parámetro tiene un valor entero, pero no en las regiones intermedias. ¿Qué significa esto? Supongamos que todos los puntos de referencia tienen la misma coordenada x, $x = x_0$. Queremos, entonces, que las curva aproximada tenga también un valor constante de x en los puntos intermedios, pero la curva aproximada es

$$x = \sum_{i=1}^{n} x_i B_i(u)$$

que para $x_i = x_0$ da

$$x = x_0 \sum_{i=1}^{n} B_i(u)$$

Sólo conseguiremos este efecto si la suma de las funciones de curvatura es 1 para todos los valores de u. Este inconveniente supone también que las curvas que deberían estar contenidas en un plano pasan de un lado a otro del mismo. Este problema podría resolverse *normalizando* las funciones, es decir, dividimos los valores obtenidos por la suma de todos ellos para cada valor de u.

Pero, este no es el único problema. Cada una de las secciones de la curva está conectada con la siguiente en un único punto de control, pero la pendiente en este punto podría no coincidir en las dos secciones. Esto significa que podrían

existir esquinas en los puntos de control y no tendríamos por tanto una curva completamente suavizada.

Por último, al obligar a la curva que pase por cada uno de los puntos de control, reducimos hasta anularlo el efecto de los restantes puntos de referencia; pero a medida que nos alejamos del punto, todos los puntos que están en la misma región adquieren un cierto control sobre la curva. De esta forma, vemos como el control de la curva por un determinado punto de referencia sube y baja a medida que varía u.

Un comportamiento más natural sería que cada punto de referencia variase su control sobre la curva desde cero, cuando estamos muy alejados de él, hasta alcanzar el máximo, cuando nos acercamos a él. Podemos conseguir esto si no obligamos a la curva a pasar por los puntos de control, sino que tiramos de ella hacia la vecindad de cada punto. El resultado será una curva que sigue los contornos generales indicados por los puntos de referencia pero que no pasa en realidad por ninguno de ellos. Un conjunto de funciones de curvatura que siguen este planteamiento y también suman siempre 1 reciben el nombre de *B splines*.

La obtención de estas funciones está más allá del objetivo de estos apuntes, pero presentaremos e implementaremos las funciones *B splines cúbicas*, que son adecuadas para la mayoría de las aplicaciones. Estas funciones interpolan a partir de cuatro puntos de referencia y son polinomios de grado 3 en u, tal como nuestras funciones caseras. Las funciones B splines cúbicas para la zona intermedia de la curva son:

$$
\begin{aligned}
B_1(u) &= (1-u)^3/6 \\
B_2(u) &= u^3/2 - u^2 + 2/3 \\
B_3(u) &= -u^3/2 + u^2/2 + u/2 + 1/6 \\
B_4(u) &= u^3/6
\end{aligned}
$$

Para empezar o terminar una curva aproximada con B splines cúbicas son necesarias dos secciones especiales. Las siguientes expresiones se deben emplear para aproximar la primera sección de la curva

$$
\begin{aligned}
B'_1(u) &= (1-u)^3 \\
B'_2(u) &= 21u^3/12 - 9u^2/2 + 3u \\
B'_3(u) &= -11u^3/12 + 3u^2/2 \\
B'_4(u) &= u^3/6
\end{aligned}
$$

La última porción de la curva necesita

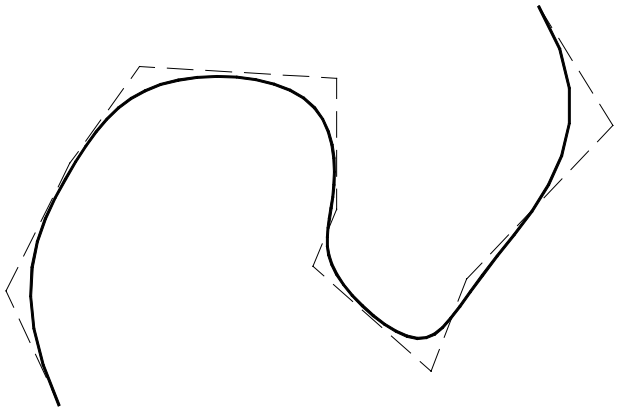
$$
\begin{aligned}
B'_{1R}(u) &= B'_4(1-u) \\
B'_{2R}(u) &= B'_3(1-u) \\
B'_{3R}(u) &= B'_2(1-u) \\
B'_{4R}(u) &= B'_1(1-u)
\end{aligned}
$$

La segunda sección de la curva también tiene sus propias funciones de curvatura

$$B_1''(u) = (1-u)^3/4$$
$$B_2''(u) = 7u^3/12 - 5u^2/4 + u/4 + 7/12$$
$$B_3''(u) = -u^3/2 + u^2/2 + u/2 + 1/6$$
$$B_4''(u) = u^3/6$$

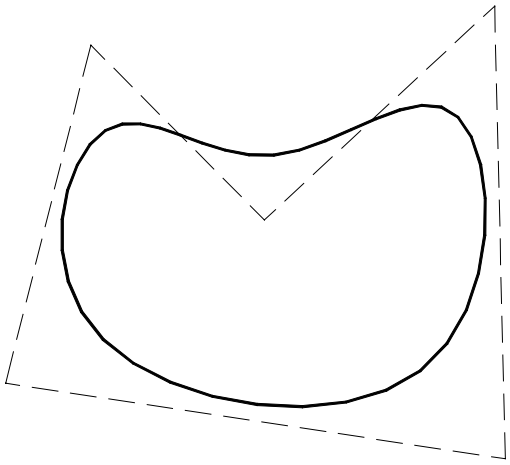
Y la penúltima sección de la curva emplea la inversa de estas funciones

$$B_{1R}''(u) = B_4''(1-u)$$
$$B_{2R}''(u) = B_3''(1-u)$$
$$B_{3R}''(u) = B_2''(1-u)$$
$$B_{4R}''(u) = B_1''(1-u)$$



Suavizado mediante *B splines*

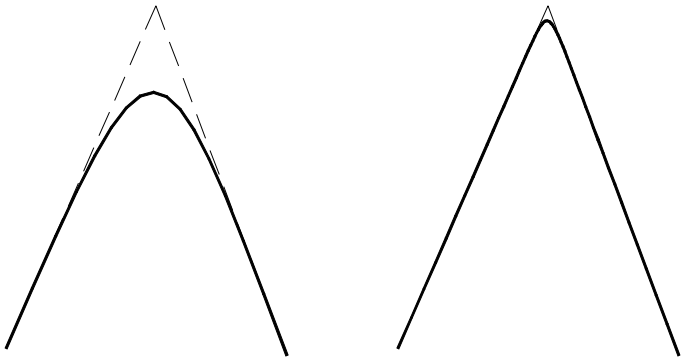
Las curvas B splines también se pueden emplear para suavizar polígonos.



Suavizado de un polígono mediante curvas de B splines cúbicas

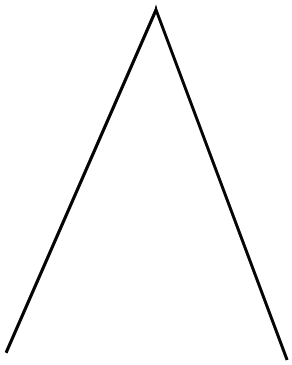
Las curvas B splines fueron diseñadas para eliminar las esquinas agudas en las curvas y, por otro lado, la curva tampoco pasa por los puntos de referencia. Sin embargo, también se pueden producir esquinas agudas y obligar a pasar la curva por los puntos de control mediante B splines, si es necesario. Esto se consigue usando varios puntos de control idénticos.

Cuando se emplean dos puntos idénticos la curva se acerca más a ese punto y la esquina se hace más aguda.



Comportamiento de las B splines con puntos repetidos

Si emplean tres puntos idénticos se obliga a la curva a pasar por ese punto



B spline obligada a pasar por un punto repetido tres veces

La aplicación de las funciones de curvatura para la interpolación y aproximación mediante pequeños segmentos rectos nos permite dibujar curvas que podrían no tener una definición matemática sencilla.

Esta técnica también nos permite emplear los sencillos métodos de transformación e intersección de rectas que veremos más adelante. El mayor inconveniente de este método es el volumen de cálculos necesario para representar incluso una curva sencilla.

APLICACIONES

LISTADO 1 - REPRESENTACIÓN DE SEGMENTOS

En el siguiente listado se presenta la rutina (BASIC) encargada de dibujar cada uno de los pixel que definen el segmento comprendido entre dos puntos. Como parámetros se definen las coordenadas de los extremos del segmento y el color con que se va a identificar en la pantalla cada pixel.

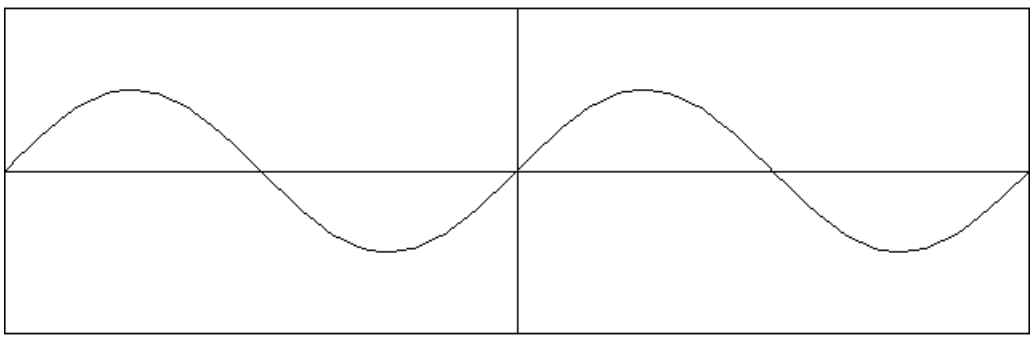
```
DECLARE FUNCTION MAX! (a!, b!)
‘ Valores de prueba
X1 = 0: Y1 = 30: X2 = 320: Y2 = 170: C = 2
' MODO 320x200
SCREEN 1
‘ Componentes del vector desplazamiento
DX = X2 - X1
DY = Y2 - Y1
‘ NUMERO DE PASOS
ST = INT(MAX(ABS(DX), ABS(DY))) + 1
‘ Tamaño del paso en cada dirección
DXS = DX / ST
DYS = DY / ST
‘ Redondeo inicial
X = X1 + .5
Y = Y1 + .5
‘ Bucle de dibujo
FOR U = 1 TO ST
   ‘ Iluminamos el pixel
   PSET (INT(X), INT(Y)), C
   ‘ Punto siguiente
   X = X + DXS
   Y = Y + DYS
NEXT U
‘ Punto final
PSET (INT(X), INT(Y)), C

FUNCTION MAX (a, b)
IF (a > b) THEN
   MAX = a
ELSE
   MAX = b
END IF
END FUNCTION
```

LISTADO 2 - REPRESENTACIÓN DE CURVAS Y=F(X)

En este otro listado se presenta un pequeño programa que permite representar curvas del tipo $y=f(x)$ definiendo los valores máximo y mínimo de las dos variables, así como el número de segmentos que se emplearán para representar la curva.

```
DECLARE FUNCTION F! (X!)
CONST PI = 3.141592654#

XMIN = -2 * PI: XMAX = 2 * PI
YMIN = -2: YMAX = 2
XS = XMAX - XMIN
YS = YMAX - YMIN
XC = .5 * (XMIN + XMAX)
YC = .5 * (YMIN + YMAX)

IF (XS / 4 > YS / 3) THEN
   SXMIN = XMIN
   SXMAX = XMAX
   SYMIN = YC - .5 * XS * 3 / 4
   SYMAX = YC + .5 * XS * 3 / 4
ELSE
   SXMIN = XC - .5 * YS * 4 / 3
   SXMAX = XC + .5 * YS * 4 / 3
   SYMIN = YMIN
   SYMAX = YMAX
END IF

' 640x480
SCREEN 12

WINDOW (SXMIN, SYMAX)-(SXMAX, SYMIN)
LINE (XMIN, YMIN)-(XMAX, YMIN)
LINE (XMAX, YMIN)-(XMAX, YMAX)
LINE (XMAX, YMAX)-(XMIN, YMAX)
LINE (XMIN, YMAX)-(XMIN, YMIN)

LINE (XC, YMIN)-(XC, YMAX)
LINE (XMIN, YC)-(XMAX, YC)

NP = 100
PX = (XMAX - XMIN) / NP

X0 = XMIN
Y0 = F(X0)

FOR I = 1 TO NP
   X1 = X0 + PX
   Y1 = F(X1)
   LINE (X0, Y0)-(X1, Y1)
   X0 = X1
   Y0 = Y1
NEXT I

FUNCTION F (X!)
   F = SIN(X)
END FUNCTION
```

Resultado de la ejecución:



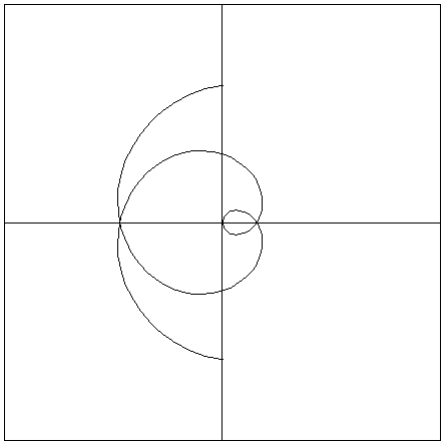
LISTADO 3 - REPRESENTACIÓN DE CURVAS X=F(T) Y=G(T)

En este otro listado se presenta un pequeño programa que permite representar curvas paramétricas del tipo $x=f(t)$, $y=g(t)$ definiendo los valores

máximo y mínimo de las dos variables y del parámetro t, así como el número de segmentos que se emplearán para representar la curva.

```
DECLARE FUNCTION FX! (T!)
DECLARE FUNCTION FY! (T!)

CONST PI = 3.141592654#

TMIN = -2 * PI: TMAX = 2 * PI
XMIN = -10: XMAX = 10
YMIN = -10: YMAX = 10
XS = XMAX - XMIN
YS = YMAX - YMIN
XC = .5 * (XMIN + XMAX)
YC = .5 * (YMIN + YMAX)
BORDER = 1.05

IF (XS / 4 > YS / 3) THEN
   SXMIN = XMIN
   SXMAX = XMAX
   SYMIN = YC - .5 * XS * 3 / 4
   SYMAX = YC + .5 * XS * 3 / 4
ELSE
   SXMIN = XC - .5 * YS * 4 / 3
   SXMAX = XC + .5 * YS * 4 / 3
   SYMIN = YMIN
   SYMAX = YMAX
END IF

' 640x480
SCREEN 12

WINDOW (BORDER * SXMIN, BORDER * SYMAX)-(BORDER * SXMAX, BORDER * SYMIN)
LINE (XMIN, YMIN)-(XMAX, YMIN)
LINE (XMAX, YMIN)-(XMAX, YMAX)
LINE (XMAX, YMAX)-(XMIN, YMAX)
LINE (XMIN, YMAX)-(XMIN, YMIN)

LINE (XC, YMIN)-(XC, YMAX)
LINE (XMIN, YC)-(XMAX, YC)

NP = 100: PT = (TMAX - TMIN) / NP
T = TMIN: X0 = FX(T): Y0 = FY(T)

FOR I = 1 TO NP
   T = T + PT: X1 = FX(T): Y1 = FY(T)
   LINE (X0, Y0)-(X1, Y1)
   X0 = X1: Y0 = Y1
NEXT I

FUNCTION FX (T!)
   FX = T * SIN(T)
END FUNCTION

FUNCTION FY (T!)
   FY = T * COS(T)
END FUNCTION
```

Resultado de la ejecución:



TRANSFORMACIONES 2D

INTRODUCCIÓN

Una de las mayores virtudes de los gráficos generados por ordenador es la facilidad con se pueden realizar algunas modificaciones sobre las imágenes. Un gerente puede cambiar la escalas de las gráficas de un informe. Un arquitecto puede ver un edificio desde distintos puntos de vista. Un cartógrafo puede cambiar la escala de un mapa. Un animador puede modificar la posición de un personaje. Estos cambios son fáciles de realizar porque la imagen gráfica ha sido codificada en forma de números y almacenada en el interior del ordenador. Los números son susceptibles a las operaciones matemáticas denominadas *transformaciones*.

Las transformaciones nos permiten alterar de una forma uniforme toda la imagen. Es un hecho que a veces es más fácil modificar toda la imagen que una porción de ella. Esto supone un complemento muy útil para las técnicas de dibujo manual, donde es normalmente más fácil modificar una pequeña porción del dibujo que crear un dibujo completamente nuevo.

Es este capítulo veremos transformaciones geométricas como el cambio de escala, la traslación y la rotación. Veremos como se expresan de una forma sencilla mediante multiplicaciones de matrices. Introduciremos las coordenadas homogéneas con el fin de tratar de una manera uniforme las transformaciones y como anticipo de las transformaciones producidas por la perspectiva en los modelos tridimensionales.

MATRICES

Las imágenes gráficas que hemos generado están compuestas por un conjunto de segmentos que están representados por las coordenadas de sus extremos. Algunos cambios en la imagen pueden ser fácilmente realizados mediante la aplicación de algunas operaciones matemáticas sobre estas coordenadas. Antes de ver algunas de las posibles transformaciones, repasemos algunas de las herramientas matemáticas que vamos a necesitar, como la multiplicación de matrices.

Para nuestro propósito, consideremos que una matriz es un conjunto bidimensional de números, por ejemplo:

$$\begin{vmatrix} 1 & 0 \\ 0 & 1 \end{vmatrix} \qquad \begin{vmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{vmatrix} \qquad \begin{vmatrix} 1 \\ -1 \\ 0 \end{vmatrix} \qquad \begin{vmatrix} 1 & 0 & 2 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{vmatrix}$$

son cuatro matrices diferentes.

Supongamos que definimos la matriz A como:

$$A = \begin{vmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{vmatrix}$$

Entonces el elemento situado en la segundo fila y la tercera columna será $A(2,3)$ y tendrá un valor de 6.

La operación matricial que más emplearemos será la multiplicación de matrices. La multiplicación de matrices es algo más complicada que la simple

multiplicación de dos números; supone varios productos sencillos y sumas de los elementos de la matriz. No todas las parejas de matrices pueden ser multiplicadas. Se pueden multiplicar dos matrices A y B si el número de columnas de la primera matriz A es igual al número de filas de la segunda matriz B. Por ejemplo, si escogemos como matriz A la última que hemos visto y como matriz B la siguiente:

$$B = \begin{vmatrix} 1 & 0 \\ -1 & 2 \\ 0 & 1 \end{vmatrix}$$

entonces podemos multiplicar A por B porque la primera tiene tres columnas y la segunda tres filas. Al contrario que la multiplicación de números, la multiplicación de matrices no es conmutativa, es decir, aunque podemos multiplicar A por B no podemos multiplicar B por A, por que B tiene sólo dos columnas que no se corresponden con las tres filas de A. Cuando multiplicamos dos matrices se obtiene como resultado otra matriz. Esta matriz producto tendrá el mismo número de filas que la primera de las matrices que se multiplican y el mismo número de columnas que la segunda. La multiplicación de la matriz 3×3 A con la matriz 3×2 B da como resultado la matriz 3×2 C.

Los elementos de la matriz producto C se expresan en función de los elementos de las matrices A por B mediante la siguiente fórmula:

$$C(i,k) = \sum_j A(i,j)B(j,k)$$

En nuestro caso particular de $C = AB$

$$C = \begin{vmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{vmatrix} \begin{vmatrix} 1 & 0 \\ -1 & 2 \\ 0 & 1 \end{vmatrix} = \begin{vmatrix} -1 & 7 \\ -1 & 16 \\ -1 & 25 \end{vmatrix}$$

La multiplicación de matrices es una operación *asociativa*. Esto significa que si tenemos varias matrices para multiplicar a la vez, no importa cuales multipliquemos primero. De forma matemática:

$$A(BC) = (AB)C$$

Esta es una propiedad muy útil; nos permitirá combinas varias transformaciones gráficas en una sola transformación, produciendo como resultado unos cálculos más eficientes.

Existe un conjunto de matrices que cuando multiplican a otra matriz, la reproducen. Por esta razón reciben el nombre de *matrices identidad*. Son matrices cuadradas (tienen el mismo número de columnas y de filas) con todos los elementos 0 excepto los elementos de la diagonal principal, que valen todos 1. Por ejemplo

$$|1| \qquad \begin{vmatrix} 1 & 0 \\ 0 & 1 \end{vmatrix} \qquad \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{vmatrix}$$

De forma matemática:

$$A = AI$$

CAMBIOS DE ESCALA

¿Cómo se aplica todo esto a los gráficos? Bueno, consideremos un punto P_1=[x_1 y_1] como una matriz 1×2. Si la multiplicamos por una matriz 2×2 T, obtendremos otra matriz 1×2 que puede ser interpretada como otro punto.

$$\begin{bmatrix} x_2 & y_2 \end{bmatrix} = P_2 = P_1 T$$

Por tanto, la matriz T es una aplicación entre el punto original P_1 y el nuevo punto P_2. Si suponemos nuestra imagen compuesta por los vértices de un polígono. ¿Qué pasará si transformamos cada uno de los punto mediante una multiplicación por una matriz T y dibujamos el resultado? ¿Qué aspecto tendrá esta nueva imagen? La respuesta, por supuesto, depende de los elementos de la matriz T. Si, por ejemplo, escogemos la matriz identidad entonces la imagen no se verá alterada.

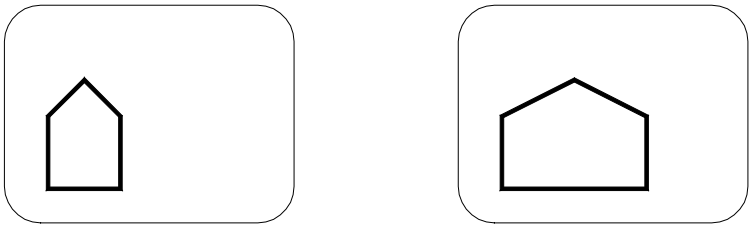
Sin embargo, si escogemos la matriz

$$T_1 = \begin{vmatrix} 2 & 0 \\ 0 & 1 \end{vmatrix}$$

entonces

$$\begin{bmatrix} x_2 & y_2 \end{bmatrix} = \begin{bmatrix} x_1 & y_1 \end{bmatrix} \begin{vmatrix} 2 & 0 \\ 0 & 1 \end{vmatrix} = \begin{bmatrix} 2x_1 & y_1 \end{bmatrix}$$

Cada una de las nuevas coordenadas x tiene el doble de valor que las antiguas. Las líneas horizontales serán dos veces más largas en la nueva imagen. La nueva imagen tendrá la misma altura, pero parecerá que la hemos estirado hasta alcanzar el doble del ancho original.

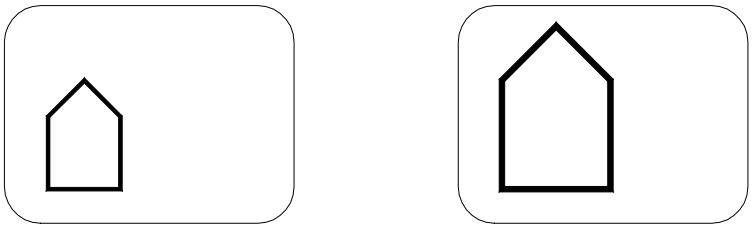


antes　　después

En general, las transformaciones de la forma

$$S = \begin{vmatrix} E_x & 0 \\ 0 & E_y \end{vmatrix}$$

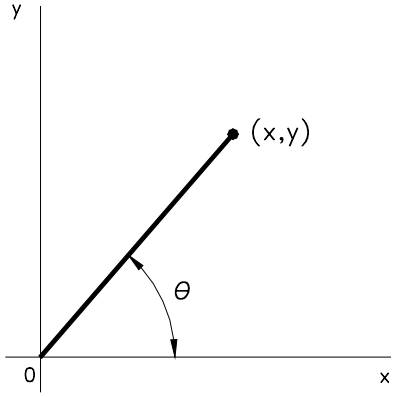
cambian el tamaño y la proporción de la imagen. Se denominan transformaciones de *escalado*. E_x es el *factor de escala* para la coordenada x y E_y es de la coordenada y.



antes　　después

RAZONES TRIGONOMÉTRICAS

La siguiente transformación gráfica que vamos a ver es la rotación. Para introducir esta transformación recordaremos brevemente algunos conceptos trigonométricos. Sea un punto $p_1=(x_1, y_1)$ y lo giramos alrededor del origen un ángulo θ para pasar a una nueva posición $p_2=(x_2, y_2)$. Queremos encontrar la transformación que convierte (x_1, y_1) en (x_2, y_2). Pero, antes de comprobar si alguna transformación es la adecuada, debemos saber primero que (x_2, y_2) debe escribirse en función de (x_1, y_1) y θ. Para esto es necesario recordar la razones trigonométricas de seno y coseno.



Definición de ángulo

A la vista de esta figura sabemos que

$$\operatorname{sen}\theta = \frac{y}{\sqrt{x^2+y^2}} \qquad \cos\theta = \frac{x}{\sqrt{x^2+y^2}}$$

Es importante señalar que cuando la longitud del segmento es la unidad

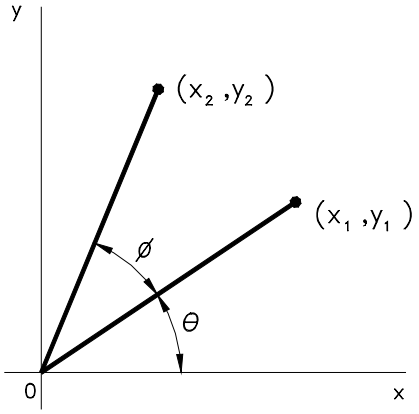
$$\operatorname{sen}\theta = y \qquad \cos\theta = x$$

También emplearemos las siguientes relaciones trigonométricas para determinar como gira un punto:

$$\operatorname{sen}(\theta+\phi) = \cos\phi\operatorname{sen}\theta + \operatorname{sen}\phi\cos\theta$$
$$\cos(\theta+\phi) = \cos\phi\cos\theta - \operatorname{sen}\phi\operatorname{sen}\theta$$

ROTACIÓN

Ahora estamos listo ya para determinar la rotación de un punto alrededor del origen.



Rotación de un punto alrededor del origen

A la vista de esta figura tenemos:

$$sen\theta = \frac{y_1}{L} \qquad \cos\theta = \frac{x_1}{L}$$

donde L es la distancia del punto al origen de coordenadas. Por otro lado:

$$\text{sen}(\theta + \phi) = \frac{y_2}{L} = \cos\phi\,\text{sen}\,\theta + \text{sen}\,\phi\cos\theta$$

que nos lleva a

$$y_2 = \cos\phi\,\text{sen}\,\theta L + \text{sen}\,\phi\cos\theta L = \cos\phi\, y_1 + \text{sen}\,\phi\, x_1$$

De forma análoga:

$$\cos(\theta + \phi) = \frac{x_2}{L} = \cos\phi\cos\theta - \text{sen}\,\phi\,\text{sen}\,\theta$$

dando

$$x_2 = \cos\phi\cos\theta L - \text{sen}\,\phi\,\text{sen}\,\theta L = \cos\phi\, x_1 - \text{sen}\,\phi\, y_1$$

A la vista de estas ecuaciones podemos imaginar una matriz que relacione las coordenadas del punto original y del punto girado:

$$\begin{bmatrix} x_2 & y_2 \end{bmatrix} = \begin{bmatrix} x_1 & y_1 \end{bmatrix} \begin{vmatrix} \cos\phi & \text{sen}\,\phi \\ -\text{sen}\,\phi & \cos\phi \end{vmatrix}$$

Así la matriz de transformación para una rotación en sentido contrario a las agujas del reloj de ángulo ϕ alrededor del origen es

$$R = \begin{vmatrix} \cos\phi & \text{sen}\,\phi \\ -\text{sen}\,\phi & \cos\phi \end{vmatrix}$$

Para una rotación en el sentido de las agujas del reloj, basta sustituir en la expresión anterior el valor del ángulo por $-\phi$. Así nos queda

$$R = \begin{vmatrix} \cos\phi & -\text{sen}\,\phi \\ \text{sen}\,\phi & \cos\phi \end{vmatrix}$$

COORDENADAS HOMOGÉNEAS Y TRASLACIÓN

Supongamos que necesitamos realizar un giro alrededor de un punto que no es el origen. Si fuésemos capaces de trasladar toda la imagen de un punto a otro de la pantalla, podríamos realizar este giro moviendo primero la imagen hasta que el centro de rotación coincida con el origen, luego realizamos la rotación y, por último, devolvemos la imagen a su posición original.

Desplazar la imagen recibe el nombre de *traslación*. Se realiza de una forma sencilla mediante la suma a cada punto de la cantidad que vamos a mover la imagen.

En general, con el fin de trasladar un imagen (T_x, T_y), cada punto (x_1, y_1) se convierte en uno nuevo (x_2, y_2) donde

$$x_2 = x_1 + T_x \qquad y_2 = y_1 + T_y$$

Desafortunadamente, esta forma de describir la traslación no hace uso de matrices, por lo tanto no podría ser combinada con las otras transformaciones mediante una simple multiplicación de matrices. Tal combinación sería deseable; por ejemplo, hemos visto que la rotación alrededor de un punto que no sea el

origen puede realizarse mediante una traslación, una rotación u otra traslación. Sería deseable combinar estas tres transformaciones en una sola transformación por motivos de eficacia y elegancia. Una forma de hacer esto es emplear matrices 3×3 en vez de matrices 2×2, introduciendo una coordenada auxiliar w. Este método recibe el nombre de *coordenadas homogéneas*. En estas coordenadas, los puntos están definidos por tres coordenadas y no por dos. Así un punto (x, y) estará representado por la tripleta (xw, yw, w). Las coordenadas x e y se pueden recuperar fácilmente dividiendo los dos primeros números por el tercero respectivamente. No emplearemos la coordenada w hasta que no veamos las transformaciones tridimensionales de perspectiva. En dos dimensiones su valor suele ser 1 para simplificar. Sin embargo, lo veremos de forma general como anticipo de las transformaciones tridimensionales.

En coordenadas homogéneas la matriz de cambio de escala

$$S = \begin{vmatrix} E_x & 0 \\ 0 & E_y \end{vmatrix}$$

se convierte en

$$S = \begin{vmatrix} E_x & 0 & 0 \\ 0 & E_y & 0 \\ 0 & 0 & 1 \end{vmatrix}$$

Si aplicamos esta matriz a un punto (xw, yw, w) obtenemos

$$\begin{bmatrix} xw & yw & w \end{bmatrix} \begin{vmatrix} E_x & 0 & 0 \\ 0 & E_y & 0 \\ 0 & 0 & 1 \end{vmatrix} = \begin{bmatrix} E_x xw & E_y yw & w \end{bmatrix}$$

Si dividimos ahora por el tercer valor w tenemos ($E_x x$, $E_y y$) que es el punto correcto cambiado de escala. En el caso de la matriz de rotación en sentido antihorario

$$R = \begin{vmatrix} \cos\theta & \operatorname{sen}\theta \\ -\operatorname{sen}\theta & \cos\theta \end{vmatrix}$$

se convierte, usando coordenadas homogéneas, en:

$$R = \begin{vmatrix} \cos\theta & \operatorname{sen}\theta & 0 \\ -\operatorname{sen}\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{vmatrix}$$

Si aplicamos esta matriz a un punto (xw, yw, w) obtenemos

$$\begin{bmatrix} xw & yw & w \end{bmatrix} \begin{vmatrix} \cos\theta & \operatorname{sen}\theta & 0 \\ -\operatorname{sen}\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{vmatrix} = \begin{bmatrix} (xw\cos\theta - yw\operatorname{sen}\theta) & (xw\operatorname{sen}\theta + yw\cos\theta) & w \end{bmatrix}$$

para dar el punto correctamente rotado

$$(x\cos\theta - y\operatorname{sen}\theta,\ x\operatorname{sen}\theta + y\cos\theta)$$

La matriz de transformación para una traslación T_x, T_y en coordenadas homogéneas es

$$T = \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ T_x & T_y & 1 \end{vmatrix}$$

Para comprobar que esto es así apliquemos la matriz a un punto genérico

$$\begin{bmatrix} xw & yw & w \end{bmatrix} \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ T_x & T_y & 1 \end{vmatrix} = \begin{bmatrix} (xw + T_x w) & (yw + T_y w) & w \end{bmatrix}$$

que nos da el punto trasladado $(x + T_x,\ \ y + T_y)$.

ROTACIÓN ALREDEDOR DE UN PUNTO CUALQUIERA

Determinemos ahora la matriz de transformación para la rotación en sentdio antihorario alrededor del punto (x_C, y_C).

Haremos esta transformación en tres pasos. Primero trasladaremos el punto (x_C, y_C) al origen, luego haremos la rotación alrededor del origen y, por último, devolveremos el centro de rotación a su posición original.

La traslación que desplaza al punto (x_C, y_C) al origen es

$$T_1 = \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -x_C & -y_C & 1 \end{vmatrix}$$

la rotación es

$$R = \begin{vmatrix} \cos\theta & \operatorname{sen}\theta & 0 \\ -\operatorname{sen}\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{vmatrix}$$

y la traslación que devuelve al centro de rotación a su posición es

$$T_2 = \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ x_C & y_C & 1 \end{vmatrix}$$

Para transformar un punto podemos realizar la siguiente multiplicación

$$\left(\left(\left[xw \ \ yw \ \ w \right] T_1 \right) R \right) T_2 \right)$$

pero, teniendo en cuenta la propiedad asociativa de la multiplicación de matrices, podemos multiplicar todas las transformaciones primero para obtener la matriz global de transformación

$$\left[xw \ \ yw \ \ w \right] \left(T_1 \left(R \ T_2 \right) \right)$$

$$T_1\ R\ T_2 = \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -x_C & -y_C & 1 \end{vmatrix} \begin{vmatrix} \cos\theta & \operatorname{sen}\theta & 0 \\ -\operatorname{sen}\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{vmatrix} \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ x_C & y_C & 1 \end{vmatrix} =$$

$$= \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -x_C & -y_C & 1 \end{vmatrix} \begin{vmatrix} \cos\theta & \operatorname{sen}\theta & 0 \\ -\operatorname{sen}\theta & \cos\theta & 0 \\ x_C & y_C & 1 \end{vmatrix} =$$

$$= \begin{vmatrix} \cos\theta & \operatorname{sen}\theta & 0 \\ -\operatorname{sen}\theta & \cos\theta & 0 \\ -x_C\cos\theta + y_C\operatorname{sen}\theta + x_C & -x_C\operatorname{sen}\theta - y_C\cos\theta + y_C & 1 \end{vmatrix}$$

Cabe destacar que esta matriz se puede formar también mediante una rotación inicial de ángulo θ y una traslación definida por los valores contenidos en la tercera fila.

OTRAS TRANSFORMACIONES

Las tres transformaciones de cambio de escala, rotación y traslación son las más útiles y las más usadas. Son también posibles otras transformaciones. Dado que una matriz 2×2 cualquiera

$$\begin{vmatrix} a & b \\ c & d \end{vmatrix}$$

puede convertirse en una matriz 3×3 en coordenadas homogéneas como

$$\begin{vmatrix} a & b & 0 \\ c & d & 0 \\ 0 & 0 & 1 \end{vmatrix}$$

sólo presentaremos algunas de estas transformaciones como matrices 2×2:

$\begin{vmatrix} -1 & 0 \\ 0 & 1 \end{vmatrix}$ reflexión respecto al eje y

$\begin{vmatrix} 1 & 0 \\ 0 & -1 \end{vmatrix}$ reflexión respecto al eje x

$\begin{vmatrix} -1 & 0 \\ 0 & -1 \end{vmatrix}$ reflexión respecto al origen

$\begin{vmatrix} 0 & 1 \\ 1 & 0 \end{vmatrix}$ reflexión respecto a la recta $y=x$

$\begin{vmatrix} 0 & -1 \\ -1 & 0 \end{vmatrix}$ reflexión respecto a la recta $y=-x$

$\begin{vmatrix} 1 & a \\ 0 & 1 \end{vmatrix}$ deformación en el eje y

$\begin{vmatrix} 1 & 0 \\ b & 1 \end{vmatrix}$ deformación en el eje x

Las primeras tres reflexiones son simples cambios de escala pero con factores negativos. Las simetrías respecto a las rectas $y=x$ e $y=-x$ pueden realizarse mediante un cambio de escale y un giro posterior. Es posible realizar las deformaciones mediante una secuencia de rotaciones y cambios de escala, aunque es mucho más fácil aplicar construir la matriz resultante. De igual forma se pueden construir transformaciones de cambio de escala y rotación a partir de las deformaciones.

TRANSFORMACIONES 3D

INTRODUCCIÓN

Algunas aplicaciones gráficas son bidimensionales: dibujos y gráficos, algunos mapas y creaciones artísticas pueden ser entidades estrictamente bidimensionales. Pero vivimos en un mundo tridimensional, y en muchas aplicaciones de diseño debemos manejar y describir objetos tridimensionales. Si un arquitecto desease ver el aspecto real de la estructura, entonces un modelo tridimensional le permitiría observarla desde diferentes puntos de vista. Un diseñador de aviones podría desear analizar el comportamiento de la nave bajo fuerzas y tensiones tridimensionales. En este caso se necesita también una descripción tridimensional. Algunas aplicaciones de simulación, como el aterrizaje de un avión, también exigen una definición tridimensional del mundo.

En este capítulo generalizaremos nuestro sistema para manejar modelos de objetos tridimensionales. Extenderemos las transformaciones que hemos visto para permitir traslaciones y rotaciones en el espacio tridimensional. Ya que la superficie de visualización es sólo bidimensional, debemos considerar la forma de proyectar nuestros objetos en esta superficie plana para formar la imagen. Se discutirán tanto las proyecciones paralelas como perspectivas.

GEOMETRÍA 3D

Empecemos nuestra discusión sobre las tres dimensiones revisando los métodos de la geometría analítica para definir objetos. El objeto más sencillo es, por supuesto, un punto. Como en el caso de las dos dimensiones, podemos definir un punto estableciendo un sistema de coordenadas y listando las coordenadas del punto. Necesitamos un eje adicional de coordenadas para la tercera dimensión. Podemos disponer este tercer eje para que forme ángulo recto con los otros dos.

Podemos utilizar este nuevo sistema de coordenadas para definir la ecuación de una recta. En dos dimensiones teníamos:

$$\frac{y - y_1}{x - x_1} = \frac{y_2 - y_1}{x_2 - x_1}$$

mientras que en el caso tridimensional, son necesarias un par de ecuaciones

$$\frac{y - y_1}{x - x_1} = \frac{y_2 - y_1}{x_2 - x_1}$$

$$\frac{z - z_1}{x - x_1} = \frac{z_2 - z_1}{x_2 - x_1}$$

Son necesarias las coordenadas de dos puntos para construir estas ecuaciones (x_1, y_1, z_1) y (x_2, y_2, z_2). Por lo tanto, todavía son necesarios dos puntos para definir una recta, como cabría esperar. De forma paramétrica, donde cada variable se expresa en función de un parámetro u:

$$x = (x_2 - x_1)u + x_1$$

$$y = (y_2 - y_1)u + y_1$$

$$z = (z_2 - z_1)u + z_1$$

Nos gustaría también trabajar con planos. Un plano, de forma general, se especifica mediante una sola ecuación de la forma:

$$Ax + By + Cz + D = 0$$

Cabe señalar que si dividimos por A la ecuación anterior tenemos

$$x + B_1 y + C_1 z + D_1 = 0$$

Esto significa que sólo son necesarias tres constantes para definir un plano. Las ecuaciones para un plano en particular pueden ser determinadas si conocemos las coordenadas de tres puntos (que no estén alineados) (x_1, y_1, z_1), (x_2, y_2, z_2) y (x_3, y_3, z_3). Podemos determina la ecuación de la siguiente manera, ya que cada punto del plano debe satisfacer la ecuación del plano

$$x_1 + B_1 y_1 + C_1 z_1 + D_1 = 0$$
$$x_2 + B_1 y_2 + C_1 z_2 + D_1 = 0$$
$$x_3 + B_1 y_3 + C_1 z_3 + D_1 = 0$$

Tenemos así tres ecuaciones con tres incógnitas que podemos resolver con la ayuda de un poco de álgebra.

Otra forma de especificar un plano es mediante un único punto que pertenezca al plano (x_p, y_p, z_p) y la dirección perpendicular al plano $[x_N, y_N, z_N]$. Este vector recibe el nombre de *vector normal*. Por otro lado, si (x, y, z) es un punto genérico del plano, entonces $[x - x_p, y - y_p, z - z_p]$ es un vector de ese plano. Si tenemos en cuenta que estos dos vectores forman un ángulo recto, su producto escalar será nulo:

$$x_N(x - x_p) + y_N(y - y_p) + z_N(z - z_p) = 0$$

Esta expresión es válida cuando (x, y, z) es un punto del plano, por lo tanto, esta es la ecuación de ese plano.

CAMBIO DE ESCALA

Ahora que sabemos expresar puntos, rectas y planos en tres dimensiones, consideremos los algoritmos que permiten al usuario modelar objetos tridimensionales.

Empezaremos por generalizar las transformaciones bidimensionales que vimos en el capítulo anterior. Así, por lo que respecta al cambio de escala, la tercera coordenada que hemos introducido pueden tener su propio facto de escala, por lo tanto será necesaria una matriz 3×3

$$\begin{vmatrix} E_x & 0 & 0 \\ 0 & E_y & 0 \\ 0 & 0 & E_z \end{vmatrix}$$

o de rango 4×4 si empleamos coordenadas homogéneas

$$E = \begin{vmatrix} E_x & 0 & 0 & 0 \\ 0 & E_y & 0 & 0 \\ 0 & 0 & E_z & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

TRASLACIÓN

Igual que en el caso bidimensional, empleamos los elementos de la fila inferior de la matriz de transformación en coordenadas homogéneas para reflejar la traslación.

$$T = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ T_x & T_y & T_z & 1 \end{vmatrix}$$

ROTACIÓN

Cuando consideramos la rotación de un objeto en dos dimensiones, vimos la matriz de rotación alrededor del origen

$$R = \begin{vmatrix} \cos\theta & \operatorname{sen}\theta \\ -\operatorname{sen}\theta & \cos\theta \end{vmatrix}$$

Podemos generalizar este concepto a una rotación tridimensional alrededor del eje z.

$$R_z = \begin{vmatrix} \cos\theta & \operatorname{sen}\theta & 0 & 0 \\ -\operatorname{sen}\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

En esta rotación, pensamos en el eje como fijo mientras que algún objeto se mueve en el espacio. Podemos pensar también que el objeto permanece inmóvil mientras los ejes se mueven. La diferencia entre uno y otro planteamiento es la dirección de la rotación. Fijar el eje y girar el objeto en sentido antihorario es lo mismo que fijar el objeto y mover el eje en sentido horario.

La rotación alrededor de los ejes x e y se realiza con una matriz de transformación similar a la anterior.

$$R_x = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & \operatorname{sen}\theta & 0 \\ 0 & -\operatorname{sen}\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

Para hacer una rotación alrededor del eje y, del tal forma que el eje z se convierte en el x utilizamos la siguiente matriz en coordenadas homogéneas

$$R_y = \begin{vmatrix} \cos\theta & 0 & -\operatorname{sen}\theta & 0 \\ 0 & 1 & 0 & 0 \\ \operatorname{sen}\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

ROTACIÓN ALREDEDOR DE UN EJE ARBITRARIO

Para que las transformaciones tridimensionales nos sean un poco más familiares, resolvamos un sencillo problema. Aunque hemos visto matrices de transformación que nos permiten girar cualquier objeto alrededor de uno de los

ejes de coordenadas, en general cualquier recta del espacio puede servir como eje de rotación. El problema consiste en establecer la matriz de rotación para un ángulo φ alrededor de una recta cualquiera. Construiremos esta transformación a partir de las que ya conocemos. Moveremos primero el origen hasta que esté situado sobre la recta. Luego realizaremos rotaciones alrededor de los ejes x e y para alinear el eje z con la recta. De esta forma la rotación alrededor de la recta se convierte en un giro alrededor del eje z. Finalmente, realizaremos transformaciones inversas a la realizadas en los giros alrededor de los ejes x e y para deshacer, por último, la traslación del origen a su posición original.

Lo primero que debemos hacer es escoger una representación adecuada de la recta que será el eje de rotación. Un punto de dicha recta, así como la dirección de la misma son suficientes para definir la recta (ecuación vectorial). El punto de la recta nos ofrece la información necesaria para la traslación del origen, y la dirección nos informa de los ángulos de rotación correctos para alinear la recta con el eje z.

Dada la recta

$$x = Au + x_1$$
$$y = Bu + y_1$$
$$z = Cu + z_1$$

un punto de esta recta es (x_1, y_1, z_1) y su dirección está especificada por el vector $[A, B, C]$.

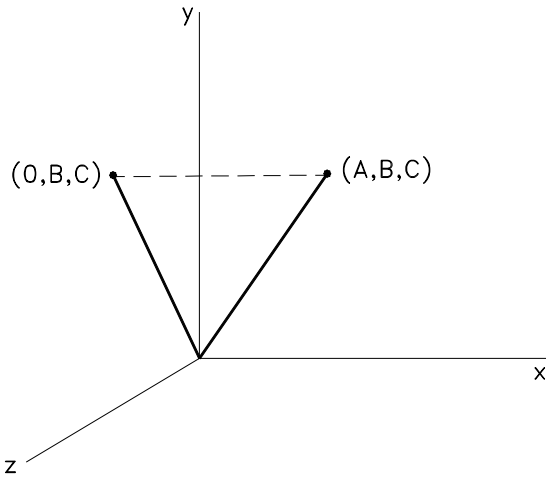
La matriz de transformación que traslada el origen al eje de giro será:

$$T = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -x_1 & -y_1 & -z_1 & 1 \end{vmatrix}$$

Necesitaremos también la inversa de traslación para devolver el origen a su posición original una vez que las rotaciones se hayan completado.

$$T^{-1} = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ x_1 & y_1 & z_1 & 1 \end{vmatrix}$$

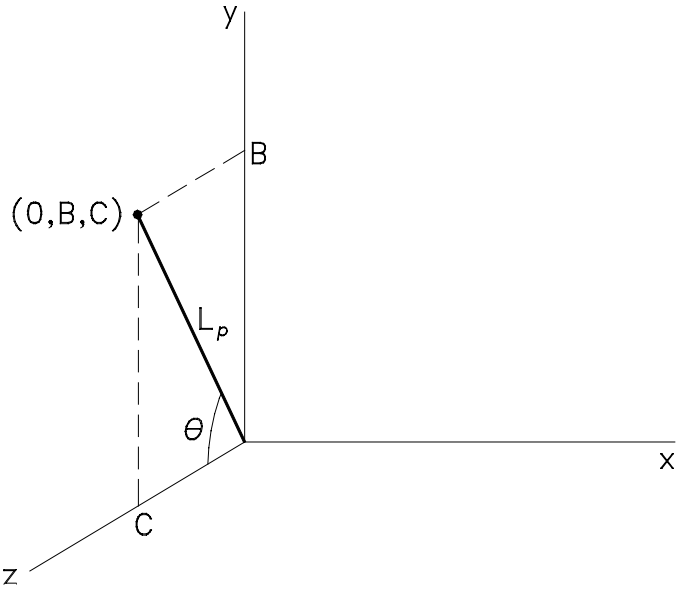
El siguiente paso en este proceso es la rotación alrededor del eje x. Deseamos realizar un giro que sitúe al eje arbitrario de rotación en el plano xz. Para determinar el ángulo de giro necesario, pongamos nuestro vector de dirección en el nuevo origen y consideremos su proyección sobre el plano yz. Podemos imaginar esto de la siguiente forma: el segmento de recta comprendido entre el origen $(0, 0, 0)$ y (A, B, C) está en la dirección del eje arbitrario de rotación, y todo el segmento está situado sobre dicho eje. Si ahora iluminamos este segmento con un haz de luz paralela al eje x y echamos un vistazo a la sombra que proyecta sobre el plano yz comprobamos que esta se extiende desde $(0, 0, 0)$ hasta $(0, B, C)$.



Si ahora realizamos un giro alrededor del eje x hasta que nuestro eje arbitrario quede en el plano xz, el segmento proyectado quedará alineado con el eje z. ¿Cuanto vale el ángulo θ?

Sabemos que la longitud del segmento proyectado es

$$L_p = \sqrt{B^2 + C^2}$$



y, teniendo en cuenta la figura anterior, es fácil deducir las razones trigonométricas del ángulo θ

$$\operatorname{sen}\theta = B/L_p$$
$$\cos\theta = C/L_p$$

Por lo tanto, la matriz de transformación para el giro alrededor del eje x será

$$R_x = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & \operatorname{sen}\theta & 0 \\ 0 & -\operatorname{sen}\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & C/L_p & B/L_p & 0 \\ 0 & -B/L_p & C/L_p & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

La transformación inversa es una rotación de igual magnitud pero en sentido contrario. Como sabemos, cambiar el signo del ángulo significa cambiar el signo de los senos y conservar el de los cosenos.

$$R_x^{-1} = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & C/L_p & -B/L_p & 0 \\ 0 & B/L_p & C/L_p & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

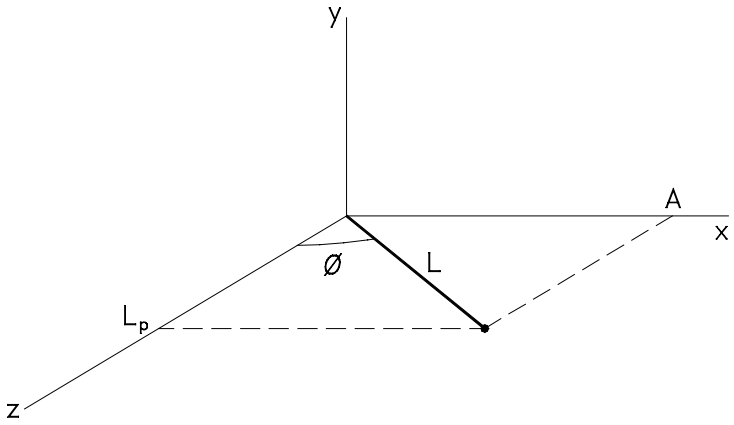
La rotación alrededor de este eje no ha modificado la coordenada x. Sabemos también que la longitud total del segmento

$$L = \sqrt{A^2 + B^2 + C^2}$$

no varía. La coordenada z será entonces

$$\sqrt{L^2 - A^2} = \sqrt{B^2 + C^2} = L_p$$

Necesitamos realizar ahora un giro de ángulo ϕ alrededor del eje y para que el segmento quede alineado con el eje z.



Sus razones trigonométricas son

$$\operatorname{sen}\phi = A/L$$
$$\cos\phi = L_p/L$$

La matriz de rotación en este caso será

$$R_y = \begin{vmatrix} \cos\phi & 0 & \operatorname{sen}\phi & 0 \\ 0 & 1 & 0 & 0 \\ -\operatorname{sen}\phi & 0 & \cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} = \begin{vmatrix} L_p/L & 0 & A/L & 0 \\ 0 & 1 & 0 & 0 \\ -A/L & 0 & L_P/L & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

La inversa de esta transformación será

$$R_y^{-1} = \begin{vmatrix} L_p/L & 0 & -A/L & 0 \\ 0 & 1 & 0 & 0 \\ A/L & 0 & L_P/L & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

Es posible ahora realizar el giro φ alrededor del eje escogido. Ya que este eje está alineado con el eje z, la matriz correspondiente a esta rotación será

$$R_z = \begin{vmatrix} \cos\varphi & \operatorname{sen}\varphi & 0 & 0 \\ -\operatorname{sen}\varphi & \cos\varphi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

De esta forma ya somos capaces de realizar una rotación φ alrededor de un eje arbitrario, mediante el producto de las transformaciones anteriores

$$R_\varphi = TR_xR_yR_zR_y^{-1}R_x^{-1}T^{-1}$$

PROYECCIÓN PARALELA

En los apartados anteriores hemos visto como manipular objetos tridimensionales, pero nuestra superficie de visualización es bidimensional. Por lo tanto, es necesario que podamos proyectar estos objetos en la pantalla de nuestro ordenador.

Quizás la forma más sencilla de hacer esto sea descarta la coordenada z de los puntos que definen el objeto. Este es un caso particular de un método conocido con el nombre de *proyección paralela*. Una proyección paralela se consigue trazando rectas paralelas desde cada uno de los vértices del objeto hasta que intersectan al plano de la pantalla. El punto de intersección es la proyección de cada vértice. Los puntos de proyección se conectan mediante segmentos que corresponden a las conexiones existentes en el objeto original.

Nuestro caso especial, en el que se elimina la coordenada z, es aquel en el que la pantalla, o superficie de visualización, es paralela al plano xy y las líneas de proyección son paralelas al eje z. A medida que nos movemos por las rectas de proyección sólo cambia la coordenada z; las coordenadas x e y permanecen constantes. Así, el punto de intersección con la pantalla tiene las mismas coordenadas x e y que el vértice que estamos proyectando.

En el caso general de proyección paralela, se puede escoger cualquier dirección para las rectas de proyección (siempre y cuando no sean paralelas a la superficie de proyección). Supongamos que la dirección de proyección está dada por el vector $\begin{bmatrix} x_p & y_p & z_p \end{bmatrix}$ y que la imagen va ser proyectada sobre el plano xy. Si tenemos un punto (x_1, y_1, z_1) del objeto, queremos determinar el punto proyectado (x_2, y_2). Empecemos por definir las ecuaciones de la recta que pasa por un punto (x, y, z) y tiene la dirección de las rectas de proyección. De forma paramétrica

$$\begin{aligned} x &= x_1 + x_p u \\ y &= y_1 + y_p u \\ z &= z_1 + z_p u \end{aligned}$$

Ahora nos preguntamos ¿dónde corta esta línea al plano xy? Es decir, cuales son los valores de x e y cuando z=0.

$$u = -\frac{z_1}{z_p} \; ; \; \begin{aligned} x_2 &= x_1 - z_1\left(x_p / z_p\right) \\ y_2 &= y_1 - z_1\left(y_p / z_p\right) \end{aligned}$$

Estas ecuaciones son de hecho una transformación que se pueden escribir en forma matricial

$$\begin{bmatrix} x_2 & y_2 \end{bmatrix} = \begin{bmatrix} x_1 & y_1 & z_1 \end{bmatrix} \begin{vmatrix} 1 & 0 \\ 0 & 1 \\ -x_p/z_p & -y_p/z_p \end{vmatrix}$$

o, de otra forma, en coordenadas homogéneas

$$\begin{bmatrix} x_2 & y_2 & z_2 & 1 \end{bmatrix} = \begin{bmatrix} x_1 & y_1 & z_1 & 1 \end{bmatrix} \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -x_p/z_p & -y_p/z_p & 0 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

PROYECCIÓN EN PERSPECTIVA

Un método alternativo para realizar la proyección de un objeto tridimensional sobre la pantalla de un ordenador es la *proyección en perspectiva*. En este tipo de proyección, cuanto más lejos está un objeto del observador, más pequeño parece. Esto permite al observador tener una idea de *profundidad*, una indicación de que porciones de la imagen corresponden a las partes del objetos que están más cerca o más lejos del observador. En la proyección en perspectiva las líneas de proyección no son paralelas, sino que convergen en un punto denominado *centro de proyección*. Estas podrían ser los rayos de luz que procedentes del objeto llegan a los ojos del observador. Será la intersección de estas rectas convergentes con el plano de la pantalla las que definan la imagen proyectada.

Si el centro de proyección está en (x_c, y_c, z_c) y el punto del objeto es (x_1, y_1, z_1) entonces el rayo proyectante será la recta que pasa por estos dos puntos y está dada, en forma paramétrica, por las ecuaciones

$$\begin{aligned} x &= x_c + (x_1 - x_c)u \\ y &= y_c + (y_1 - y_c)u \\ z &= z_c + (z_1 - z_c)u \end{aligned}$$

En punto proyectado (x_2, y_2) será el punto de intersección de esta recta con el plano xy, donde la coordenada z es nula.

$$u = -\frac{z_c}{z_1 - z_c} \; ; \quad \begin{aligned} x_2 &= x_c - z_c \frac{x_1 - x_c}{z_1 - z_c} \\ y_2 &= y_c - z_c \frac{y_1 - y_c}{z_1 - z_c} \end{aligned}$$

de otra forma

$$\begin{aligned} x_2 &= \frac{x_c z_1 - x_1 z_c}{z_1 - z_c} \\ y_2 &= \frac{y_c z_1 - y_1 z_c}{z_1 - z_c} \end{aligned}$$

Esta proyección puede ponerse en la forma de una matriz de transformación si tenemos en cuenta las propiedades de las coordenadas homogéneas. La forma de esta matriz será

$$P = \begin{vmatrix} -z_c & 0 & 0 & 0 \\ 0 & -z_c & 0 & 0 \\ x_c & y_c & 0 & 1 \\ 0 & 0 & 0 & -z_c \end{vmatrix}$$

Para demostrar que esta transformación funciona, consideremos el punto (x_1, y_1, z_1), en coordenadas homogéneas tendremos $[x_1w_1 \;\; y_1w_1 \;\; z_1w_1 \;\; w_1]$ que multiplicado por la matriz anterior nos da

$$[x_2w_2 \;\; y_2w_2 \;\; z_2w_2 \;\; w_2] = [x_1w_1 \;\; y_1w_1 \;\; z_1w_1 \;\; w_1] \begin{vmatrix} -z_c & 0 & 0 & 0 \\ 0 & -z_c & 0 & 0 \\ x_c & y_c & 0 & 1 \\ 0 & 0 & 0 & -z_c \end{vmatrix} =$$

$$= [-x_1w_1z_c + z_1w_1x_c \;\; -y_1w_1z_c + z_1w_1y_c \;\; 0 \;\; z_1w_1 - z_cw_1]$$

De esta expresión se deducen fácilmente las coordenadas del punto proyectado que coinciden con las obtenidas anteriormente de forma analítica.

PARÁMETROS DE VISUALIZACIÓN

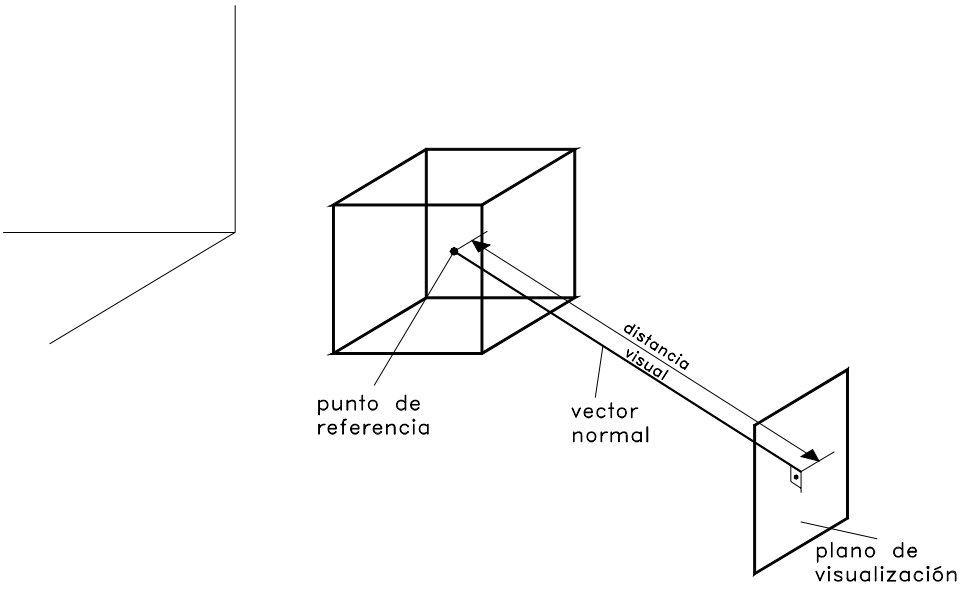
Hemos visto hasta ahora como realizas proyecciones paralelas y en perspectiva para forma la imagen bidimensional de un objeto tridimensional tal como un observador lo vería desde el frente. Pero ¿cómo podríamos ver este mismo objeto desde un lado, desde arriba, o incluso desde atrás? Todo lo que necesitamos hacer son varías rotaciones antes de realizar la proyección. Existen, como ya hemos visto, dos forma equivalentes de pensar acerca de esto. Podemos pensar que el plano de proyección (es decir, la pantalla del ordenador) es algo fijo y el objeto es el que gira, o podemos pensar en el objeto como lago fijo y que el plano de proyección cambia de posición en cada caso.

Este segundo planteamiento es el que vamos a ver a continuación. Podemos considerar que el plano de proyección es la película de una cámara de fotos. La cámara puede moverse en cualquier dirección y así es posible ver el objeto desde cualquier ángulo. La imagen capturada por la cámara es lo que veremos en la pantalla.

El primer parámetro que debemos tener en cuenta son las coordenadas del *punto de referencia* (x_R, y_R, z_R). Este punto es algo así como el centro de atención. El resto de los parámetros de visualización se expresan respecto a este punto. Si rotamos la vista, será una rotación alrededor del punto de referencia (no alrededor del origen). Podemos pensar en el punto de referencia como un gancho donde hemos atado una cuerda. Nuestra cámara sintética está atada al otro extremo de esta cuerda. Mediante el cambio del resto de parámetros de visualización, podemos desplazar la cámara formando un arco cambiar la longitud de la cuerda. Uno de los extremos de la cuerda está siempre unido al punto de referencia.

La dirección de esta cuerda imaginaría esta dada por el *vector normal* al plano de visualización. Este vector sigue la dirección perpendicular al plano de proyección (que es la película de nuestra cámara). Esto significa que la cámara siempre mira hacia el punto de referencia siguiendo la dirección de la cuerda.

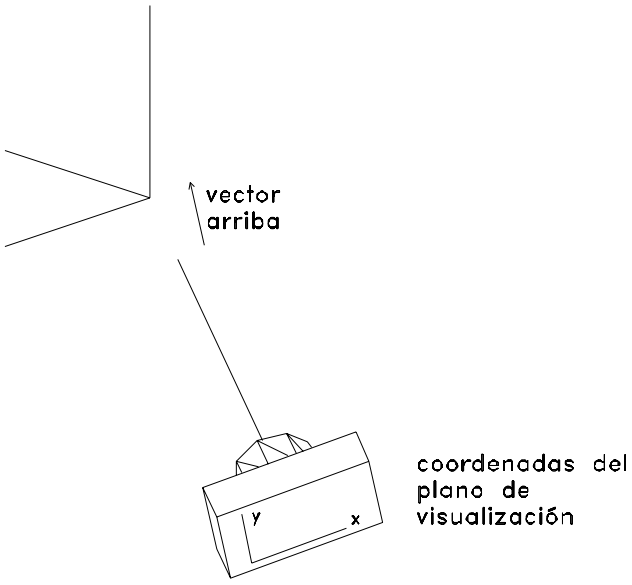
La longitud de esta cuerda recibe el nombre de *distancia visual*. Este parámetro indica lo lejos que está la cámara del punto de referencia. El plano de visualización esta situado a esta distancia del punto de referencia en la dirección del vector normal.



Parámetros de visualización tridimensionales

Tenemos así dos sistemas de coordenadas: las coordenadas del objeto que empleamos para construirlo y las coordenadas del plano de visualización. Podemos situar el origen de las coordenadas del plano de proyección en el punto donde la cuerda sujeta a la cámara, es decir, donde una recta paralela al vector normal, que pasa por el punto de referencia, corta al plano de visualización.

Queda todavía otro parámetro que debe ser discutido. Si mantenemos la cuerda fija podemos rotar la cámara alrededor de ella tomándola como eje de giro. Para cada ángulo, la imagen capturada mostrará la misma escena, pero girada de tal forma que una parte diferente del objeto estará orientada hacia arriba.



La dirección hacia *arriba* $\left[x_A,\ y_A,\ z_A\right]$ fija el ángulo de giro de la cámara.

Por otro lado, si cambiamos el punto de referencia también cambiará la porción del objeto que está situada en el origen de coordenadas

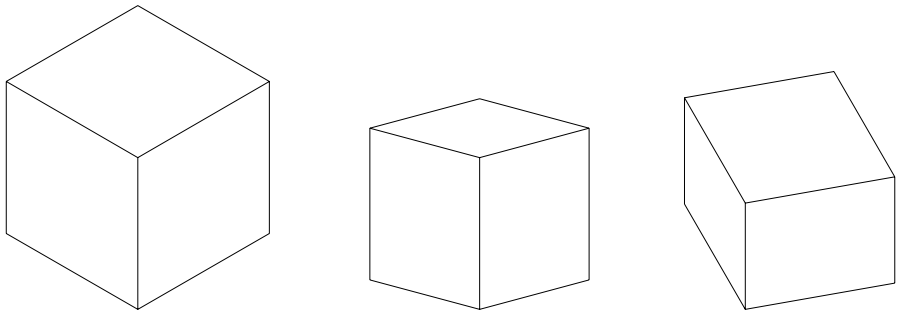
PROYECCIONES ESPECIALES

El problema de representar en un superficie bidimensional una vista de un objeto tridimensional ya existía mucho tiempo antes de la aparición de los ordenadores. Una clase de proyección muy usada en el dibujo industrial es la proyección *axonométrica*. Estas son proyecciones paralelas en las que la dirección de proyección es perpendicular al plano de visualización. Podemos modificar la dirección de una proyección axonométrica para obtener una vista diferente del objeto, siempre que el plano de visualización siga siendo perpendicular a la nueva dirección de proyección. Supongamos que estamos mirando a un cubo cuyos lados son paralelos a los ejes de coordenadas. Podemos empezar con una vista perpendicular a una de las caras del cubo, de tal forma que el cubo parezca un cuadrado.

Si desplazamos la dirección de proyección ligeramente hacia un lado, entonces una de las caras laterales de cubo se hará visible, mientras que los lados de la cara frontal se acortarán. Si elevamos el ángulo de visión, los lados de la cara superior se alargan mientras que las aristas de las caras laterales se acortan.

Existe una dirección de proyección en particular en la que todas las aristas aparecen afectadas por el mismo factor de escala. Esta dirección especial recibe el nombre de proyección *isométrica*. Una proyección isométrica de un cubo mostrará una esquina del cubo en el medio de la imagen rodeada por tres caras idénticas.

Si escogemos una transformación en la que sólo los lados paralelos a dos de los ejes de coordenadas se ven afectados por el mismo cambio de escala, entonces la proyección recibe el nombre de *dimétrica*. Una proyección *trimétrica* es aquella en la todos los lados se ven afectados por diferentes factores de escala.

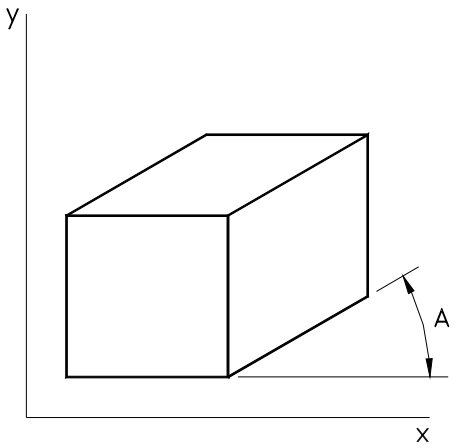


Ejemplos de proyecciones isométrica, dimétrica y trimétrica

Si la dirección de una proyección paralela no es perpendicular al plano de visualización tenemos lo que se denomina proyección *oblicua*. Veremos dos casos especiales de proyecciones oblicuas, la proyección *caballera* y la proyección de *gabinete*.

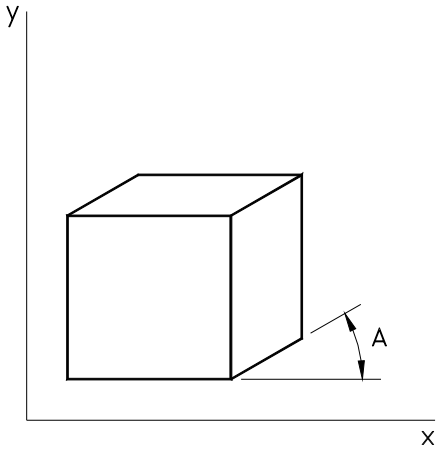
Veamos primero la perspectiva caballera. Para ello, supongamos que estamos observando un objeto cuyos lados son paralelos a los ejes de coordenadas. El plano de visualización será paralelo a la cara frontal (paralelo al plano xy). Para la perspectiva caballera, la dirección de proyección esta inclinada de tal forma que los puntos con coordenadas z positivas serán proyectados abajo y a la izquierda en el plano de visualización. Los puntos con coordenadas z negativas serán proyectados arriba a la derecha. El ángulo del eje z proyectado puede ser el que queramos, pero la distancia a la que el punto está situado en la dirección z proyectada debe ser igual a la distancia real del punto al plano de visualización.

Esta restricción facilita la tarea del delineante. Sin embargo, el resultado es un objeto que parece alargarse a lo largo del eje *z*.



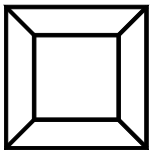
vPerspectiva caballera

Una alternativa, que sigue siendo fácil de construir, es llevar sólo la mitad de la distancia real sobre el eje *z* proyectado. Esta es la llamada proyección de gabinete.



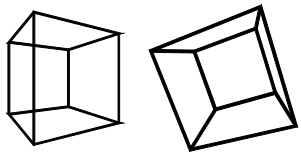
Perspectiva de gabinete

Las proyecciones en perspectiva pueden ser clasificadas en tres grandes grupos según el número de puntos de fuga: uno, dos o tres. La proyección con un punto de fuga se presenta cuando una de las caras de un objeto rectangular es paralela al plano de visualización.



Perspectiva con un punto de fuga de un cubo

Una perspectiva con dos puntos de fuga se presenta cuando un conjunto de aristas son paralelas al plano de visualización, pero ninguna de las caras lo es. Por último, la perspectiva con tres puntos de fuga se da cuando ninguna de las aristas es paralela al plano de visualización.



Perspectiva con dos y tres puntos de fuga de un cubo

APLICACIONES

LISTADO 4 - REPRESENTACIÓN DE OBJETOS TRIDIMENSIONALES

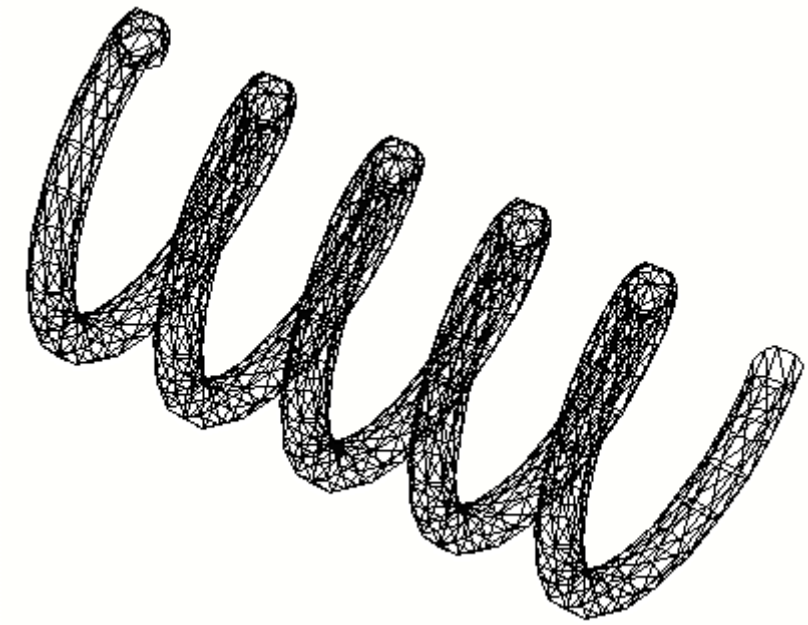
En el siguiente listado se presenta un programa que lee desde un fichero, suministrado por el usuario, la definición de un objeto. En la primera línea de este fichero se indican el número de puntos, elementos y vértices que componen el objeto. En las siguientes líneas se indican las coordenadas de los puntos y los índices de los vértices que constituyen cada una de las caras poligonales del objeto.

```
DECLARE FUNCTION MIN! (A!, B!)
DECLARE FUNCTION MAX! (A!, B!)
DECLARE SUB TRAS (XC!, YC!, ZC!)
DECLARE SUB DIRP (XP!, YP!, ZP!)
DECLARE SUB MMAT (A!(), B!())
DECLARE SUB PROYECTA (P!(), Q!())
DECLARE SUB ROTA (EJE$, A!)

DIM SHARED MATP!(4, 4): DIM SHARED MATR!(4, 4): DIM SHARED MATT!(4, 4)
DIM P1!(4), P2!(4), Q1!(4), Q2!(4)

MATP(1, 1) = 1: MATP(1, 2) = 0: MATP(1, 3) = 0: MATP(1, 4) = 0
MATP(2, 1) = 0: MATP(2, 2) = 1: MATP(2, 3) = 0: MATP(2, 4) = 0
MATP(3, 1) = 0: MATP(3, 2) = 0: MATP(3, 3) = 1: MATP(3, 4) = 0
MATP(4, 1) = 0: MATP(4, 2) = 0: MATP(4, 3) = 0: MATP(4, 4) = 1

CLS: FILES "*.geo": PRINT
INPUT "FICHERO: "; FICH$: OPEN FICH$ FOR INPUT AS #1

INPUT #1, NP, NE, DUMMY: DIM XMAT!(NP), YMAT!(NP), ZMAT!(NP)

XMIN = 999999!: YMIN = 999999!: ZMIN = 999999!
XMAX = -XMIN: YMAX = -YMIN: ZMAX = -ZMIN

FOR I = 1 TO NP
   INPUT #1, XMAT!(I), YMAT!(I), ZMAT!(I)

   XMIN = MIN(XMIN, XMAT(I))
   YMIN = MIN(YMIN, YMAT(I))
   ZMIN = MIN(ZMIN, ZMAT(I))

   XMAX = MAX(XMAX, XMAT(I))
   YMAX = MAX(YMAX, YMAT(I))
   ZMAX = MAX(ZMAX, ZMAT(I))
NEXT I

ROTA "Z", 30: MMAT MATR(), MATP()
ROTA "X", 30: MMAT MATR(), MATP()
DIRP 0, 0, 1: MMAT MATT(), MATP()

P1(1) = XMIN: P1(2) = YMIN: P1(3) = ZMAX: P1(4) = 1
PROYECTA P1(), Q1()

P2(1) = XMAX: P2(2) = YMAX: P2(3) = ZMIN: P2(4) = 1
PROYECTA P2(), Q2()

SCREEN 12: BORDER = 2

XS = ABS(Q2(1) - Q1(1)): YS = ABS(Q2(2) - Q1(2))
XC = .5 * (Q1(1) + Q2(1)): YC = .5 * (Q1(2) + Q2(2))

IF (XS / 4 > YS / 3) THEN
   SXMIN = Q1(1): SXMAX = Q2(1)
   SYMIN = YC - .5 * XS * 3 / 4
   SYMAX = YC + .5 * XS * 3 / 4
ELSE
   SXMIN = XC - .5 * YS * 4 / 3
   SXMAX = XC + .5 * YS * 4 / 3
   SYMIN = Q1(2): SYMAX = Q2(2)
END IF

WINDOW (BORDER * SXMIN, BORDER * SYMAX)-(BORDER * SXMAX, BORDER * SYMIN)
```

```
   FOR I = 1 TO NE
      INPUT #1, NV, NI: NF = NI

      FOR K = 2 TO NV
         INPUT #1, NJ

         P1(1) = XMAT(NI): P1(2) = YMAT(NI): P1(3) = ZMAT(NI): P1(4) = 1
         P2(1) = XMAT(NJ): P2(2) = YMAT(NJ): P2(3) = ZMAT(NJ): P2(4) = 1

         PROYECTA P1(), Q1(): PROYECTA P2(), Q2()
         LINE (Q1(1) + XC, Q1(2) + YC)-(Q2(1) + XC, Q2(2) + YC)
         NI = NJ
      NEXT K

      P1(1) = XMAT(NI): P1(2) = YMAT(NI): P1(3) = ZMAT(NI): P1(4) = 1
      P2(1) = XMAT(NF): P2(2) = YMAT(NF): P2(3) = ZMAT(NF): P2(4) = 1

      PROYECTA P1(), Q1(): PROYECTA P2(), Q2()
      LINE (Q1(1) + XC, Q1(2) + YC)-(Q2(1) + XC, Q2(2) + YC)
   NEXT I
   SLEEP: CLOSE

   SUB DIRP (XP!, YP!, ZP!)
   MATT(1, 1) = 1: MATT(1, 2) = 0: MATT(1, 3) = 0: MATT(1, 4) = 0
   MATT(2, 1) = 0: MATT(2, 2) = 1: MATT(2, 3) = 0: MATT(2, 4) = 0
   MATT(3, 1) = -XP / ZP: MATT(3, 2) = -YP / ZP: MATT(3, 3) = 0: MATT(3, 4) = 0
   MATT(4, 1) = 0: MATT(4, 2) = 0: MATT(4, 3) = 0: MATT(4, 4) = 1
   END SUB

   FUNCTION MAX (A, B)
   IF (A > B) THEN
      MAX = A
   ELSE
      MAX = B
   END IF
   END FUNCTION

   FUNCTION MIN (A, B)
   IF (A < B) THEN
      MIN = A
   ELSE
      MIN = B
   END IF
   END FUNCTION

   SUB MMAT (A(), B())
   DIM C!(4, 4)

   FOR I = 1 TO 4
      FOR J = 1 TO 4
         C(I, J) = 0
         FOR K = 1 TO 4
            C(I, J) = C(I, J) + A(I, K) * B(K, J)
         NEXT K
      NEXT J
   NEXT I

   FOR I = 1 TO 4
      FOR J = 1 TO 4
         B(I, J) = C(I, J)
      NEXT J
   NEXT I
   END SUB

   SUB PROYECTA (P(), Q())
   FOR I = 1 TO 4
      Q(I) = 0
      FOR J = 1 TO 4
         Q(I) = Q(I) + P(J) * MATP(I, J)
      NEXT J
   NEXT I

   Q(1) = Q(1) / Q(4): Q(2) = Q(2) / Q(4): Q(3) = Q(3) / Q(4)
   END SUB
```

```
SUB ROTA (EJE$, A!)
CONST PI = 3.141592654#
ANG = A * PI / 180!

IF (EJE$ = "X") THEN
   MATR(1, 1) = 1: MATR(1, 2) = 0: MATR(1, 3) = 0: MATR(1, 4) = 0
   MATR(2, 1) = 0: MATR(2, 2) = COS(ANG): MATR(2, 3) = SIN(ANG): MATR(2, 4) = 0
   MATR(3, 1) = 0: MATR(3, 2) = -SIN(ANG): MATR(3, 3) = COS(ANG): MATR(3, 4) = 0
   MATR(4, 1) = 0: MATR(4, 2) = 0: MATR(4, 3) = 0: MATR(4, 4) = 1
END IF

IF (EJE$ = "Y") THEN
   MATR(1, 1) = COS(ANG): MATR(1, 2) = 0: MATR(1, 3) = -SIN(ANG): MATR(1, 4) = 0
   MATR(2, 1) = 0: MATR(2, 2) = 1: MATR(2, 3) = 0: MATR(2, 4) = 0
   MATR(3, 1) = SIN(ANG): MATR(3, 2) = 0: MATR(3, 3) = COS(ANG): MATR(3, 4) = 0
   MATR(4, 1) = 0: MATR(4, 2) = 0: MATR(4, 3) = 0: MATR(4, 4) = 1
END IF

IF (EJE$ = "Z") THEN
   MATR(1, 1) = COS(ANG): MATR(1, 2) = SIN(ANG): MATR(1, 3) = 0: MATR(1, 4) = 0
   MATR(2, 1) = -SIN(ANG): MATR(2, 2) = COS(ANG): MATR(2, 3) = 0: MATR(2, 4) = 0
   MATR(3, 1) = 0: MATR(3, 2) = 0: MATR(3, 3) = 1: MATR(3, 4) = 0
   MATR(4, 1) = 0: MATR(4, 2) = 0: MATR(4, 3) = 0: MATR(4, 4) = 1
END IF
END SUB
```

Resultado de la ejecución:

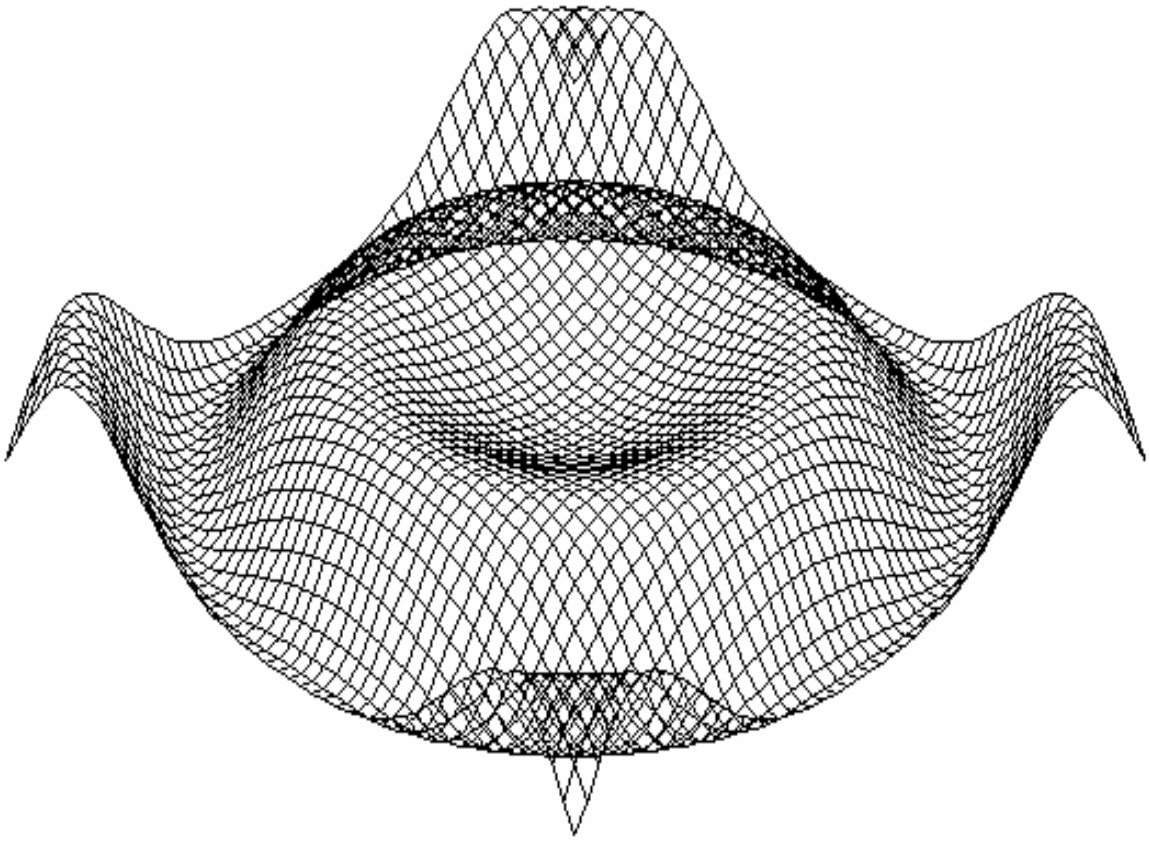


LISTADO 5 - REPRESENTACIÓN DE SUPERFICIES Z=F(X,Y)

En este listado se presenta un pequeño programa que permite representar superficies del tipo $z=f(x, y)$ definiendo los valores máximo y mínimo de las variables, así como el número de segmentos que se emplearán para representar la superficie.

```
DECLARE SUB PROYECTA (P!(), Q!())
DECLARE FUNCTION F! (X!, Y!)
DIM P1(3), Q1(3), P2(3), Q2(3)

XMIN = -5: XMAX = 5
YMIN = -5: YMAX = 5
ZMIN = -5: ZMAX = 5

NX = 50: NY = 50
SX = (XMAX - XMIN) / NX
SY = (YMAX - YMIN) / NY

P1(1) = XMAX: P1(2) = YMIN: P1(3) = 0: PROYECTA P1(), Q1()
XpMIN = Q1(1)
P1(1) = XMIN: P1(2) = YMAX: P1(3) = 0: PROYECTA P1(), Q1()
XpMAX = Q1(1)
P1(1) = XMIN: P1(2) = YMIN: P1(3) = ZMAX: PROYECTA P1(), Q1()
YpMAX = Q1(2)
P1(1) = XMAX: P1(2) = YMAX: P1(3) = ZMIN: PROYECTA P1(), Q1()
YpMIN = Q1(2)

XS = XpMAX - XpMIN
YS = YpMAX - YpMIN
XC = .5 * (XpMAX + XpMIN)
YC = .5 * (YpMAX + YpMIN)

IF (XS / 4 > YS / 3) THEN
   SXMIN = XpMIN: SXMAX = XpMAX
   SYMIN = YC - .5 * XS * 3 / 4
   SYMAX = YC + .5 * XS * 3 / 4
ELSE
   SXMIN = XC - .5 * YS * 4 / 3
   SXMAX = XC + .5 * YS * 4 / 3
   SYMIN = YpMIN: SYMAX = YpMAX
END IF

SCREEN 12: BORDER = 1
WINDOW (BORDER * SXMIN, BORDER * SYMAX)-(BORDER * SXMAX, BORDER * SYMIN)

P1(2) = YMIN
FOR I = 0 TO NX
   P1(1) = XMIN
   FOR J = 0 TO NY
      P1(3) = F(P1(1), P1(2))
      PROYECTA P1(), Q1()

      IF (I <> NX) THEN
         P2(1) = P1(1): P2(2) = P1(2) + SY: P2(3) = F(P2(1), P2(2))
         PROYECTA P2(), Q2(): LINE (Q1(1), Q1(2))-(Q2(1), Q2(2))
      END IF

      IF (J <> NY) THEN
         P2(1) = P1(1) + SX: P2(2) = P1(2): P2(3) = F(P2(1), P2(2))
         PROYECTA P2(), Q2(): LINE (Q1(1), Q1(2))-(Q2(1), Q2(2))
      END IF

      P1(1) = P1(1) + SX
   NEXT J
   P1(2) = P1(2) + SY
NEXT I

FUNCTION F! (X!, Y!)
   F = SIN((X * X + Y * Y) / 5)
END FUNCTION

SUB PROYECTA (P(), Q())
   CONST C30 = .8660254037844387#
   CONST S30 = .5#
   Q(1) = (P(2) - P(1)) * C30
   Q(2) = -(P(1) + P(2)) * S30 + P(3)
   Q(3) = P(3)
END SUB
```

Resultado de la ejecución:

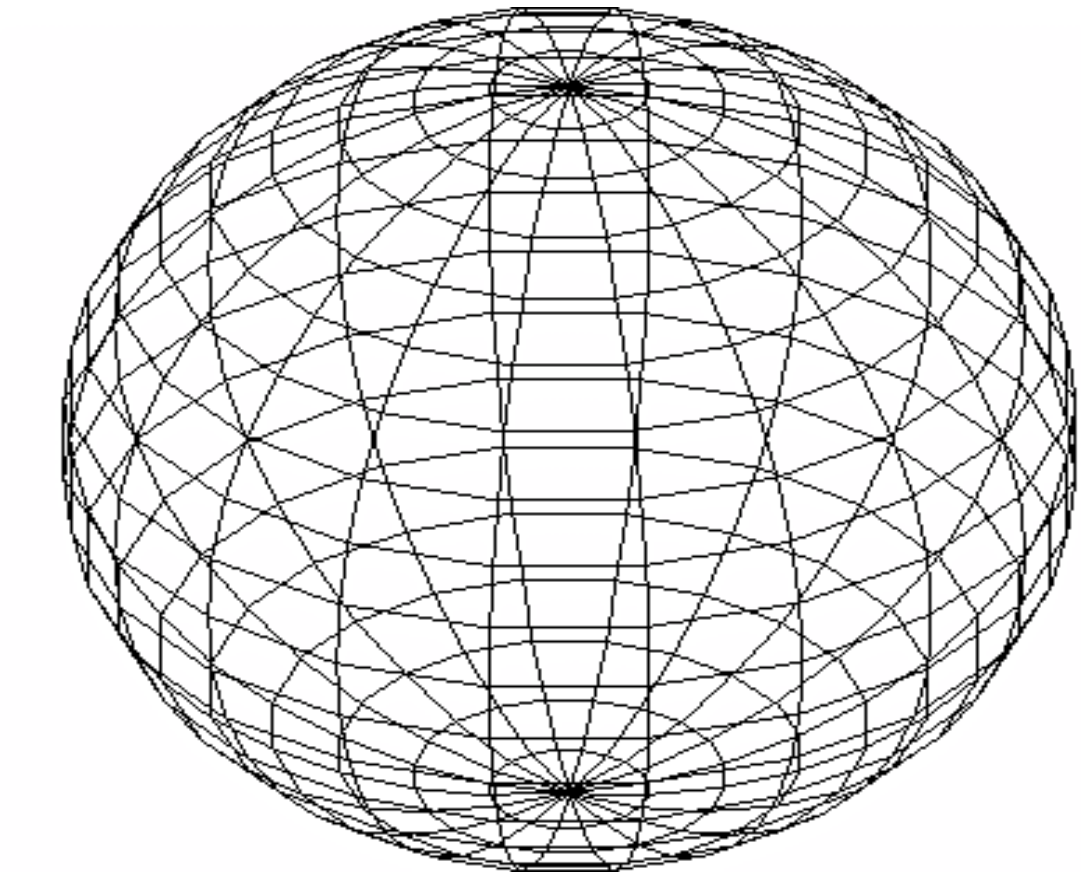


LISTADO 6 - REPRESENTACIÓN DE SUPERFICIES PARAMÉTRICAS

En este listado se presenta un pequeño programa que permite representar superficies paramétricas definiendo los valores máximo y mínimo de las variables y los parámetros, así como el número de segmentos que se emplearán para representar la superficie.

```
DECLARE SUB PROYECTA (P!(), Q!())
DECLARE FUNCTION X! (u!, v!)
DECLARE FUNCTION Y! (u!, v!)
DECLARE FUNCTION Z! (u!, v!)
DIM P1(3), Q1(3), P2(3), Q2(3)

CONST PI = 3.141592653589793#
XMIN = -5: XMAX = 5: YMIN = -5: YMAX = 5: ZMIN = -5: ZMAX = 5
uMin = -PI: uMAX = PI: vMin = -PI / 2: vMAX = PI / 2
Nu = 20: Nv = 20
Su = (uMAX - uMin) / Nu
Sv = (vMAX - vMin) / Nv

P1(1) = XMAX: P1(2) = YMIN: P1(3) = 0: PROYECTA P1(), Q1()
XpMIN = Q1(1)
P1(1) = XMIN: P1(2) = YMAX: P1(3) = 0: PROYECTA P1(), Q1()
XpMAX = Q1(1)
P1(1) = XMIN: P1(2) = YMIN: P1(3) = ZMAX: PROYECTA P1(), Q1()
YpMAX = Q1(2)
P1(1) = XMAX: P1(2) = YMAX: P1(3) = ZMIN: PROYECTA P1(), Q1()
YpMIN = Q1(2)

XS = XpMAX - XpMIN: YS = YpMAX - YpMIN
XC = .5 * (XpMAX + XpMIN): YC = .5 * (YpMAX + YpMIN)

IF (XS / 4 > YS / 3) THEN
   SXMIN = XpMIN: SXMAX = XpMAX
   SYMIN = YC - .5 * XS * 3 / 4
   SYMAX = YC + .5 * XS * 3 / 4
ELSE
   SXMIN = XC - .5 * YS * 4 / 3
   SXMAX = XC + .5 * YS * 4 / 3
   SYMIN = YpMIN: SYMAX = YpMAX
END IF

SCREEN 12: BORDER=1
WINDOW (BORDER * SXMIN, BORDER * SYMAX)-(BORDER * SXMAX, BORDER * SYMIN)

v = vMin
FOR I = 0 TO Nu
   u = uMin
   FOR J = 0 TO Nv
      P1(1) = X(u, v): P1(2) = Y(u, v): P1(3) = Z(u, v)
      PROYECTA P1(), Q1()
      IF (I <> Nu) THEN
         P2(1) = X(u, v + Sv): P2(2) = Y(u, v + Sv): P2(3) = Z(u, v + Sv)
         PROYECTA P2(), Q2(): LINE (Q1(1), Q1(2))-(Q2(1), Q2(2))
      END IF
      IF (J <> Nv) THEN
         P2(1) = X(u + Su, v): P2(2) = Y(u + Su, v): P2(3) = Z(u + Su, v)
         PROYECTA P2(), Q2(): LINE (Q1(1), Q1(2))-(Q2(1), Q2(2))
      END IF
      u = u + Su
   NEXT J
   v = v + Sv
NEXT I

SUB PROYECTA (P(), Q())
   CONST C30 = .8660254037844387#
   CONST S30 = .5#
   Q(1) = (P(2) - P(1)) * C30
   Q(2) = -(P(1) + P(2)) * S30 + P(3)
   Q(3) = P(3)
END SUB

FUNCTION X! (u!, v!)
   X = 5 * COS(v) * COS(u)
END FUNCTION

FUNCTION Y! (u!, v!)
   Y = 5 * COS(v) * SIN(u)
END FUNCTION

FUNCTION Z! (u!, v!)
   Z = 5 * SIN(v)
END FUNCTION
```

Resultado de la ejecución:



OCULTAMIENTO DE LÍNEAS Y SUPERFICIES

INTRODUCCIÓN

Hasta el momento hemos aprendido como construir y proyectar objetos tridimensionales, pero siempre se ven todas las partes del objeto. Esto da a nuestros dibujos una cierta calidad de transparencia. Las figuras que se obtienen de esta forma reciben el nombre de *modelos alámbricos* ya que sólo presentan los contornos de objetos supuestamente sólidos. Los objetos complejos se pueden convertir rápidamente en un amasijo de segmentos sin sentido. Podría ser difícil identificar que líneas pertenecen a la parte frontal de objeto y cuales están situadas detrás. En este capítulo intentaremos eliminar aquellas líneas que estarían ocultas por alguna de las partes del objeto. Si podemos asignar esta tarea al ordenador, entonces seríamos libres de construir el modelo completo de un objeto (frente, dorso, interior e exterior) y ser capaces todavía de representarlo tal como se vería en la realidad.

Este problema no es tan sencillo como cabría pensar en un principio. De echo, es el más complejo en términos de programación de todos los que veamos en este texto. Lo que naturaleza hace con facilidad (y una gran cantidad de procesamiento paralelo) debemos hacerlo nosotros con un número elevado de cálculos. Existen varias soluciones para el problema de las caras y líneas ocultas. Aquí veremos sólo una de ellas que no es ni la más elegante ni la más eficiente, pero que plantea el problema de una forma directa.

ELIMINACIÓN DE CARAS OCULTAS

La eliminación de las líneas ocultas de un objeto puede ser un proceso bastante costoso, por lo tanto esto nos obliga a aplicar sencillas comprobaciones para simplificar el problema todo lo que podamos antes de realizar un estudio pormenorizado. Existe una comprobación muy sencilla que eliminará la mayoría de las caras que no se pueden ver. Este test identifica las caras que mirán en la dirección opuesta al observador. Estas son las que constituyen la parte trasera del objeto y no pueden verse porque la masa del objeto se encuentra entre ellas y el observador. Esto, sin embargo, no resuelve la totalidad del problema de la superficies ocultas ya que es posible que la parte delantera del objeto esté oculta por un segundo objeto o por cualquier otra parte del mismo objeto. A pesar de ello, esta comprobación elimina aproximadamente la mitad de las superficies que deben ser verificadas y por tanto simplifica el problema.

Debemos empezar nuestro estudio destacando que sólo las caras poligonales serán motivo de comprobación. Las líneas de por si no pueden ocultar nada, pero aunque podrían estar tapadas, sólo se encuentran como aristas de las caras que definen el objeto. Debido a esto, los polígono son generalmente suficientes para la mayoría de los modelos.

Los polígonos tienen dos caras, la frontal y la dorsal, igual que una hoja de papel. Podemos imaginar nuestros polígono con una cara pintada de color claro y otra pintada de color oscuro. Pero dadas las coordenadas de los vértices de un polígono, ¿cómo podemos saber cuál es cada cara? Fácil, podemos decir que la cara frontal es aquella que se dibuja en sentido horario. Si movemos el punto de vista al lado contrario, de tal forma que la cara oscura sea visible, entonces este mismo polígono parece que se dibuja en sentido antihorario.

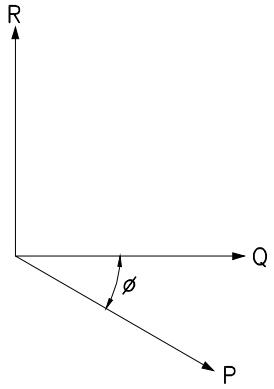
Un poco de matemáticas nos permitirá averiguar la dirección hacia la que mira cada una de las caras del objeto (la normal al plano en el que está contenido el polígono). La operación que necesitamos es el *producto vectorial*. El producto

vectorial de dos vectores es otro vector que tiene por módulo el producto de las longitudes de los vectores por el seno del ángulo que forman y, lo más importante para nosotros, una dirección perpendicular al plano que definen estos dos vectores.

$$\left[R_x \;\; R_y \;\; R_z\right] = \left[P_x \;\; P_y \;\; P_z\right] \times \left[Q_x \;\; Q_y \;\; Q_z\right]$$

donde

$$R_x = P_y Q_z - P_z Q_y$$
$$R_y = P_z Q_x - P_x Q_z$$
$$R_z = P_x Q_y - P_y Q_x$$



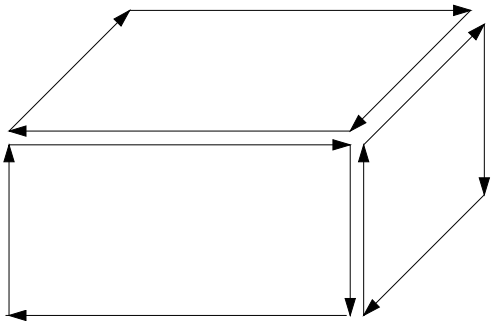
Producto vectorial de dos Vectores

Si tenemos en cuenta, en cualquier pareja de lados de un polígono definen dos vectores en el plano del polígono, entonces el producto vectorial de dos lados del polígono nos da un vector que apunta hacia afuera desde la cara del polígono. ¿Apuntará este vector hacia afuera desde la cara vista o desde la cara oculta del polígono? Eso depende del ángulo que formen los dos lados escogidos, si es cóncavo o convexo.

Supongamos que tenemos dos lados adyacentes que no están sobre la misma línea y que se encuentran en un vértice convexo, es decir, un vértice donde los dos lados se unen formando un ángulo convexo. El producto vectorial nos dará un vector que apunta hacia afuera desde la cara oculta.

Si ahora establecemos la regla de que todos los objetos que construyamos tengan sus caras frontales apuntando hacia el exterior, entonces las caras ocultas estarán en contacto con el material del interior. Esto significa que cuando veamos el objeto desde el exterior, aparecerá dibujado en sentido horario.

Si un polígono es visible, podremos ver su cara frontal y no su cara dorsal, que apuntará en dirección opuesta al observador. Ya que es posible calcular un producto vectorial que nos dice cual es la cara dorsal del polígono, veremos la cara frontal cuando este vector apunte en dirección opuesta al observador. En otro caso, la cara del objeto no es visible y debe ser eliminada.



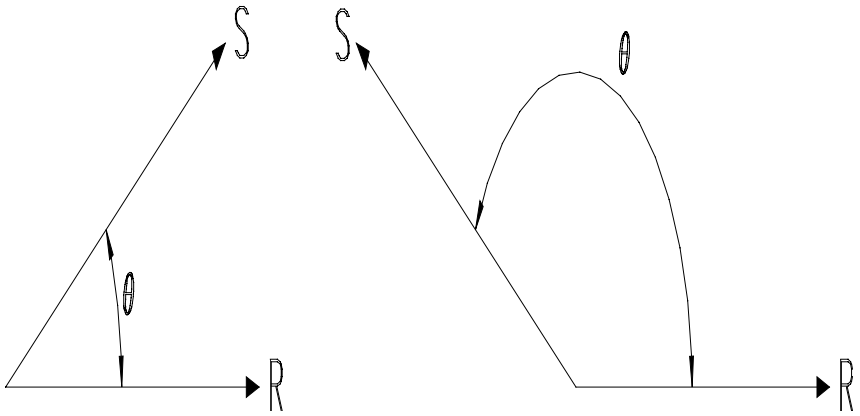
Las caras exteriores de un objeto se dibujan en sentido horario

¿Cómo podemos saber si un vector apunta hacia el observador o no? Este es un problema de comparación de vectores, el vector normal a la superficie y el vector director de la proyección (después de realizar todas las transformaciones del objeto). Para las proyecciones en perspectiva, se puede formar un vector que

una el centro de proyección y un punto del polígono. Para una proyección paralela, la dirección de proyección podría servir siempre que apunte desde el objeto hacia el observador. Debemos tener la precaución de indicar de esta forma la dirección cuando empleemos las proyecciones paralelas. Veremos más tarde que la normal al plano de visualización debe apuntar desde el observador hacia el objeto, en dirección contraria a la de proyección.

Tenemos dos vectores (R y S) y queremos comparar sus direcciones. Para hacer esto empleamos su producto escalar

$$a = R \cdot S = |R||S|\cos\theta$$



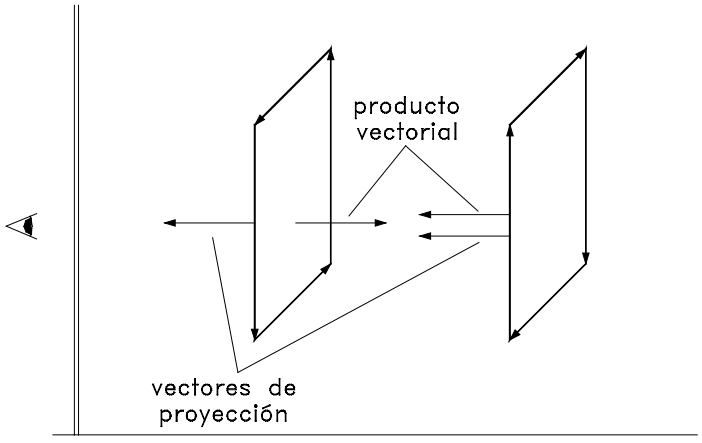
cos θ > 0 cos θ < 0

El valor del coseno es importante porque si los vectores tienen más o menos la misma dirección ($0 \le \theta < \pi/2$) entonces es positivo y el producto escalar también lo es; pero si las direcciones son opuestas ($\pi/2 < \theta \le \pi$), el coseno es negativo y también el producto escalar.

Para determinar si un polígono es una cara frontal o dorsal del objeto, utilizamos el producto vectorial para determinar su normal y luego calculamos el producto escalar con el vector de proyección. Un valor negativo significa que la cara dorsal mira en dirección opuesta a la del observador y, por tanto, la cara frontal puede ser observada. La expresión para determinar el valor del producto escalar es

$$a = R \cdot S = \begin{bmatrix} R_x & R_y & R_z \end{bmatrix} \cdot \begin{bmatrix} S_x & S_y & S_z \end{bmatrix} = R_x S_x + R_y S_y + R_z S_z$$

Por lo tanto, la comprobación de ocultamiento de la cara se reduce a la determinación del signo de a.



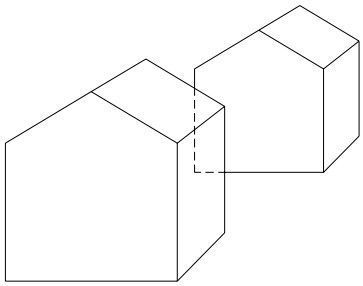
El plano de la izquierda será visible (a<0) mientras que el derecho está oculto (a>0)

Debemos tener cuidado a la hora de elegir los lados del polígono P y Q. Estos vectores deben formar un ángulo convexo. ¿Cómo encontramos este ángulo convexo? Fácil, seleccionamos un extremo del polígono como el punto que está más a la derecha o más a la izquierda o más arriba. Con los dos puntos anterior y posterior al escogido definimos los dos vectores que necesitamos, comprobando que los tres puntos no están alineados. Si ocurre que el tercer punto está alineado con los dos primeros, se escoge el siguiente vértice del polígono y se intenta de nuevo. Se usamos todos los vértices del polígono sin éxito, entonces se trata de una línea simplemente y puede ser descartado sin más.

Todas estas transformaciones deben realizarse justo antes de proyectar el objeto sobre el plano de visualización.

ALGORITMO DEL PINTOR

Aunque el algoritmo que hemos visto en el apartado anterior puede eliminar muchas de las caras y líneas ocultas, no resuelve completamente el problema. Si tenemos dos objetos separados, entonces alguna de la caras vistas de uno de ellos podría ocultar parcialmente alguna de las caras vistas del otro.



Líneas ocultas por caras vistas

El algoritmo que veremos a continuación resuelve este problema para superficies convexas en dispositivos *raster*. Para ello se hace uso de algunas propiedades de este tipo de dispositivos. Sólo tendremos en cuenta polígonos con su interior relleno.

El algoritmo recibe su nombre por la forma en que un pintor crea un óleo. El artista empieza pintando el fondo para luego dibujar los objetos del frente. No es necesario borrar determinadas partes del fondo, el artista simplemente pinta encima de ellas. La nueva pintura cubre a la vieja de tal forma que sólo es visible la última capa de pintura.

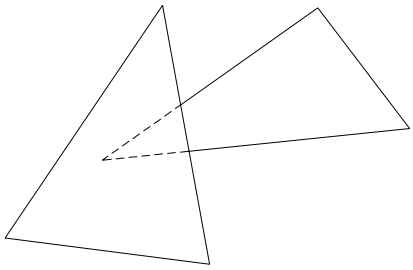
Los dispositivos raster tienen esta misma propiedad. Un polígono relleno se representa cambiando el color de los pixel correspondientes a su interior. Si dibujamos un segundo polígono *encima* del primero, algunos de estos mismos puntos cambiarán para reflejar el interior del segundo polígono. Esto implica que primero debemos dibujar los polígonos que están más lejos del observador, dejando para el final los que están más cerca del observador. De esta forma, las superficies se *cubren* si ordenamos correctamente los polígonos antes de dibujarlos.

ELIMINACIÓN DE LÍNEAS OCULTAS

En el apartado anterior trabajamos con polígonos rellenos para eliminar las caras ocultas. Veremos ahora como podemos eliminar los lados ocultos de los polígonos cuando no podemos recurrir al algoritmo del pintor.

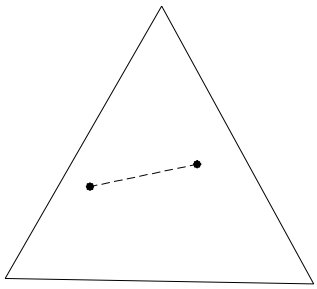
Así, examinaremos todos y cada uno de los lados de los polígonos que constituyen el objeto antes de dibujarlos en pantalla. Compararemos cada lado con todos los triángulos (previamente se habrán descompuesto los polígonos en

triángulos para facilitar los cálculos) que están situados enfrente de él para comprobar si está parcial o totalmente oculto.



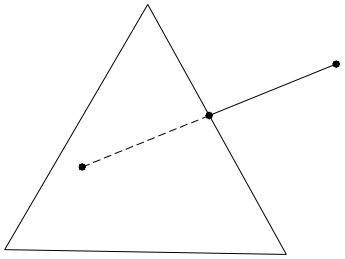
Segmentos eliminados por ocultamiento

Un triángulo puede ocultar a un segmento de varias formas diferentes. Si los dos extremos del segmento están dentro del triángulo, entonces todo el segmento es invisible.



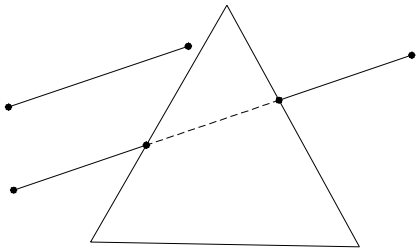
Si los dos extremos están en el interior del triángulo el segmento es invisible

Si un extremo está dentro del triángulo y otro en el exterior sólo una parte del segmento será invisible. En este caso debemos comprobar los tres lados del triángulo para determinar la intersección del segmento con alguno de ellos. El punto de intersección divide el segmento en dos partes. La parte que queda dentro del triángulo será invisible, mientras que la otra porción es todavía visible y debe comprobarse si la oculta algún otro triángulo.



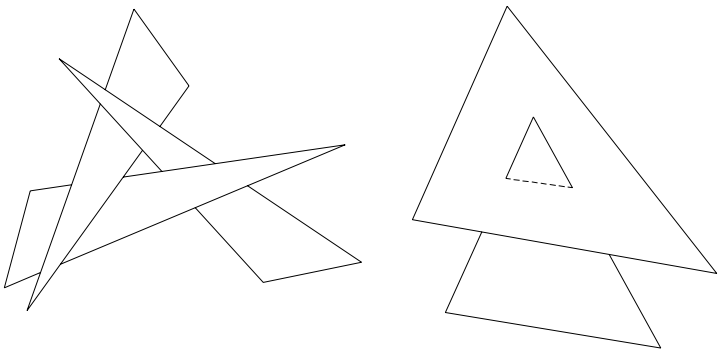
Si un extremo está dentro y otro fuera del triángulo, sólo una porción es invisible

Si los dos puntos están fuera del triángulo, existen dos posibilidades. En un caso, el segmento está totalmente fuera del triángulo. En este caso, el segmento es visible en su totalidad (a no ser que lo oculte otro triángulo). En el segundo caso, el segmento entra y sale del triángulo, es decir, corta a dos de los lados del triángulo, quedando dividido en tres partes. La porción central estará oculta, mientras que las otras dos deben ser comprobadas con el resto de triángulos.



Los dos extremos están fuera del triángulo, puede haber o no intersección

A pesar de todas las comprobaciones realizadas existen caso todavía que no pueden ser tratados con estos algoritmos.



Casos que no pueden ser manejados con las técnicas presentadas

APLICACIONES

GNUPLOT: UN PROGRAMA PARA REPRESENTACIÓN DE FUNCIONES

GNUPLOT es un programa interactivo de representación de funciones. Puede ser ejecutado en diferentes plataformas (UNIX, Windows, MSDOS, VMS, etc.) y es gratis, a pesar de estar sometido a las leyes de propiedad intelectual. Soporta, además, muchos tipos diferentes de terminales, trazadores e impresoras (incluyendo muchos dispositivos en color). Maneja tanto curvas (2 dimensiones) como superficies (3 dimensiones).

Las superficies pueden ser presentadas mediante una malla cuadrada que se ajusta a la función especificada en el espacio tridimensional de coordenadas, o como un mapa de isolíneas situado en el plano xy. Para los dibujos 2D existen múltiples tipos de diseños incluyendo líneas, puntos, líneas con puntos, barras de error e impulsos (puros gráficos de barras). Los gráficos pueden ser etiquetados por el usuario. El programa incluye en la mayoría de las plataformas la capacidad de editar las líneas de comandos.

Este programa distingue entre mayúsculas y minúsculas (los comandos y nombre de funciones escritos en minúsculas no son los mismos que los escritos con mayúsculas). Todos los comandos pueden ser abreviados, mientras que la abreviación no sea ambigua. En una misma línea de comando pueden aparecer varios comandos, siempre que este separados por punto y coma (;). Las cadenas se presentan entre comillas, que pueden ser dobles (“) o sencillas (‘). Por ejemplo

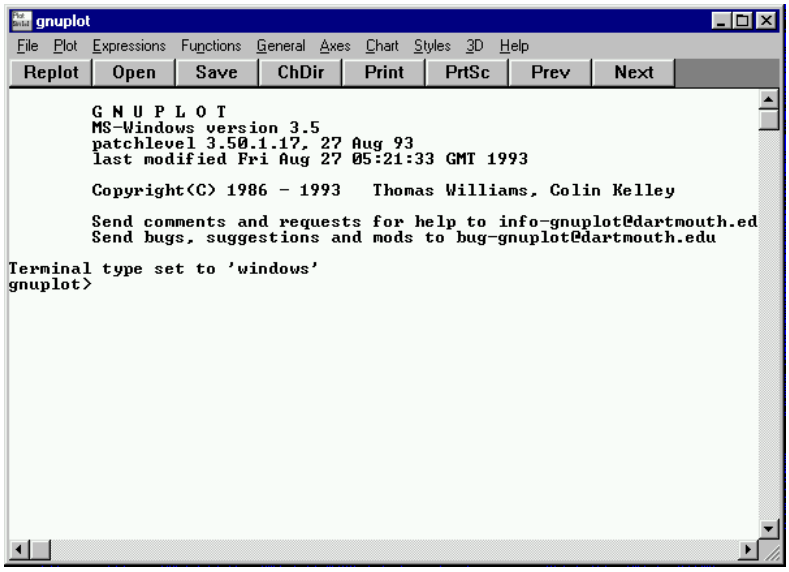
```
load "filename"
cd 'dir'
```

Los argumentos que se pasan en la línea de comandos se supone que son nombres de ficheros que contienen comandos de GNUPLOT, con la excepción de los argumentos estándar X11, que se procesan en primer lugar. Cada fichero se carga con el comando `load`, según el orden establecido. GNUPLOT termina después de que el último fichero ha sido procesado. Cuando no se indica ningún fichero para ser cargado, entonces GNUPLOT arranca en el modo interactivo.

Los comandos se pueden extender a lo largo de varias líneas, siempre que cada una de las líneas termina con una barra invertida (\). La barra invertida debe ser el último de los caracteres de cada línea. El efecto es el mismo que si no estuviesen allí la barra invertida y el salto de línea. Es decir, no se incluye ningún espacio en blanco, ni se terminan los comentarios. Por lo tanto, comentar una línea que continua en la siguiente supone eliminar todo el comando (ver `comment`)

En este capítulo, las llaves ({}) indican argumentos opcionales en muchos de los comandos, y una barra vertical (|) separa opciones mutuamente excluyentes. Las palabras claves de GNUPLOT se indican en **negrita**. Los símbolos (<>) se utilizan para indicar elementos reemplazables.

En la versión Windows, como se aprecia en la siguiente figura, los diferentes comandos y opciones se han implementado mediante menús desplegables. A medida que se van escogiendo las opciones desde el menú correspondiente, estas aparecen en la línea de comandos. Es posible, también, teclear directamente el comando deseado, tal como se haría en el resto de plataformas.



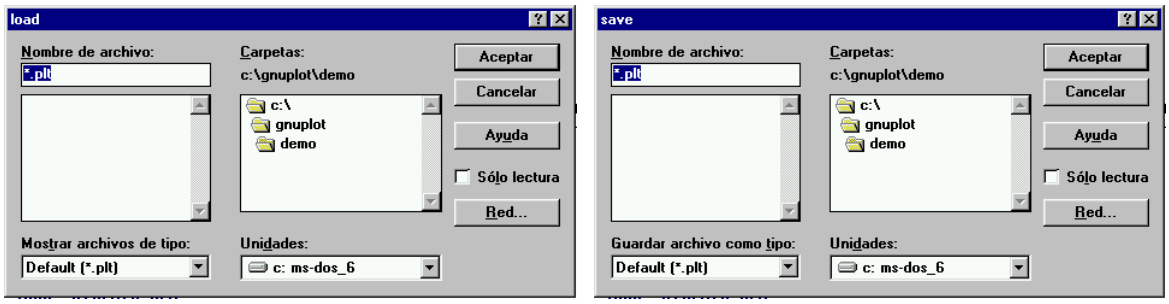
Pantalla principal de la versión para Windows de GNUPlot

Algunos comandos, de uso más frecuente, se presentan en una botonera especial que facilita el acceso a ellos.

Botonera de acceso rápido a los comandos más frecuentes

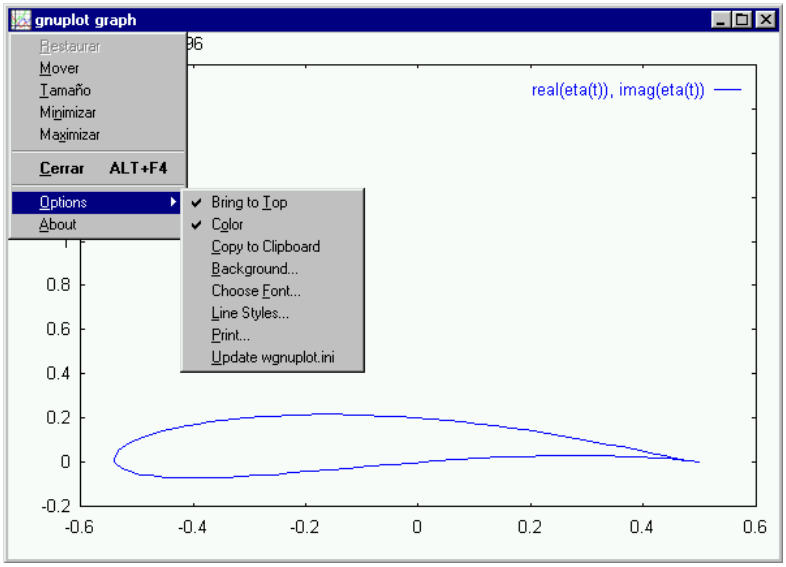
`replot`: redibuja el último gráfico creado con `plot` o `splot`
`open`: despliega el diálogo de lectura de ficheros de comandos
`save`: despliega el diálogo de escritura de ficheros de comandos
`chdir`: cambia el directorio de trabajo
`print`: imprime el último dibujo en la impresora seleccionada
`prtsc`: copia la ventana gráfica en el dispositivo seleccionado
`prev`: muestra el anterior comando de la lista
`next`: muestra el siguiente comando de la lista

Así, las opciones de leer y grabar ficheros hacen uso de las ventanas de diálogo tan frecuentes en Windows y que son comunes a todas las aplicaciones que se ejecutan en este entorno.

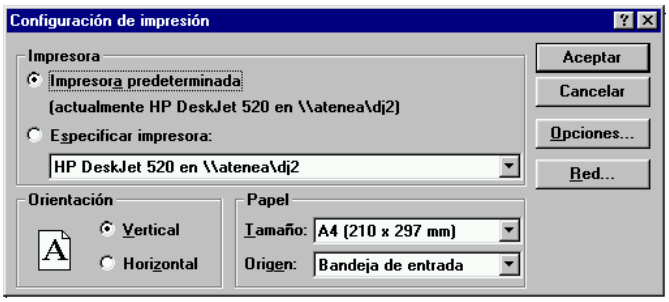


Diálogo para la lectura y escritura de ficheros de comandos

También, las posibilidades de imprimir los gráficos generados con GNUPLOT mejoran sensiblemente gracias a la utilización de los dispositivos de impresión instalados en Windows. De este modo, la ventana donde se presentan los gráficos dispone de un menú de opciones que permite al usuario modificar de forma interactiva distintas opciones (tipo de letra, tipos de líneas, etc), así como crear una copia en papel del gráfico creado. No debemos olvidar tampoco las posibilidades que ofrece Windows para incluir los gráficos generados por GNUPLOT en otras aplicaciones (procesadores de texto, hojas de cálculo, etc.) que se ejecutan en este sistema operativo.



Menú de opciones de la venta gráfica de GNUPLOT



Menú de configuración de la impresora

Para empezar veremos los comandos `plot` y `splot`, que son el núcleo del programa. Son los encargados de representar las funciones en un montón de formas diferentes. `plot` se emplea para representar funciones y datos 2D, mientras que `splot` se utiliza para funciones y datos 3D.

Sintáxis:

```
plot {ranges} {<function> | {"<datafile>" {using ...}}}
     {title} {style} {, <function> {title} {style}...}

splot {ranges} {<function> | {"<datafile>" {index i} {using ...}}}
     {title} {style} {, <function> {title} {style}...}
```

donde se indica una <función> o el nombre de un fichero de datos entre comillas. Una función es una expresión matemática, o un par (`plot`) o una tripleta (`splot`) de expresiones matemáticas en el caso de funciones paramétricas. Por supuesto, también es posible definir nuevas funciones (de 1 a 5 variables) o constantes en cualquier momento. La sintaxis para la definición de funciones es:

```
<nombre-de-función> ( <var1> {,<var2> {, ...} } ) = <expresión>
```

donde <expresión> está definida en función de las variables <var1>, <var2> hasta <var5>. La sintaxis para definición de constantes es

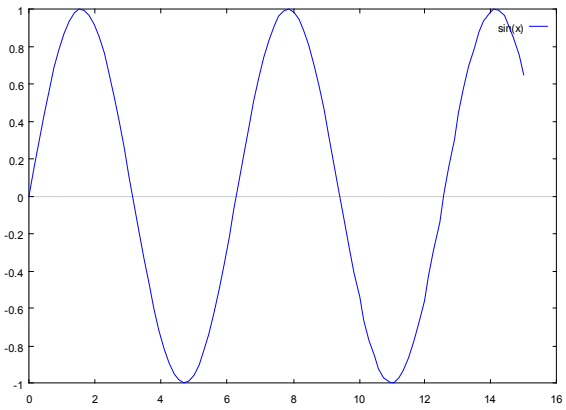
```
<nombre-de-constante> = <expresión>
```

Ejemplos de funciones definidas por el usuario:

```
w = 2
q = floor(tan(pi/2 - 0.1))
f(x) = sin(w*x)
sinc(x) = sin(pi*x)/(pi*x)
delta(t) = (t == 0)
ramp(t) = (t > 0) ? t : 0
min(a,b) = (a < b) ? a : b
comb(n,k) = n!/(k!*(n-k)!)
len3d(x,y,z) = sqrt(x*x+y*y+z*z)
```

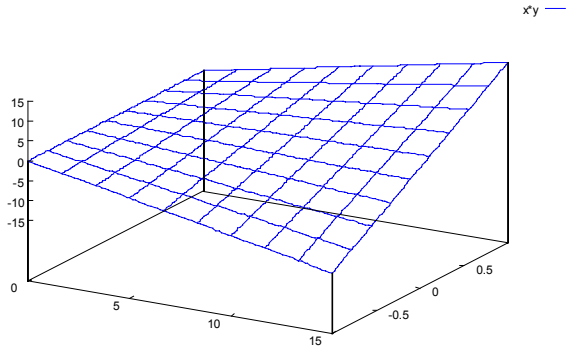
Los comandos plot y splot pueden ser tan sencillos como

```
plot sin(x)
```



y

```
splot x * y
```



o tan complejo como

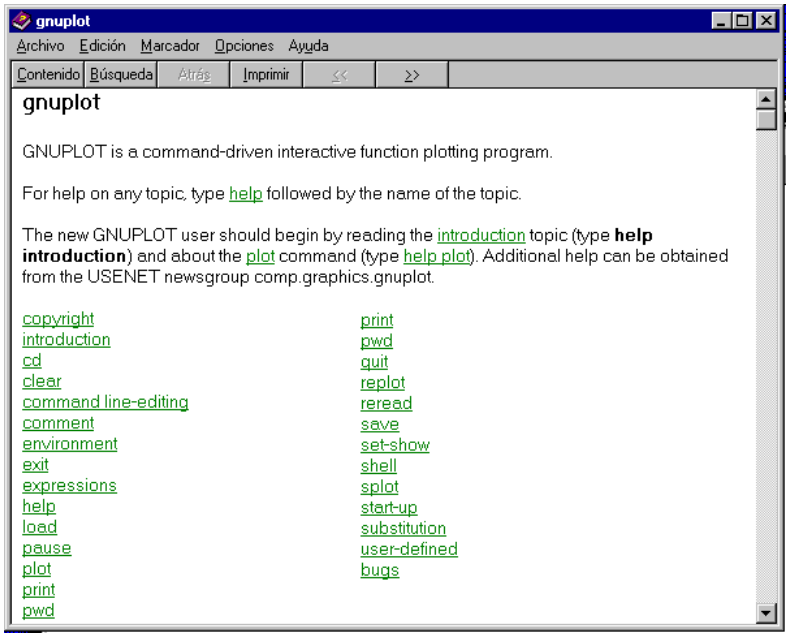
```
plot [t=1:10] [-pi:pi*2] tan(t), "data.1" using 2:3 with lines, t**2 with points
```

aquí la expresión `[t=1:10]` le dice a GNUPLOT que el parámetro `t` varía entre 1 y 10, mientras que `[-pi:pi*2]` indica que sólo se van a presentar los valores de la función comprendidos en este rango. Separadas por comas, en este comando aparecen tres funciones; el segundo de los gráficos se obtiene al representar como `x` los valores contenidos en la segunda columna del fichero `data.1` y como variable dependiente los contenidos en la tercera columna. En cada caso la opción `with` indica el tipo de representación escogida para cada función. Tanto la constante `pi` como la función `tan()` están definidas dentro del programa.

En la siguiente tabla se recogen todas las funciones definidas dentro de GNUPLOT.

abs	acos	arg	asin	atan	besj0	besj1
besy0	besy1	ceil	cos	cosh	erf	erfc
exp	floor	gamma	ibeta	inverf	igamma	imag
invnorm	int	lgamma	log	log10	norm	rand
real	sgn	sin	sinh	sqrt	tan	tanh

Por otro lado, GNUPLOT facilita al usuario toda la ayuda necesaria para el correcto funcionamiento del programa. Para ello sólo es necesario teclear `?` o `help` en la línea de comandos.



Ventana de ayuda de GNUPLOT bajo Windows

También es posible evaluar expresiones sin necesidad de crear ningún gráfico, esto convierte a GNUPLOT en una potente calculadora. Para ello solo es necesario utilizar el comando `print`.

```
gnuplot> print cos(pi/4)
     0.707107
gnuplot> print exp(-0.5*(1.96)**2)/sqrt(2*pi)
     0.0584409
```

Para completar esta capacidad, debemos tener en cuenta también que uno de los múltiples tipos de terminales de los que dispone el programa, en vez de representar gráficamente el resultado de un comando `plot/splot`, crea un tabla con los valores de x, $f(x)$ para `plot` y x, y, $f(x,y)$ para `splot`.

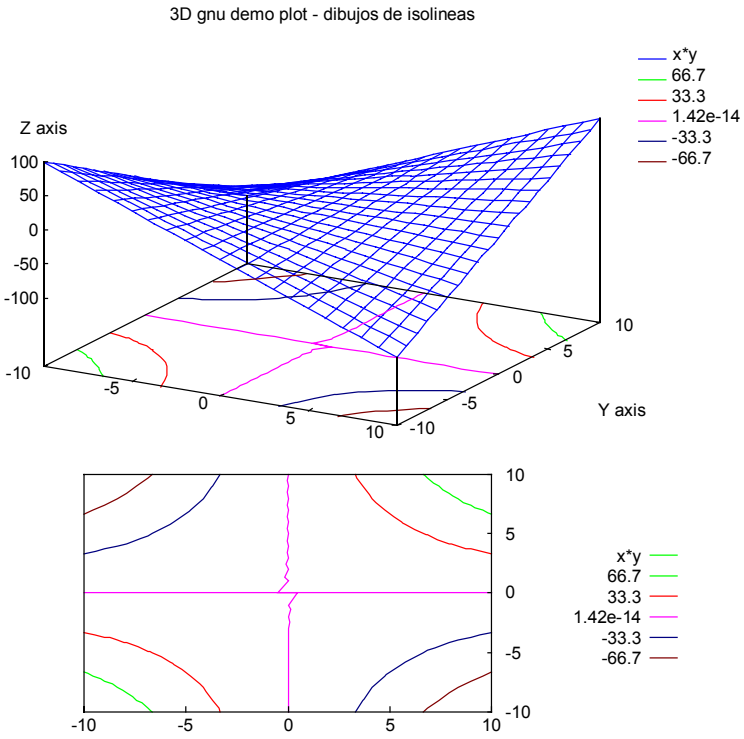
```
gnuplot> set samples 10
gnuplot> set term table
     Terminal type set to 'table'
gnuplot> plot [0:pi] sin(x)
     Curve 0, 10 points
     i x=0 y=0
     i x=0.349066 y=0.34202
     i x=0.698132 y=0.642788
     i x=1.0472 y=0.866025
     i x=1.39626 y=0.984808
     i x=1.74533 y=0.984808
     i x=2.0944 y=0.866025
     i x=2.44346 y=0.642788
     i x=2.79253 y=0.34202
     i x=3.14159 y=0
```

Otras posibilidades interesantes de GNUPLOT son la creación de isolíneas y la eliminación de líneas ocultas. En el primer caso se debe activar esta opción haciendo uso del comando:

```
set contour { base | surface | both }
set nocontour
```

Si no se especifica ninguna opción, por defecto se toma `base`. Estas tres opciones indican la posición que van a ocupar las isolíneas en el dibujo:

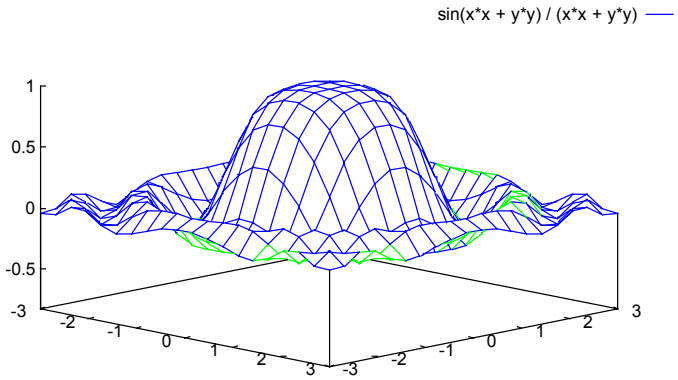
`base`: las isolíneas se dibujan en el plano de referencia xy
`surface`: las isolíneas se dibujan sobre la propia superficie
`both`: se dibujan en las dos posiciones anteriores



Por lo que se refiere a la eliminación de líneas ocultas esta opción se activa mediante el comando:

```
set hidden3d
set nohidden3d
show hidden3d
```

Todas las superficies dibujadas se ven afectadas por este comando, eliminándose las partes ocultas incluso por otras superficies. Esta opción sólo tiene sentido cuando las superficies se representan mediante retículas.



Ejemplo de eliminación de líneas ocultas

LISTADO 7 - OCULTACIÓN DE CARAS EN OBJETOS TRIDIMENSIONALES

En el siguiente listado se presenta un programa que lee desde un fichero, suministrado por el usuario, la definición de un objeto. En la primera línea de este fichero se indican el número de puntos, elementos y vértices que componen el objeto. En las siguientes líneas se indican las coordenadas de los puntos y los índices de los vértices que constituyen cada una de las caras poligonales del objeto. Por último, se ordenan las caras del objeto de tal forma que se dibujan primero las más alejadas del observador.

Debido al número de cálculos necesarios se ha escogido para la realización de este programa el lenguaje C.

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <conio.h>
#include <alloc.h>
#include <math.h>

typedef struct {
   int nv, *v;
   float z;
} ELEM;

float matt[4][4], matr[4][4], matp[4][4];

void mmat (float a[4][4], float b[4][4])
{
   int i, j, k;
   float c[4][4];
   for (i=0;i<4;i++)
   for (j=0;j<4;j++) {
      c[i][j]=0.;
      for (k=0;k<4;k++)
         c[i][j] +=a[i][k]*b[k][j];
   }
   for (i=0;i<4;i++)
   for (j=0;j<4;j++)
      b[i][j]=c[i][j];
}

void proyecta (float p[4], float q[4])
{
   int i, j;
   for (i=0;i<4;i++) {
      q[i]=0.;
      for (j=0;j<4;j++)
         q[i] +=p[j] * matp[j][i];
   }
   for (i=0;i<3;i++)
      q[i] /= q[3];
}

void rota (char eje, float a)
{
   float ang=a * M_PI / 180.;
   if (eje=='X') {
      matr[0][0]=1; matr[0][1]=0; matr[0][2]=0; matr[0][3]=0;
      matr[1][0]=0; matr[1][1]=cos(ang); matr[1][2]=sin(ang); matr[1][3]=0;
      matr[2][0]=0; matr[2][1]=-sin(ang); matr[2][2]=cos(ang); matr[2][3]=0;
      matr[3][0]=0; matr[3][1]=0; matr[3][2]=0; matr[3][3]=1;
   }
   if (eje=='Y') {
      matr[0][0]=cos(ang); matr[0][1]=0; matr[0][2]=-sin(ang); matr[0][3]=0;
      matr[1][0]=0; matr[1][1]=1; matr[1][2]=0; matr[1][3]=0;
      matr[2][0]=sin(ang); matr[2][1]=0; matr[2][2]=cos(ang); matr[2][3]=0;
      matr[3][0]=0; matr[3][1]=0; matr[3][2]=0; matr[3][3]=1;
   }
   if (eje=='Z') {
      matr[0][0]=cos(ang); matr[0][1]=sin(ang); matr[0][2]=0; matr[0][3]=0;
      matr[1][0]=-sin(ang); matr[1][1]=cos(ang); matr[1][2]=0; matr[1][3]=0;
      matr[2][0]=0; matr[2][1]=0; matr[2][2]=1; matr[2][3]=0;
      matr[3][0]=0; matr[3][1]=0; matr[3][2]=0; matr[3][3]=1;
   }
}
```

OCULTACIÓN DE CARAS EN OBJETOS TRIDIMENSIONALES (CONT.)

```
int main(int argc, char *argv[])
{
```

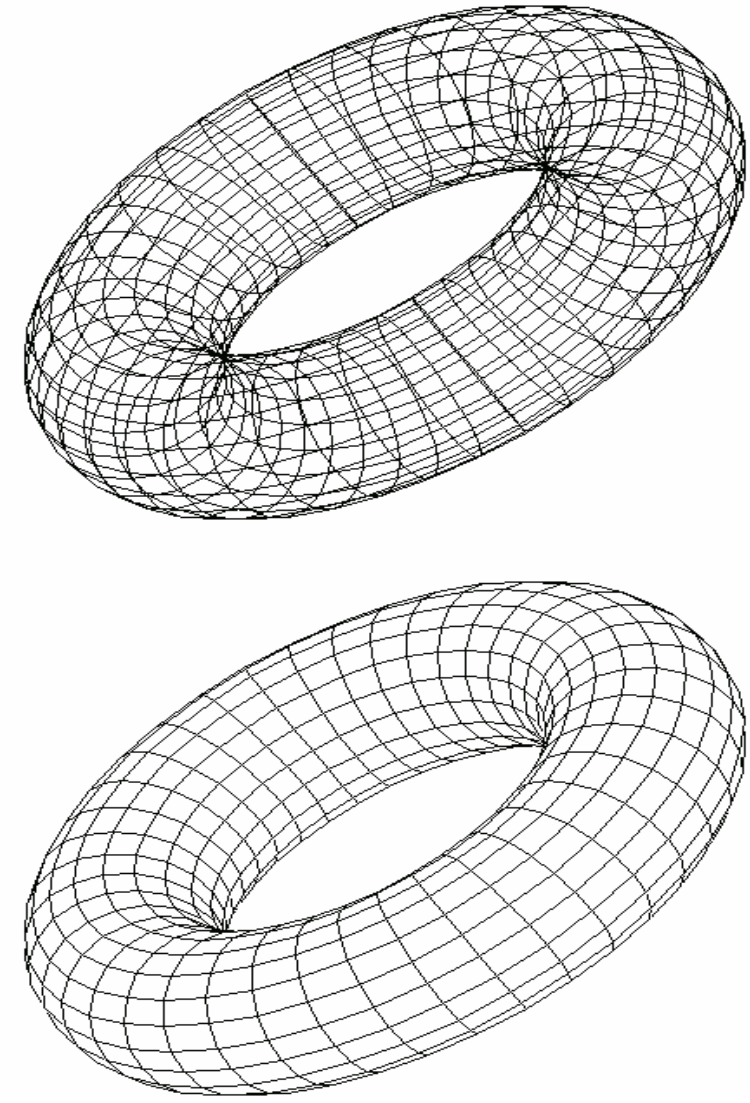
```
int gdriver=DETECT, gmode, errorcode;
int i, j, k, ni, nj, nf, np, ne, nv, pol[128], *id;
char *pv, linea[256];
FILE *fi;
ELEM *el, *ve;
float *x, *y, *z, p[4], q[4];
float xmin, xmax, ymin, ymax, zmin, zmax, xc, yc, zc;
float e, xm, ym, angx, angy, angz, *xp, *yp, *zp;

if (argc !=2) {
   fprintf(stderr, "uso: obj fich.geo\n");
   exit(1);
}

matp[0][0]=1; matp[0][1]=0; matp[0][2]=0; matp[0][3]=0;
matp[1][0]=0; matp[1][1]=1; matp[1][2]=0; matp[1][3]=0;
matp[2][0]=0; matp[2][1]=0; matp[2][2]=1; matp[2][3]=0;
matp[3][0]=0; matp[3][1]=0; matp[3][2]=0; matp[3][3]=1;

if ((fi=fopen(argv[1], "r"))==NULL) {
   fprintf(stderr, "error: no puedo abrir %s\n", argv[1]);
   exit(2);
}

fscanf(fi, "%d %d %d", &np, &ne, &nv);
clrscr();

printf("Ang X: "); scanf("%f", &angx);
printf("Ang Y: "); scanf("%f", &angy);
printf("Ang Z: "); scanf("%f", &angz);

/* coordenadas originales */
x=(float *) calloc (np, sizeof (float));
y=(float *) calloc (np, sizeof (float));
z=(float *) calloc (np, sizeof (float));

/* coordenadas proyectadas */
xp=(float *) calloc (np, sizeof (float));
yp=(float *) calloc (np, sizeof (float));
zp=(float *) calloc (np, sizeof (float));

/* vector de elementos */
ve=(ELEM *) calloc (ne, sizeof (ELEM));
id=(int *) calloc(ne, sizeof (int));

rota('X', angx); mmat(matr, matp);
rota('Y', angy); mmat(matr, matp);
rota('Z', angz); mmat(matr, matp);

xmin = ymin = zmin = 999999.;
xmax = ymax = zmax = -xmin;

for (i=0;i<np;i++) {
   fscanf (fi, "%f%f%f\n", &x[i], &y[i], &z[i]);

   p[0]=x[i]; p[1]=y[i]; p[2]=z[i]; p[3]=1.;
   proyecta (p, q);
   xp[i]=q[0]; yp[i]=q[1]; zp[i]=q[2];

   xmin = min(xmin, xp[i]);
   xmax = max(xmax, xp[i]);
   ymin = min(ymin, yp[i]);

   ymax = max(ymax, yp[i]);
   zmin = min(zmin, zp[i]);
   zmax = max(zmax, zp[i]);
}
xc = 0.5 * (xmin+xmax);
yc = 0.5 * (ymin+ymax);
zc = 0.5 * (zmin+zmax);

/* initialize graphics and local variables */
initgraph(&gdriver, &gmode, "");

/* read result of initialization */
errorcode=graphresult();
/* an error occurred */
if (errorcode !=grOk) {
   printf("Graphics error: %s\n", grapherrormsg(errorcode));
   printf("Press any key to halt:");
   getch();
   exit(1);
}
```

OCULTACIÓN DE CARAS EN OBJETOS TRIDIMENSIONALES (CONT.)

```
setcolor(getmaxcolor());
xm = 0.5 * (getmaxx() + 1);
ym = 0.5 * (getmaxy() + 1);
```

```
   e = 1.5 * min(xm/(xmax-xmin), ym/(ymax-ymin));

   for (i=0;i<ne;i++) {
      fgets(linea, 256, fi);
      pv=strtok(linea, " ");
      sscanf(pv, "%d", &nv);
      ve[i].nv = nv;
      id[i]=i;
      ve[i].v = (int *) calloc (nv, sizeof (int));
      for (j=0;j<nv;j++) {
         pv=strtok(NULL, " ");
         sscanf(pv, "%d", &ni); ni--;
         ve[i].v[j] = ni;
      }
   }
   fclose(fi);

   for (i=0;i<ne;i++) {
      el = &ve[i];
      ni = el->v[0];
      el->z = zp[ni];
      for (j=1;j<el->nv;j++) {
         nj = el->v[j];
         line (xm + (xp[ni]-xc)*e, ym - (yp[ni]-yc)*e,
            xm + (xp[nj]-xc)*e, ym - (yp[nj]-yc)*e);
         ni = nj;
         el->z += zp[ni];
      }
      el->z /= el->nv;
      nj = el->v[0];
      line (xm + (xp[ni]-xc)*e, ym - (yp[ni]-yc)*e,
         xm + (xp[nj]-xc)*e, ym - (yp[nj]-yc)*e);
   }

   getch();

   for (i=0;i<ne;i++)
   for (j=i;j<ne;j++)
      if (ve[id[i]].z > ve[id[j]].z) {
         k = id[i];
         id[i] = id[j];
         id[j] = k;
      }

   for (i=0;i<ne;i++) {
      el = &ve[id[i]];
      for (j=0, k=0;j<el->nv;j++) {
         ni = el->v[j];
         pol[k++] = xm + (xp[ni]-xc)*e;
         pol[k++] = ym - (yp[ni]-yc)*e;
      }
      setfillstyle (SOLID_FILL, getbkcolor());
      fillpoly(el->nv, pol);
   }
   getch(); /* clean up */
   closegraph();
   return 0;
}
```

Resultado de la ejecución:



FOTOREALISMO

INTRODUCCIÓN

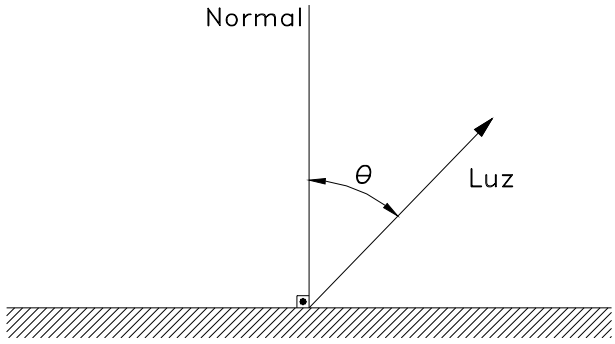
En el capítulo anterior hemos visto como construir objetos utilizando caras poligonales. Somos capaces ahora de proyectar y representar imágenes en perspectiva de objetos para obtener una idea de profundidad. Hemos visto también como eliminar las líneas y las superficies ocultas de los objetos para dar a las imágenes un aspecto más realista. Los polígonos que constituyen los objetos pueden ser sombreados. Generalmente, será posible dibujar polígonos con diferentes tramas de relleno, consiguiendo de esta forma reflejar los diferentes tonos de color que tienen las superficies del objeto.

ILUMINACIÓN DIFUSA

Consideremos primero una fuente de luz indirecta. Supondremos que nuestro objeto está iluminado por una luz que no proviene de una fuente en particular de luz si no que ilumina al objeto desde todas las direcciones. Esta recibe el nombre de luz de fondo, luz que es reflejada por las paredes, suelo y techo. Suponemos que esta luz es uniforme, que tiene la misma intensidad en todos los puntos. No habrá puntos brillantes ni una dirección predominante. De esta forma cualquier superficie del objeto recibe la misma intensidad de luz difusa sea cual sea su orientación espacial.

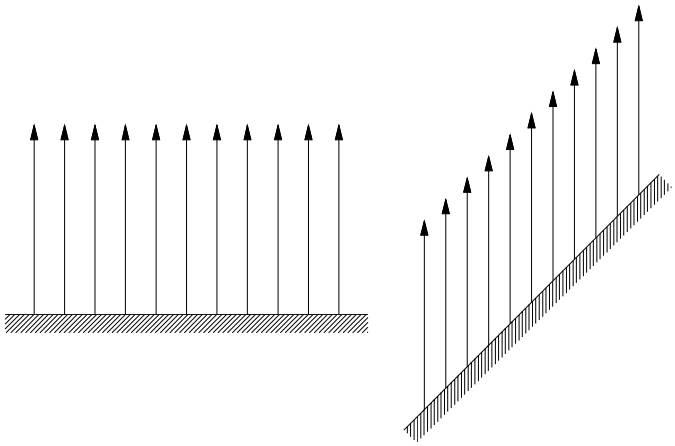
De toda la luz que incide sobre el objeto, una porción es absorbida por su superficie mientras que el resto es reflejado o reemitido. La proporción de luz reflejada respecto al total recibe el nombre de coeficiente de *reflexión* o *reflectividad*. Una superficie blanca refleja casi la totalidad de la luz que incide sobre ella, es decir, su coeficiente de reflexión es próximo a 1. Una superficie negra, por el contrario, absorbe la mayor parte de la luz que llega a ella, es decir, su reflectividad es casi nula. Una superficie color gris tendrá un coeficiente de reflexión intermedio. La mayor parte de los objetos reflejan unos colores mejor que otros. Por ejemplo, si un objeto tiene una elevada reflectividad para el color rojo y valores pequeños para el resto de colores, entonces dicho objeto se verá de color rojo. Si empleamos un monitor en color que da a cada punto de la pantalla diferentes intensidades de verde, rojo y azul, deberemos especificar tres valores de reflectividad con el fin de conocer las características de color y sombra del objeto. Pero para que las cosas no sean demasiado complicadas sólo tendremos en cuenta los monitores con tonos de gris. Así un único coeficiente de reflexión (R) nos indicará el tono de gris del objeto que vamos a representar.

Hemos supuesto que la cantidad de luz que incide sobre el objeto no cambia con la posición de este porque la luz proviene uniformemente de todas las direcciones, pero ¿varía la cantidad de luz reflejada hacia nuestros ojos con las diferentes orientaciones? La respuesta es que el objeto se verá igual de brillante independientemente de su posición. Este resultado se produce por la cancelación de dos efectos. El primer efecto es que se emite más luz en la dirección perpendicular a la superficie que en la dirección paralela a ella. La relación exacta entre las dos direcciones recibe el nombre de Ley de Lambert, y establece que la reflexión de la luz por una superficie perfectamente difusora varía con el coseno del ángulo que forman la normal a la superficie y la dirección del rayo reflejado.



Dirección de la luz respecto a la normal a la superficie

Por lo tanto la cantidad de luz procedente de un punto de la superficie del objeto disminuye con el coseno del ángulo a medida que la superficie gira alejándose de la dirección de visualización. El segundo efecto, que compensa al anterior, es que nosotros no vemos punto sino áreas. A medida que la superficie gira alejándose el número de puntos emisores de luz dentro del área de visión aumenta, y este aumento es precisamente proporcional al inverso del coseno. Por lo tanto, a medida que gira la superficie del objeto, recibimos menos luz desde cada punto, pero estos parecen aproximarse unos a otros, por lo que el efecto conjunto es que la superficie mantiene su brillo.



Una superficie vista de lado aumenta el área productora de luz

Una cancelación similar ocurre si el objeto se aleja desde el observador. La luz procedente de su superficie se reparte sobre área mayor, que crece con el cuadrado de la distancia, por lo que la luz que llega a nuestro ojo disminuye en el mismo factor. Aplicando el mismo razonamiento que antes, aunque al ojo llega menos luz, lo hace sobre un área más pequeña de la retina y el brillo del objeto no varía.

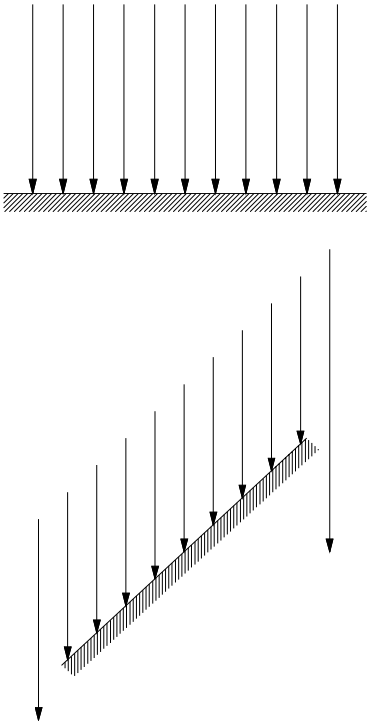
Podemos dar ahora una expresión para el brillo de un objeto iluminado por una luz difusa de fondo. Si F es la intensidad de la luz de fondo y R es la reflectividad del objeto, entonces la intensidad de la luz que procede de su superficie visible será

$$TONO = FR$$

Si podemos modificar los valores de F y R, podemos pintar nuestro objeto con diferentes tonos de gris. Pero en este sencillo modelo, cualquier plano de un determinado objeto tendrá el mismo color. Sin embargo, este no es el aspecto real de objeto, para obtener un coloreado más realista debemos incluir también las fuentes puntuales de luz.

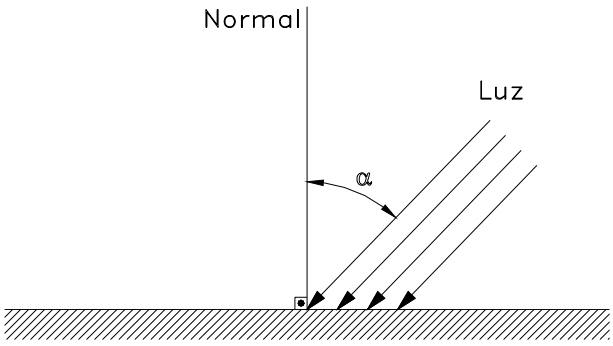
ILUMINACIÓN PUNTUAL

Las fuentes de luz puntual son abstracciones de las fuentes reales de luz como las bombillas, las velas o el propio Sol. La luz se origina en un determinado lugar, proviene de una dirección determinada desde una distancia determinada. Para las fuentes puntuales, la posición y la orientación relativa de la superficie del objeto determinara cuanta luz recibirá y, como consecuencia de esto, que brillo tendrá. Las superficies que miren hacia la luz y que estén situadas más cerca de la fuente recibirán más luz que las superficies que estén orientadas en dirección opuesta o se encuentren más lejos de la luz.



Una superficie perpendicular recibe más luz que otra que forme un ángulo

Empecemos considerando los efectos de la orientación. Utilizando argumentos similares a los del capítulo anterior, sabemos que la superficie, a medida que gira alejándose de la dirección de la luz, aparece menos brillante. La iluminación disminuye con el coseno del ángulo α que forma la normal a la superficie y la dirección de luz incidente. Este ángulo recibe el nombre de *ángulo de incidencia*.



Ángulo de Incidencia

Si tenemos un vector L de longitud unidad apuntado hacia la fuente de luz y un vector N de longitud unidad en la dirección normal a la superficie, entonces el producto escalar de estos dos vectores es

$$\cos\alpha = L \cdot N$$

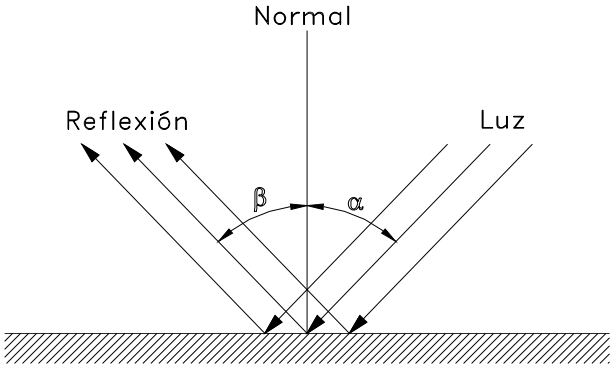
Supongamos que P es la cantidad de luz que proviene de la luz puntual, entonces el tono de una cara determinada de un objeto será

$$TONO = FR + P(L \cdot N)R$$

REFLEXIÓN ESPECULAR

Nuestro modelo puede ser mejorado todavía más. La luz es reflejada por los objetos de dos formas: existe una reflexión difusa, que hemos tenido en cuenta antes, y existe la *reflexión especular*. La reflexión difusa sólo depende del ángulo de incidencia, pudiendo la luz reflejada tener un color determinado, ya que interviene el coeficiente de reflexión. La reflexión especular actúa de una forma diferente. Es el tipo de reflexión que se produce en un espejo. Toda la luz es reflejada, no solamente algunos colores, y lo hace en una dirección casi única, no en todas las direcciones como establecía la ley de Lambert. Para la reflexión especular, la luz llega, alcanza la superficie y rebota. El ángulo β que forma el rayo reflejado con la normal a la superficie recibe el nombre de *ángulo de reflexión* y tiene una magnitud igual al ángulo de incidencia.

La luz procedente de una reflexión especular puede observarse cuando la dirección desde el objeto al ojo del observador es la misma que la dirección de la luz reflejada. Otra forma de expresar esto es forma un vector a medio camino entre la dirección de la luz incidente y la dirección del observador. Si comparamos este vector con la normal a la superficie, situado a mitad de camino entre el rayo incidente y el rayo reflejado, cuando coinciden el observador puede ver la luz reflejada.

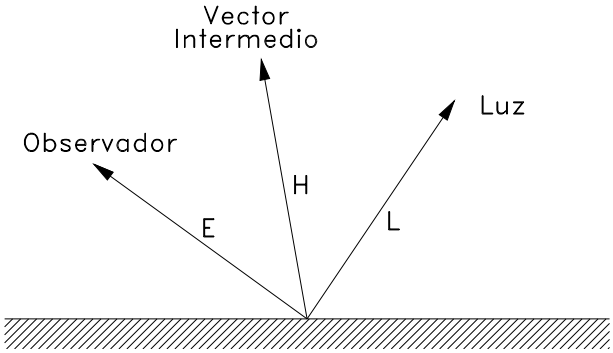


Reflexión Especular

Si L es el vector de longitud unidad en la dirección del rayo incidente pero saliendo de la superficie y E es el vector unitario en la dirección del observador, entonces

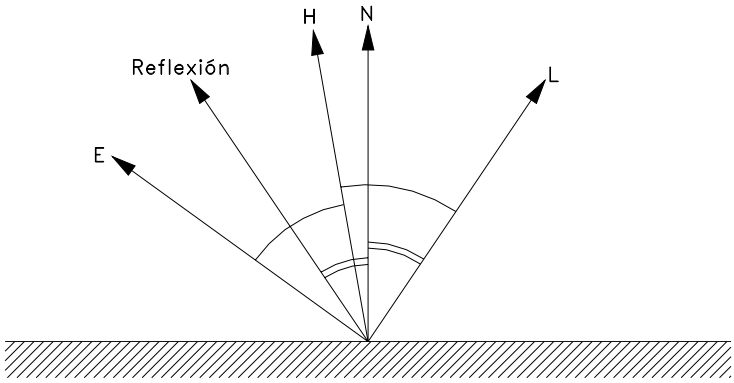
$$H = \frac{L + E}{|L + E|}$$

será un vector unitario a medio camino entre ambos.



Vector intermedio entre la fuente de luz y el observador

Si N es el vector unitario normal a la superficie, el producto escalar de estos dos vectores nos dará el coseno del ángulo que forman. Cuando la reflexión puede verse, el ángulo entre estos dos vectores será casi nulo y, por lo tanto, su coseno estará próximo a 1.

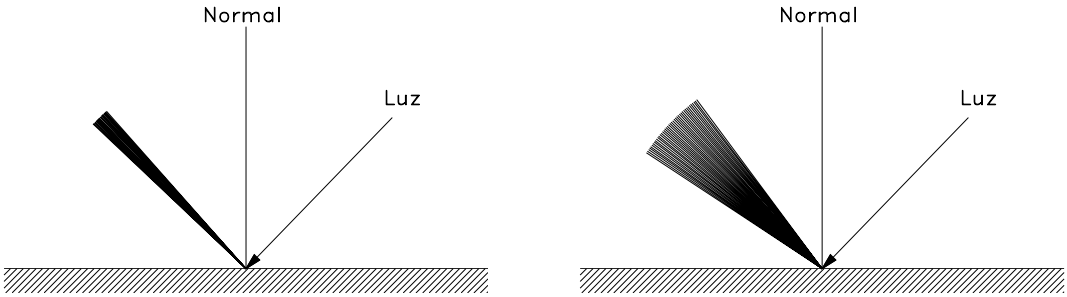


Vectores empleados para la determinación de lo que el ojo en la reflexión

Necesitamos entonces una función que valga 1 cuando podemos ver la luz reflejada, pero cae rápidamente hasta 0 para cualquier otro ángulo. Esto puede conseguirse elevando el coseno a una potencia grande.

$$F = (N \cdot H)^a$$

El valor de a determina la brillantez de un objeto. Cuando la potencia es elevada se obtiene un efecto especular muy brillante.



Efecto de la potencia de a: a grande (izquierda) y a pequeña (derecha)

Si la cantidad de luz reflejada especularmente es S, entonces nuestra función mejorar de sombreado será

$$TONO = FR + P(L \cdot N)R + S(N \cdot H)^a$$

La cantidad de luz reflejada puede depender del valor del ángulo de incidencia. De hecho el parámetro S debería ser una función de α. Esta puede ser una función complicada que varía con el tipo de material. Para simplificar nuestro modelo, consideraremos, sin embargo, que es constante para cada objeto. Incluida en S está la potencia de la luz puntual P.

Por último, debemos tener en cuenta el efecto de la distancia. En teoría, la iluminación procedente de una fuente de luz puntual debería disminuir con el cuadrado de la distancia entre esta y el objeto. Utilizando esta relación, sin embargo, parece que se produce un cambio demasiado grande con la distancia. Quizás sea debido en parte al hecho de que las fuentes de luz puntual son más grandes que un simple punto. Un objeto iluminado por una luz muy grande cercana a él no mostrará apenas un cambio en la iluminación para pequeños cambios de la distancia. En cualquier caso, necesitamos una función que disminuya con la distancia, pero menos rápidamente que el inverso del cuadrado. Y también queremos acercarnos a un límite finito para pequeñas distancias, de tal forma que no sea necesario preocuparse por las divisiones por cero cuando una fuente de luz esta mal colocada.

Se propone la siguiente función

$$G = \frac{1}{C + D}$$

donde C es una constante y D es la distancia desde la fuente de luz al objeto.

Nuestra función de sombreado se convierte es

$$TONO = FR + \frac{P(L \cdot N)R + S(N \cdot H)^a}{C + D}$$

o de forma generalizada para varias fuentes de luz

$$TONO = FR + \sum_j \frac{P_j(L_j \cdot N)R + S_j(N \cdot H_j)^a}{C + D_j}$$

TRANSPARENCIAS Y SOMBRAS

Algunas de las propiedades que no han sido incluidas en este trabajo son las transparencias y las sombras. El efecto de transparencia se consigue mostrando parte de la luz procedente de un objeto *oculto*. Cuanto mayor es el porcentaje de luz que se deja pasar a través del objeto, más transparente es. Para incluir objetos transparentes en nuestro modelo sería necesario modificar el algoritmo del pintor de tal forma que atenuase el color de un objeto cuando dibujamos otro encima, en vez de eliminarlo por completo, para luego añadir la luz procedente del segundo.

El problema de las sombras es muy parecido al problema de la superficies ocultas. Un objeto sombreado es uno que no recibe directamente la luz pero que está en nuestra línea de visión. Los cálculos necesarios para determinar las superficies ocultas y las sombreadas se suelen realizar simultáneamente por eficiencia. Sin embargo, para poder realizar este proceso es necesario un planteamiento distinto del algoritmo del pintor que nosotros hemos visto para las superficies ocultas. Para tener en cuenta las sombras, es necesario identificar las porciones de las superficies que están a la sombra de un objeto y no sólo eliminarlas.

BIBLIOGRAFÍA

Ajenjo, A.D.
Tratamiento Digital de Imágenes
Ed. Anaya Multimedia, 1993

Berg, M. de; Kreveld, M. van; Overmars, M.; Schwarzkopf, O.
Computational Geometry, 2nd ed. Algorithms and Applications
Ed. Springer-Verlag,1997-2000

Blinn, J.F.
Models of Light Reflection for Computer Synthesized Pictures
Computer Graphics, vol. 11, no. 2, pp 193-198. 1977

Chasen, S.H.
Geometric Principles and Procedures for Computer Graphics Application
Prentice-Hall, Englewood Cliffs, N.J. 1978

Félez, J.; Martínez, M.L.; Cabanellas, J.M.; Carretero. A.
Fundamentos de Ingeniería Gráfica
Ed. Síntesis, 1996

Foley, J.
Computer Graphics
Ed. Addison-Wesley Publishing Company, 1990 (2ª Edición)

Forrest, A.R.
Curves for Computer Graphics
Pertinent Concepts in Computer Graphics, Univ. of Ill. Press, pp. 31-47. 1969

Glaeser, G.
Fast Algorithms for 3D-Graphics
Ed. Springer-Verlag, 1994

Harrington, S.
Computer Graphics: A programming Approach
McGraw-Hill Book Company, 1985 (2ª Edición)

Newman, W.M.; Sproull, R.F.
Principles of Interactive Computer Graphics, 2d ed.
McGraw-Hill, New York. 1979

O'Rourke, J.
Computational Geometry in C
Cambridge University Press, 1994

Phong, Bui-Tuong
Illumination for Computer-Generated Pictures
Comm. ACM, vol. 18, no. 6, pp. 311-317. Junio 1977

Rogers, D.F.; Adams, J.A.
Mathematical Elements for Computer Graphics
McGraw-Hill Book Company, 1976

Rooney, J; Steadman, P.
Principles of Computer-aided Design
UCL Press, Open University, 1993