Universidad de Salamanca

Facultad de Ciencias

Ingeniería en Informática

INFORMÁTICA GRÁFICA

OpenGL

"Manipulación del espacio: Trasformaciones y proyecciones"

Alberto Sánchez Castaño *52410311-G*
José Germán Sánchez Hernández *70889076-V*

Índice

Lista de Figuras

6.1 Introducción

En capítulos anteriores, se ha mostrado como dibujar diferentes elementos gráficos como puntos, rayas y polígonos.

Es muy común que dispongamos de una especificación del objeto en un cierto sistema de coordenadas (sistema de coordenadas del objeto o sistema local). Por ejemplo, a la hora de modelar un polígono en la que las coordenadas de los vértices y la orientación de los vectores normales vienen definida respecto a un sistema ortogonal situado por ejemplo, en el centro de masas del polígono. Sin embargo, los diferentes objetos situados en una escena comparten un sistema de referencia absoluto (sistema de coordenadas del mundo, de la escena, o sistema global). Por tanto necesitamos un mecanismo para transformar la posición, orientación y escala de los objetos de forma que podamos combinarlos en una escena común con otros, mover el objeto o efectuar diferentes copias del mismo en diferentes posiciones y tamaños.

La forma en que estas transformaciones de modelado se efectúan en las librerías gráficas suele ser a través de un sistema de coordenadas actual que se almacena en el contexto gráfico. Este estado actual del sistema de coordenadas se representa por medio de una matriz que expresa la transformación que en cada instante lo liga con el sistema absoluto del mundo. Supongamos que tenemos una función que dibuja un cubo cuyos vértices han sido definidos respecto a con un sistema de coordenadas situado en su centro.

¿Cómo podemos dibujar el cubo en una posición cualquiera de la escena? Suponiendo que el sistema de coordenadas actual de la escena está inicialmente en el origen del sistema del mundo, tendríamos que trasladarlo al punto donde queremos situar el cubo, y entonces llamar a la función que dibuja el cubo.

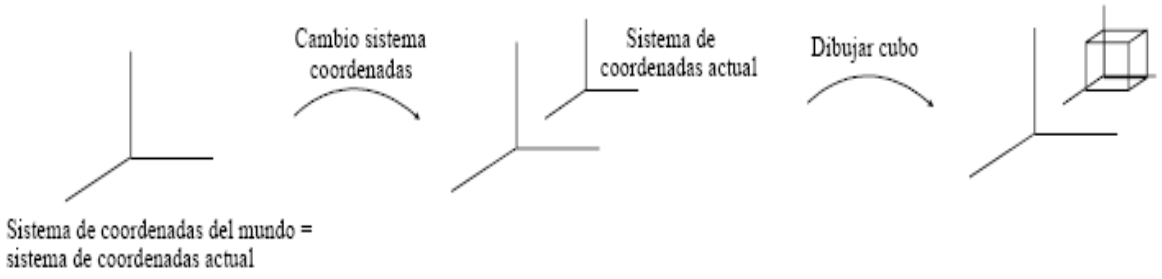

Figura 1. Cambio de sistema de coordenadas.

En este capítulo se pretende exponer los pasos que se siguen para generar una escena que incluya esos elementos, cómo representarla en pantalla y cómo modificar la escena resultante aplicando transformaciones, tales como rotación, traslación o escalado de la propia escena o de los elementos que la componen.
Una vez que se han expuesto las diferentes técnicas de transformación de coordenadas surge la necesidad de crear una imagen bidimensional constituida por objetos definidos en espacios tridimensionales, de forma que puedan ser representados en una pantalla. A partir de un objeto real en tres dimensiones se proyectaría en un conjunto de pixeles de dos dimensiones sobre la pantalla, el proceso es similar al proceso de realización de una fotografía con una cámara, las diferentes técnicas de proyección serán expuestas detalladamente a lo largo de este documento.

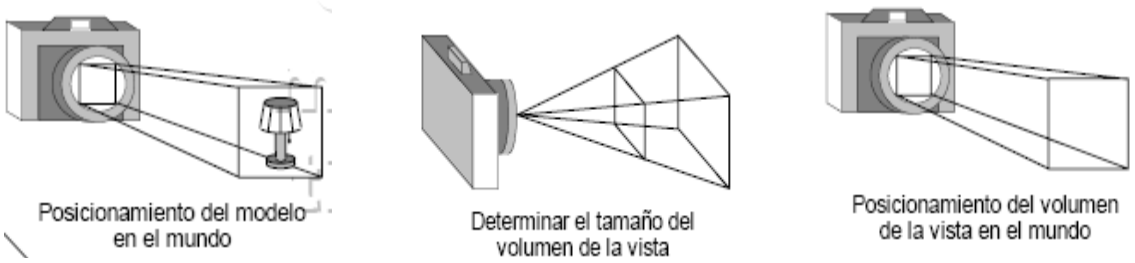

Figura 2. Proyecciones.

6.2 Transformaciones

Se define transformación como un conjunto de modificaciones que se aplican a un modelo en tres dimensiones, de modo que sea posible su representación en una pantalla bidimensional.

Las operaciones básicas que permiten realizar las transformaciones sobre los objetos se pueden dividir en tres: rotaciones alrededor de los ejes de coordenadas, traslaciones y cambios de posición y escalado o cambios en las dimensiones de los objetos.

Antes de comenzar el estudio de las transformaciones lineales en el espacio tridimensional, se han de repasar algunos conceptos básicos en este campo.

6.2.1 Consideraciones previas

Previo a detallar los diferentes tipos de transformaciones, se explicarán brevemente diferentes conceptos y elementos matemáticos necesarios para la correcta realización de las diferentes transformaciones.

6.2.1.1 Sistemas de referencia

Para hacer una correcta visualización de los objetos, éstos han de ser ubicados en un sistema universal de referencia (*SUR*), en el cual el eje Z, p. ej., puede tener su origen en el plano Z— (orientación derecha), por lo que se dirige hacia el observador (Figura 3-a), o bien tener orientación izquierda (Figura 3-b). Todas las coordenadas de los distintos objetos han de estar dadas en uno de estos sistemas de referencia.

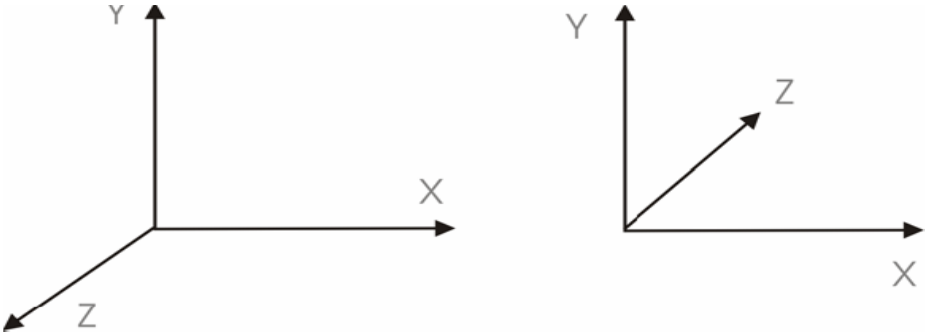

Figura 3. Sistemas de coordenadas.

Cuando se aplique una transformación a los elementos de la escena, será necesario modificar el sistema de coordenadas, con lo que se tendrá un nuevo sistema de coordenadas transformado, pero las coordenadas seguirán siendo las mismas, ya que son fijas y constituyen un marco de referencia.

OpenGL utiliza el sistema de coordenadas cartesiano para la representación de objetos en la pantalla, en el cual la dirección positiva del eje X se representa hacia la derecha y la dirección positiva del eje Y hacia arriba. El eje Z se corresponde con la línea imaginaria que une el observador con el centro de coordenadas. Su dirección negativa se corresponde con los valores de los puntos más alejados del observador, y la dirección positiva, con los puntos más cercanos a éste.

6.2.1.2 Transformaciones lineales y matrices

Las herramientas matemáticas utilizada para realizar la variación de la posición y/o el tamaño de los objetos, con respecto a los sistemas de referencia, son las

transformaciones lineales. Las transformaciones lineales que se expondrán serán las siguientes: traslación, cambio de escala, giro, y reflexión.

En la Informática Gráfica suele utilizarse la notación matricial para describir las transformaciones lineales de los objetos. La convención más utilizada es que el punto (vértice) que se va a transformar, se exprese mediante un vector horizontal multiplicado por la matriz de transformación. Por ejemplo, en la expresión *(x', y') = (x, y)·M*, la matriz correspondiente a la transformación lineal estaría indicada por M; el punto inicial (antes de la transformación) sería el *(x, y)*, y el resultado (o sea, la ubicación del punto en el sistema de referencia después de la transformación lineal) sería el *(x', y').*

6.2.1.3 Sistemas Homogéneos

De acuerdo a lo expuesto en el apartado anterior, siendo $V = (x, y)$ el vector de un punto inicial en el sistema de coordenadas, $T = (t_x , t_y)$ el vector de translación, y V' = (x', y') las coordenadas del punto resultante, ocurre que la traslación del punto se ha de realizar calculando

$$\mathbf{V'} = \mathbf{V} + \mathbf{T},$$

ya que $(x', y') = (x, y) + (t_x , t_y)$, o lo que es igual, $x' = x + t_x$, *e* $y' = y + t_y$. Esta manera de operar es extensible a cualquier dimensión.

Por otro lado, si *E* y *G* fuesen las matrices de escalado y giro, respectivamente, ocurriría que

$$\mathbf{V'} = \mathbf{V}\cdot\mathbf{E} \text{ y } \mathbf{V'} = \mathbf{V}\cdot\mathbf{G}$$

Vemos entonces que las traslaciones lineales de los puntos en el espacio se efectúan sumando, mientras que los giros y los cambios de escala se consiguen multiplicando. La heterogeneidad de los operadores supone un problema a la hora de generalizarlos procesos de las transformaciones, por lo que para evitarlo, normalmente se utilizan **sistemas** [de referencia] **homogéneos**. Básicamente, un sistema de coordenadas homogéneo es el resultante de añadir una dimensión extra a un sistema de referencia dado. Así, en el caso anterior, los vectores homogéneos de los puntos inicial y final estarían dados por

$$\mathbf{V} = (x, y, w), \mathbf{V'} = (x', y', w)$$

Por comodidad y sencillez, normalmente w = 1.

En definitiva, utilizando un sistema homogéneo, las traslaciones lineales de los

puntos del plano pueden quedar expresadas como *V' = V · T*, si se utiliza la matriz de traslación (*T*) apropiada. Así, siendo

$$\mathbf{T} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_x & t_y & 1 \end{bmatrix}$$

es fácil comprobar que *(x', y', 1) = (x, y, 1)·**T**.*

6.2.1.4 Composición de matrices

El ultimo concepto matemático que se va a introducir es la composición de matrices, dicha operación es una de las principales razones para trabajar con sistemas homogéneos. Matemáticamente consiste en la multiplicación de las matrices en un orden determinado, ya que el orden en que se multipliquen es a menudo importante.

Supongamos que tenemos el punto *P = (x, y, z, 1)* del espacio tridimensional, que como vemos está expresado en un sistema homogéneo. Al hacer

(x', y' ,z', 1) = (x, y,z, 1) **T**
*(x'', y'',z'',) = (x', y',z', 1)***G**

lo que se consigue es mover primero el vértice *P*, y luego a continuación girarlo. Se puede llegar al mismo resultado final, si multiplicamos el vector P por la matriz resultante de componer *T* y *G* (en este orden). Así, siendo *M = T·G*, se tiene que

*(x'', y'',z'',*1*)* **=** *(x, y,z,*1*)***M**

El orden en que se multiplican las matrices es importante ya que, en general, el producto de matrices no es conmutativo. En la composición de matrices pueden intervenir tantos factores (matrices) como se requieran. Así, siendo *Mn* la matriz compuesta o neta resultante de la composición de las matrices *T1, G, T2, y E* (o sea, *Mn = T1·G·T2·E*), al multiplicar un punto por esta matriz obtendremos el mismo punto final que hallaríamos multiplicándolo sucesivamente por las matrices que la componen.

En 3D, por lo común, la expresión general de una matriz neta (compuesta) es de la forma

$$\begin{vmatrix} a_{11} & a_{12} & a_{13} & 0 \\ a_{21} & a_{22} & a_{23} & 0 \\ a_{31} & a_{32} & a_{33} & 0 \\ d_x & d_y & d_z & 1 \end{vmatrix}$$

donde la submatriz *Aij* representa el cambio de escala y rotación neta, y *Di* el vector de desplazamiento neto de los puntos. Si en la composición de la matriz se incluyen las matrices que gobiernan las proyecciones en perspectiva, el aspecto de la matriz neta sería

$$\begin{vmatrix} a_{11} & a_{12} & a_{13} & p_1 \\ a_{21} & a_{22} & a_{23} & p_2 \\ a_{31} & a_{32} & a_{33} & p_3 \\ d_x & d_y & d_z & 1 \end{vmatrix}$$

siendo *Pj* el vector de perspectiva.

En general, las transformaciones de vectores homogéneos por medio de este tipo de matrices netas se denominan transformaciones bilineales.

6.2.2 Transformaciones del modelo

Una vez explicadas brevemente las herramientas de apoyo se van a estudiar cada una de las diferentes transformaciones que puede experimentar un objeto de una escena dado un sistema de coordenadas.

Este tipo de transformaciones se refiere a las modificaciones o alteraciones que se aplican sobre los objetos que forman una escena. En realidad, los mecanismos empleados para las transformaciones geométricas en tres dimensiones se extienden de los métodos utilizados en entornos bidimensionales, incluyendo las consideraciones sobre el eje Z.

Como en el caso de las dos dimensiones, las transformaciones se representan en forma de matriz, y cualquier secuencia de transformaciones se puede representar como una única matriz resultado de la concatenación de las matrices correspondientes a las transformaciones individuales.

Las transformaciones geométricas básicas que se pueden aplicar a los objetos son la traslación, rotación y escalación, aunque también se aplican con frecuencia otras como la reflexión o el recorte.

Estas operaciones de transformación son siempre matriciales, pueden descomponerse en sumas y multiplicaciones, y por tanto caen dentro de la categoría de operaciones lineales que podemos incluir de forma eficiente en el procesado gráfico de los vértices que definen a los objetos.

Para visualizar cómo se combinan diferentes transformaciones de modelado podemos utilizar estructuras arbóreas como ésta; en las que se especifica el orden de las operaciones y a qué objetos afecta (a todos aquellos bajo el nodo que indica la transformación).

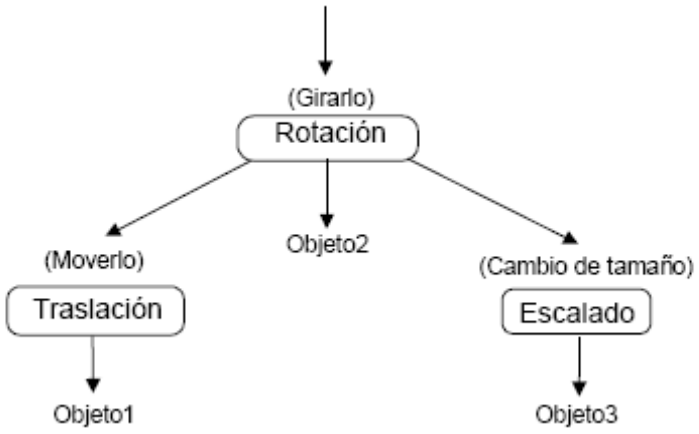

Figura 4. Tipos de transformaciones.

6.2.2.1 Traslación

La transformación de translación consiste en mover una cierta distancia un objeto en una dirección determinada, como consecuencia el objeto se encontrará en una posición nueva.

En realidad, son los puntos que definen el objeto los que se trasladan de forma individual. Para trasladar un objeto que se representa como un conjunto de

polígonos, se trasladan cada uno de los vértices de los polígonos que lo componen y a continuación se redibujan dichos polígonos en la nueva ubicación.

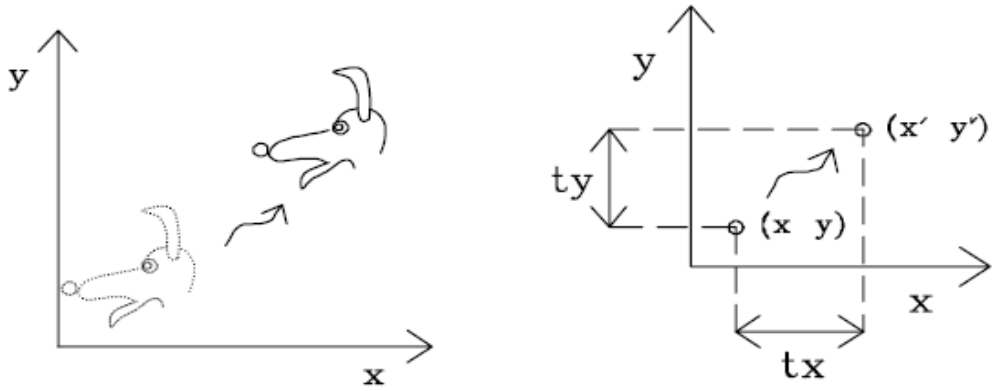

Figura 5. Translación de objetos y coordenadas.

En 3D, el sistema de referencia homogéneo tendrá 4 dimensiones, por lo que la traslación del punto *V = (x, y, z, 1)* quedará indicada como *V' = (x', y', z', 1) = (x, y, z, 1)·T*, siendo *T*,

$$\mathbf{T}=\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ t_x & t_y & t_z & 1 \end{bmatrix}$$

la matriz de traslación en 3D. (*tx*, *ty*, *tz*) se conoce como el *vector de traslación*. La expresión anterior es equivalente al sistema de ecuaciones

$$\begin{aligned} x' &= x + t_x \\ y' &= y + t_y \\ z' &= z + t_z \\ 1 &= 1 \end{aligned}$$

$$T(t_x, t_y, t_z) = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad \begin{pmatrix} x' \\ y' \\ z' \\ \omega' \end{pmatrix} = T(t_x, t_y, t_z) \cdot \begin{pmatrix} x \\ y \\ z \\ \omega \end{pmatrix} = \begin{pmatrix} x + t_x \cdot \omega \\ y + t_y \cdot \omega \\ z + t_z \cdot \omega \\ \omega \end{pmatrix}$$

Matriz de translación

Para realizar la traslación inversa a la efectuada mediante la matriz *T*, se ha de aplicar la *matriz inversa*, es decir, la *T-1*, que se obtiene cambiando el signo (multiplicando por −1) el *vector de traslación*. Por tanto,

$$\mathbf{T}^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -t_x & -t_y & -t_z & 1 \end{bmatrix}$$

Así, *V = (V·T)·T-1*, dado que *T·T-1 = 1*.

6.2.2.2 Rotación

Para realizar el giro de los objetos en 3D, el usuario ha de establecer un eje de rotación, así como el ángulo y el sentido de giro alrededor de dicho eje. Según sea la naturaleza del ángulo, los giros podrán ser relativos, o bien absolutos. Así, cuando se deba girar el objeto un ángulo ϕ sobre un eje dado, partiendo de la posición actual del objeto, el giro será relativo. En cambio, si el objeto se ha de girar un ángulo Φ a partir del estado cero (sistema de referencia), entonces se trataría de un giro absoluto.

En las rotaciones, es especialmente importante el sentido en el que giran los objetos y esta es la razón por la cual se trata de seguir una convención para que no se produzcan ambigüedades. En general, las medidas de ángulos de rotación positiva generan giros en sentido antihorario, cuando se mira hacia el origen, desde una posición con coordenadas positivas.

Normalmente, los giros en 3D se realizan aprovechando la base trigonométrica de las rotaciones en 2D, es decir, descomponiendo los giros 3D en sus componentes

ortogonales. Por esta razón, primero se mostrará cómo se realizan los giros en 2D.

6.2.2.2.1 Giros en 2D

Puesto que se ha de establecer el sentido de giro, vamos a suponer que el giro alrededor de un eje ortogonal será positivo, cuando sea contrario al sentido de giro de las agujas del reloj, mirando cada eje de coordenadas desde fuera hacia el origen. Así, un giro positivo alrededor del eje Z de un punto (P) situado inicialmente en las coordenadas (X, Y, Z), y que luego se ha girado un ángulo α, pasando a estar ahora en las coordenadas (X', Y', Z'). La coordenada Z no varía por lo tanto Z' = Z.

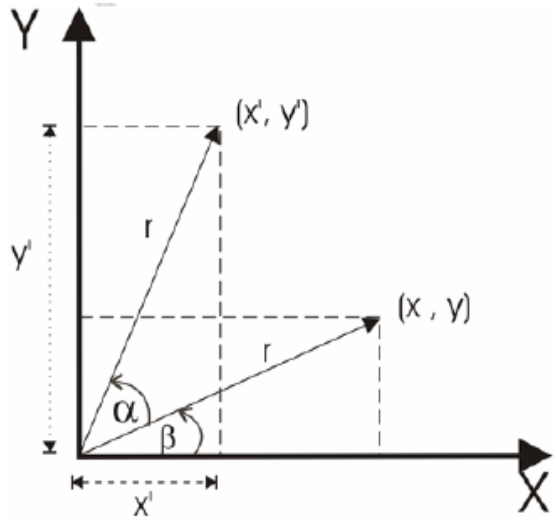

Figura 6. Giro en dos dimensiones.

La representación matricial de la rotación de un objeto con respecto a un eje de coordenadas dependerá del eje utilizado. Así a continuación se muestran las matrices de rotación correspondientes a cada uno de los ejes.

$$M_x(\theta) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & sin\theta & 0 \\ 0 & -sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$ Matriz de rotación alrededor del eje x

$$M_y(\theta) = \begin{pmatrix} \cos\theta & 0 & sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$ Matriz de rotación alrededor del eje y

$$M_z(\theta) = \begin{pmatrix} \cos\theta & sin\theta & 0 & 0 \\ -sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$ Matriz de rotación alrededor del eje z

Donde α representa el ángulo de rotación y la posición resultante del punto rotado será $P_2=(x_2,y_2,z_2)$.

6.2.2.2.2 Giros relativos respecto a ejes paralelos a los ejes de coordenadas

Antes de abordar la situación general para cualquier eje de coordenadas, se va a estudiar un caso relativamente sencillo y que servirá a la hora de generalizar a cualquier eje, es el caso de las rotaciones respecto a ejes paralelos al eje de coordenadas.

Para comprender mejor cómo se efectúan estos giros se tomará como ejemplo la siguiente figura, donde se muestra un objeto en un sistema de referencia local *X', Y', Z'*, cuyo eje *X'* es paralelo al eje *X* del sistema de coordenadas global. Para girar el objeto entorno a *X'* se ha de proceder como sigue:

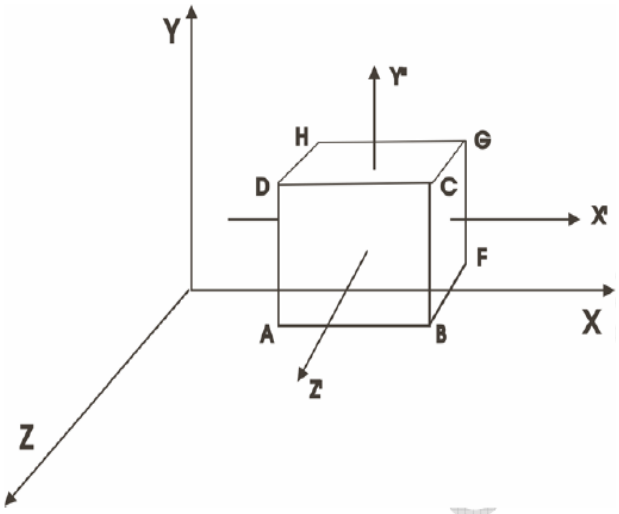

Figura 7. Giro sobre un eje paralelo al eje X.

1) Una vez establecida la ubicación del eje local de giro (*X'*), se ha de trasladar el objeto de forma que el eje de giro quede alineado con el eje global *X*. Suponiendo que un punto cualquiera del eje *X'* tiene las coordenadas globales *(xc,yc,zc,1),* la matriz que traslada el objeto de modo que el eje *X'* quede alineado con el *X* es

$$\mathbf{T} = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & -y_c & -z_c & 1 \end{vmatrix}$$

2) A continuación se gira el objeto sobre el eje *X* un ángulo θ, utilizando la matriz de giro correspondiente. Como se vio anteriormente, para ello se han de multiplicar todos los puntos significativos del objeto por la matriz, que en este caso es

$$\mathbf{G}_x = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta) & \sin(\theta) & 0 \\ 0 & -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

3) Finalmente se ha de trasladar el objeto a su posición original, utilizando la matriz de traslación inversa a la del paso 1), que no es otra que la

$$\mathbf{T}^{-1} = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & y_c & z_c & 1 \end{vmatrix}$$

La siguiente figura muestra como se realiza este proceso gráficamente.

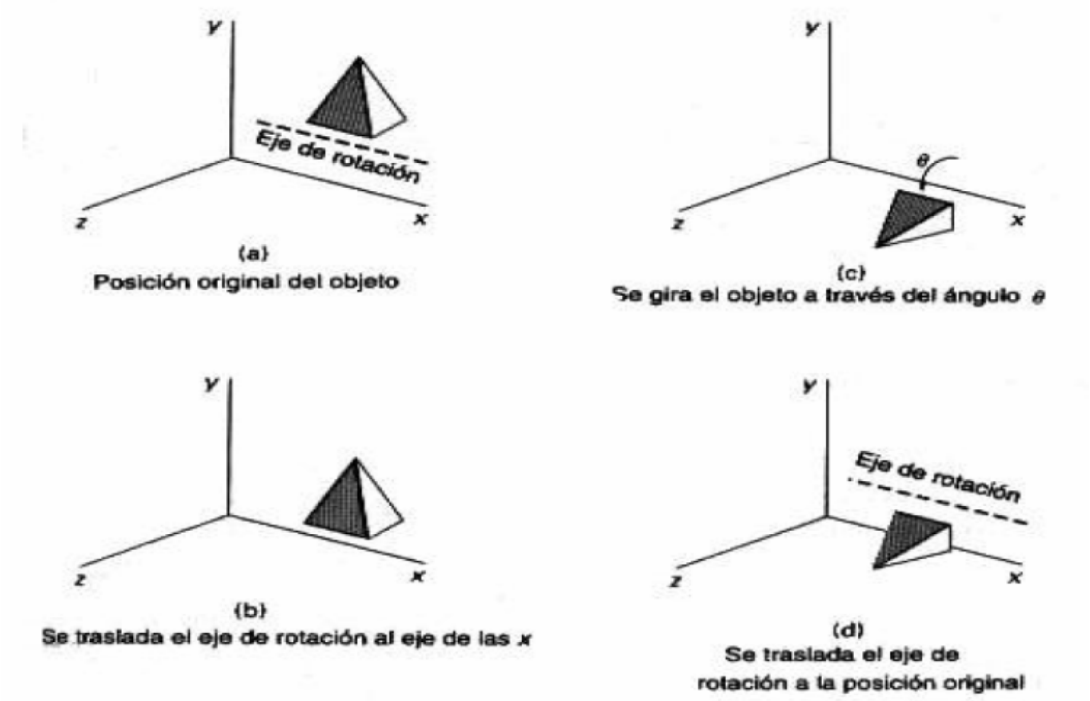

Figura 8. Proceso de giro sobre un eje paralelo al eje X.

El resultado final del giro sobre el eje X' sería algo similar a lo mostrado en esta figura

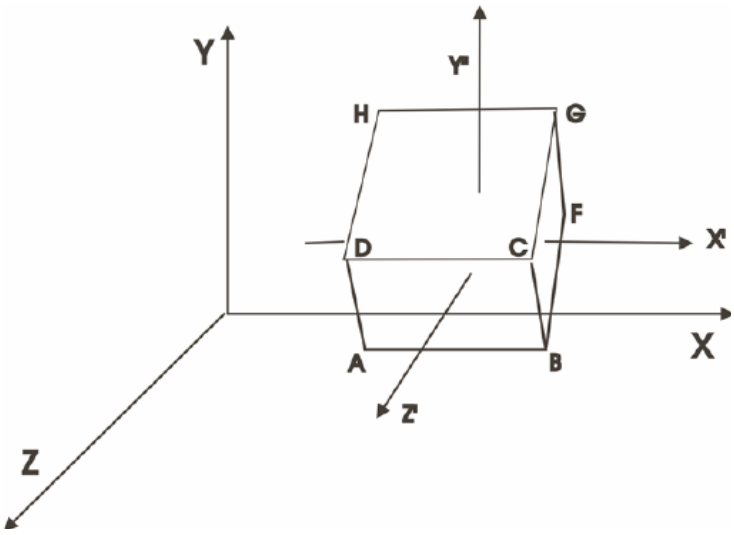

Figura 9. Ejemplo de giro.

Como el giro anterior requiere una secuencia de transformaciones lineales simples, el giro neto puede hallarse empleando una matriz compuesta, que se calcula multiplicando las anteriores, siguiendo el orden indicado. Ésta sería igual a

$$\mathbf{M} = \mathbf{T} \cdot \mathbf{G}_x \cdot \mathbf{T}^{-1} = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta) & \sin(\theta) & 0 \\ 0 & -\sin(\theta) & \cos(\theta) & 0 \\ 0 & y_c(1-\cos(\theta)) + z_c \sin(\theta) & z_c(1-\cos(\theta)) - y_c \sin(\theta) & 1 \end{vmatrix}$$

6.2.2.2.3 Giros relativos alrededor de un eje cualquiera

Los giros relativos estudiados hasta el momento no son suficientes para las necesidades de un sistema gráfico general, dado que se ha de tener la posibilidad de poder girar los objetos sobre un eje cualquiera.

Puesto que las herramientas matemáticas disponibles solamente son capaces de girar los objetos alrededor de los ejes ortogonales del sistema global de coordenadas, para poder girar un objeto sobre un eje cualquiera no queda más

remedio que alinear dicho eje con alguno de los ejes ortogonales del sistema de referencia.
Una vez establecida la ubicación y orientación del eje de giro, así como el ángulo de giro (θ), los pasos a seguir son:

1) Trasladar el objeto de forma que el eje de giro (del objeto) pase por el origen de coordenadas.

2) Girar el objeto de manera que el eje de rotación coincida con alguno de los ejes de coordenadas.

3) Realizar el giro indicado (θ).

4) Utilizar las matrices de giro inversas, y en orden inverso a como se hizo en el punto 2), para devolver el eje de giro a su orientación original.

5) Aplicar la matriz de traslación inversa a la utilizada en el punto 1), para devolver el eje de giro (y el objeto) a su posición original.

Antes de ver en detalle los pasos anteriores, estableceremos que los giros en el sentido de las agujas del reloj serán negativos. Además, el eje ortogonal que se utilizará para girar el objeto será el Z.

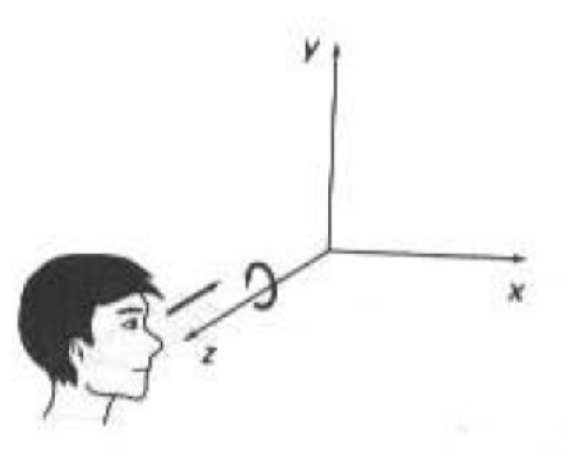

Figura 10. Sentido de giro con angulos positivos.

Previo al primer paso, se debe definir el eje de giro, este eje queda establecido mediante dos puntos *P = (x', y', z')* y *Q = (x", y",z")*, que definen un vector (*v*), que será normalizado dando paso al vector *u*.

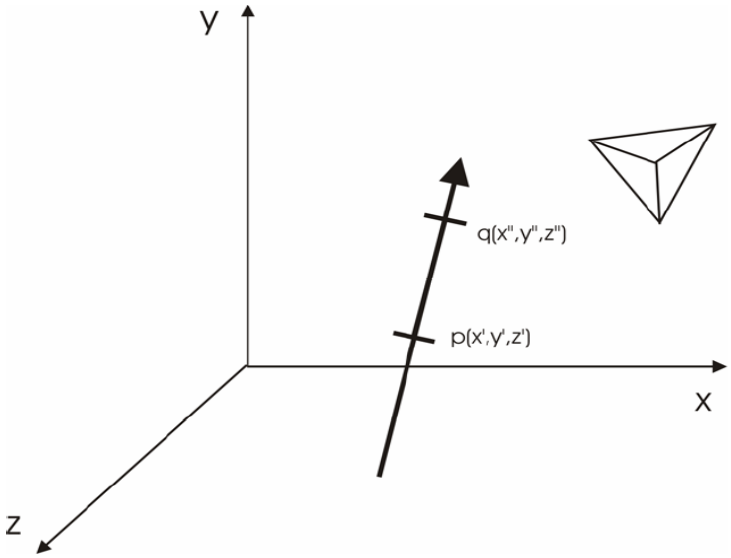

Figura 11. Normalización de un giro.

Ahora que se conoce el eje de giro normalizado (*u*), se verá paso a paso cómo se lleva a cabo el giro del objeto sobre este eje.

1) En primer lugar, se ha de utilizar una matriz de traslación sobre el objeto, de modo que el eje de giro pase por el origen de coordenadas (0, 0, 0).

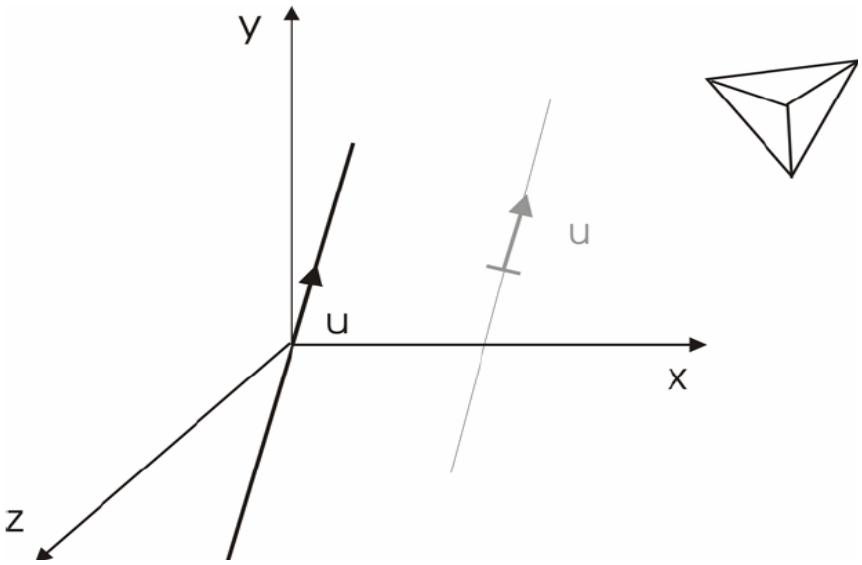

Figura 12. Traslación del eje al centro de coordenadas.

2) El siguiente paso es situar u sobre uno de los ejes de coordenadas, que en este caso será el Z. Esta operación se ha de realizar en dos fases:

a) Girar u alrededor del eje X, hasta el plano XZ.
b) Girar el vector u hasta el eje Z, alrededor del eje Y

3) Una vez situado el eje de giro sobre uno de los ejes de coordenadas, (en esta ocasión el eje Z) se realizará sobre dicho eje el giro establecido para el objeto, cuyo ángulo (en este caso) está indicado por θ. Por tanto, la matriz de giro en Z será la

$$\mathbf{G}_z(\theta) = \begin{vmatrix} \cos(\theta) & sen(\theta) & 0 & 0 \\ -sen(\theta) & \cos(\theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

4) Hecho el giro alrededor del eje Z, el objeto ya se encuentra girado el ángulo deseado, por lo que debemos volver el eje de giro a su orientación original. Para ello aplicaremos las matrices de giro inversas a las utilizadas en el paso 2), siguiendo el orden inverso.

5) Finalmente, una vez que el eje haya recuperado su orientación original, también se ha de llevar, junto con el objeto, a su posición inicial. Para ello se ha de aplicar la matriz de traslación inversa a la realizada en el paso 1).

Como se puede apreciar, el tiempo de cálculo de los giros sobre un eje cualquiera podría ser grande, pues al intervenir muchas matrices el total de multiplicaciones se dispara, a medida que crece el número de puntos significativos del modelo. Dicho tiempo puede reducirse notoriamente utilizando la matriz compuesta, la cual se obtiene multiplicando las matrices que aparecen, y en el mismo orden. Por tanto, la matriz neta para el giro en un eje cualquiera será

$$\mathbf{G}(\theta) = \mathbf{T} \cdot \mathbf{G}_x(\varphi) \cdot \mathbf{G}_y(-\alpha) \cdot \mathbf{G}_z(\theta) \cdot \mathbf{G}_y^{-1}(-\alpha) \cdot \mathbf{G}_x^{-1}(\varphi) \cdot \mathbf{T}^{-1}$$

6.2.2.3 Cambios de escala

Dentro de un espacio de referencia los objetos pueden modificar su tamaño relativo en uno, dos, o los tres ejes. Para ello se ha de aplicar la matriz de escalado, las nuevas coordenadas de cada uno de los puntos, viene dada por la expresión:

$$\begin{pmatrix} x_2 \\ y_2 \\ z_2 \\ 1 \end{pmatrix} = \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ y_1 \\ z_1 \\ 1 \end{pmatrix}$$

Donde los factores de escala s_x,s_y,s_z, representan el grado en el que se aumentan las coordenadas del punto original con respecto a cada uno de los ejes, toman siempre valores positivos y no necesariamente iguales.

Las coordenadas del punto final serán x' = xSx, y' = ySy, z' = zSz; Sx, Sy, Sz deberán ser números reales positivos.

Cuando Sx = Sy = Sz el cambio de escala es uniforme; en cualquier otro caso el cambio de escala será no uniforme.

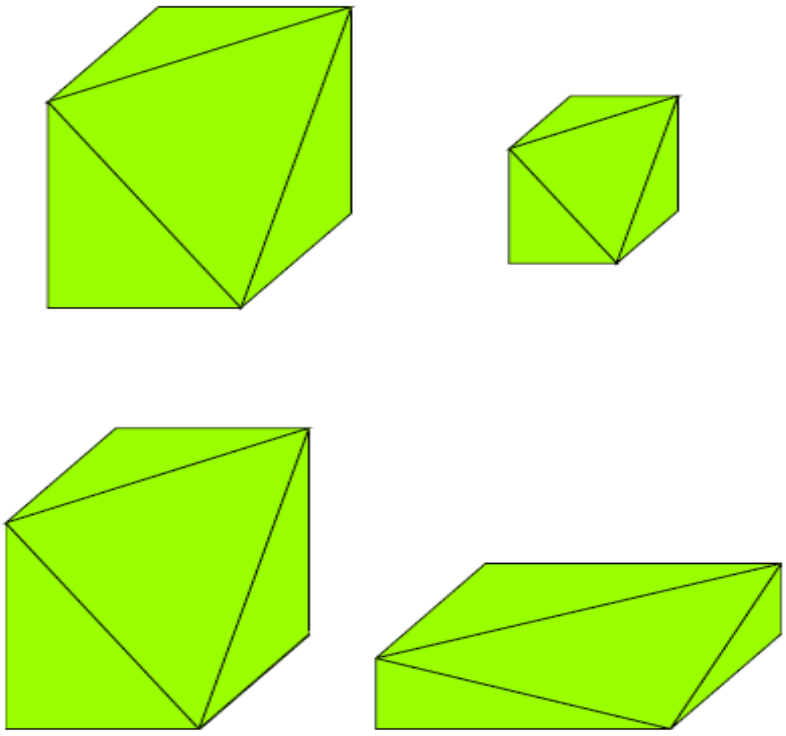

Figura 13. Escala uniforme y no uniforme.

Si no se toman precauciones, los cambios de escala, además de suponer una variación en las proporciones de los objetos, también implican una traslación de los mismos, un efecto colateral no deseado en muchas ocasiones.

Para evitar este efecto no deseado, la escala se realiza utilizando un cambio de escala respecto a un punto fijo, esta técnica consiste en escalar un objeto, sin que éste se vea afectado por un cambio de posición, respecto a un punto establecido.

Los pasos necesarios se explican a continuación:

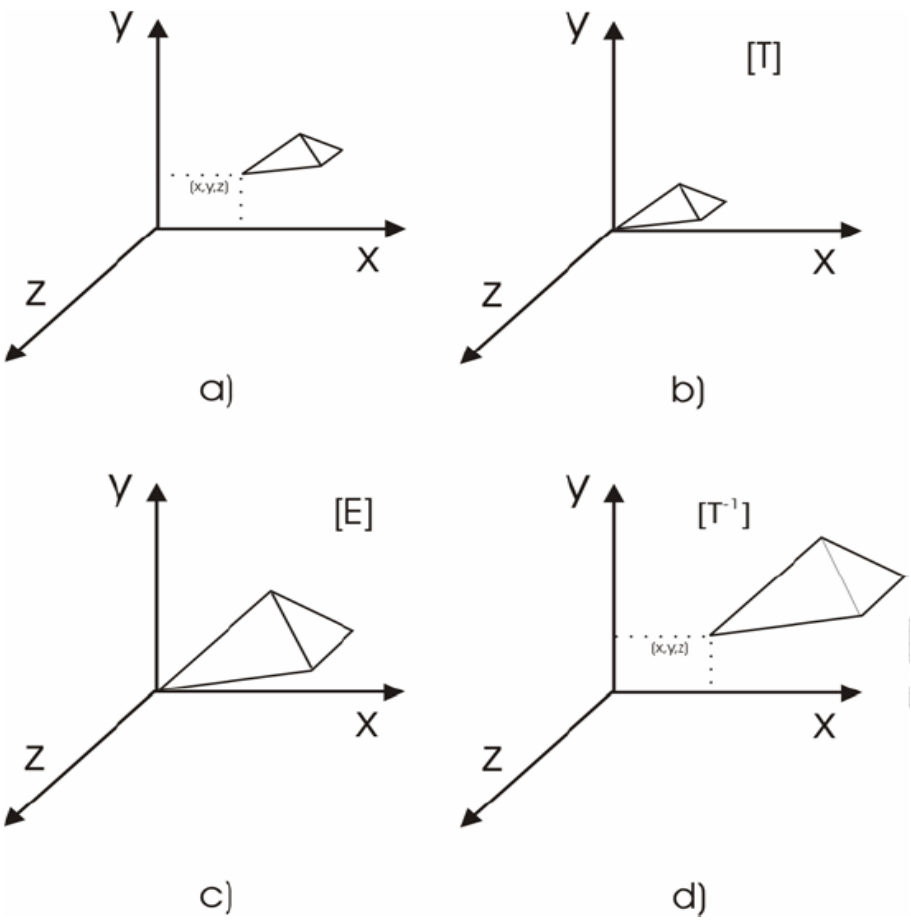

Figura 14. Pasos para realizar la escala.

a) En primer lugar se aplica una transformación al punto de referencia, trasladándolo al eje de coordenadas. Como ya hemos repetido en varias ocasiones, éste proceso en realidad consiste en desplazar los ejes de coordenadas de forma que el origen de los mismos coincida con el punto de referencia.

b) A continuación tienen lugar las transformaciones de escala específicas para un punto individual o para cada uno de los puntos que definen el objeto. Esto se lleva a cabo efectuando la operación matricial anteriormente expuesta.

c) Por último, es necesario deshacer la transformación realizada en el primero de los pasos, utilizándose para ello la matriz de traslación inversa. Con ello se consigue que los ejes de coordenadas vuelvan a su posición original y el objeto quede escalado de forma apropiada.

6.2.2.4 Otras transformaciones

6.2.2.4.1 Reflexiones

Algunas orientaciones deseables para los objetos tridimensionales no pueden ser obtenidas usando solamente giros. Con la reflexión se consigue un efecto "espejo", de modo que los objetos se ven reflejados en un plano.

Una reflexión tridimensional es una transformación que se puede realizar con respecto a un eje de reflexión o con respecto a un plano de reflexión. En el primero de los casos, la reflexión de un elemento sobre un eje es análoga a una rotación de 180º de dicho elemento alrededor de dicho eje.

De igual forma, la reflexión con respecto a un plano es equivalente a una rotación de 180º pero esta vez sobre un espacio cuatridimensional. Concretamente, cuando se utiliza como plano de reflexión el plano XY o cualquier otro plano definido por los ejes de coordenadas, el resultado obtenido es similar a intercambiar la especificación de las coordenadas del sistema del lado derecho y el izquierdo. Esto se puede comprobar fácilmente observando

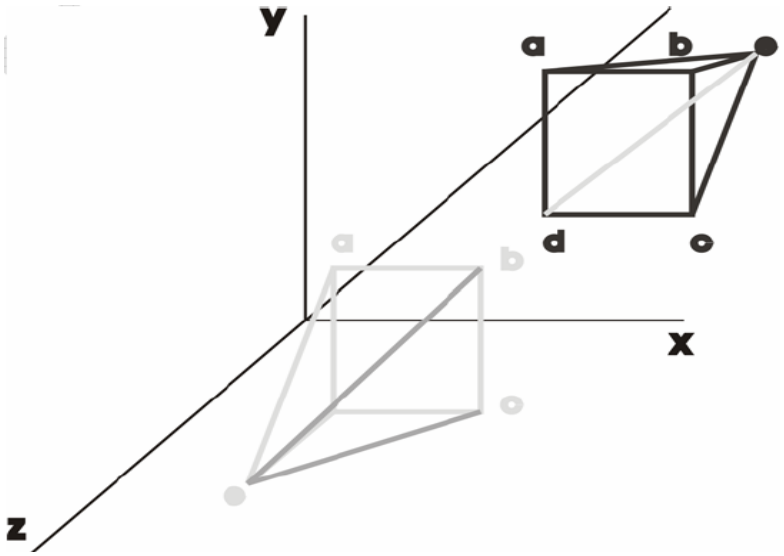

Figura 15. Reflexión en el plano XY.

Cuando la reflexión se hace sobre uno de los planos ortogonales ($x = 0$, o $y = 0$, o bien $z = 0$) la matriz de transformación es sencilla, pues es similar a la *matriz* identidad, aunque siendo –1 el elemento que representa a la coordenada que es nula en el plano de reflexión. Así, las matrices de reflexión para el plano *XY* es

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Se definen de forma similar para los planos *XZ YZ*. El proceso de reflexión se resume en los siguientes puntos:

a) Trasladar el punto establecido del plano al origen de coordenadas.

b) Realizar los giros oportunos para hacer coincidir el vector normal al plano de reflexión con uno de los ejes de coordenadas; así el problema se reduce a una simple reflexión sobre alguno de los planos del sistema de referencia. Por ejemplo, si el eje escogido es el *Z*, el plano de reflexión sería el
XY.

c) Realizar la reflexión sobre el plano seleccionado.

d) Aplicar las transformaciones inversas para devolver el plano de reflexión a su posición original.

6.2.2.4.2 Recorte

Para alterar la forma de un objeto se utiliza una transformación conocida como recorte. En espacios tridimensionales, también se utiliza para realizar transformaciones de proyección.

Matemáticamente, la operación de recorte de un objeto sobre el eje Z, viene definida por la siguiente matriz:

$$\begin{pmatrix} 1 & 0 & a & 0 \\ 0 & 1 & b & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Los parámetros a y b pueden tomar cualquier valor y el efecto obtenido con esta transformación es una alteración de los valores de las coordenadas X e Y del objeto, en una cantidad proporcional al valor de la coordenada Z, mientras que dicha coordenada Z permanece inalterable. Este efecto se puede observar en la en la siguiente figura, en la que se tiene un cubo de arista 1 y un valor para a y b igual a 1.

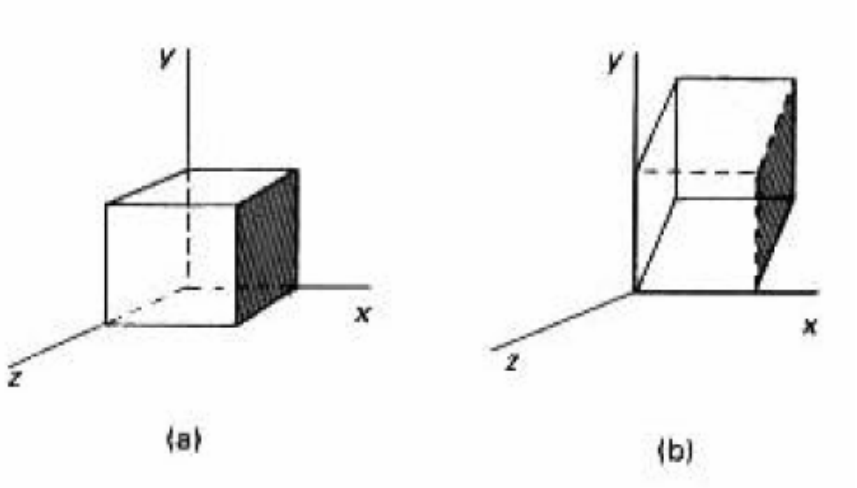

Figura 16. Recorte de un cubo.

6.2.2.4.3 Transformaciones deformantes

Las transformaciones descritas hasta este punto eran transformaciones lineales. Ahora se van a describir brevemente las transformaciones deformantes, que hacen que un objeto se estreche, se contorsione o se doble.

Para conseguir que un objeto se estreche a lo largo de un eje (el Z, por ejemplo), hay que aplicar a ese objeto un cambio de escala no uniforme. Así, a cada vértice del objeto se le ha de aplicar el cambio de escala definido por

$$x' = rx,\ y' = ry,\ z' = z$$

donde $r = f(z)$.

Esto significa que las coordenadas z de los vértices del objeto no variarán, mientras que las coordenadas x e y dependerán de los valores de z, ya que están multiplicadas por una función de z.

La contorsión de un objeto se consigue aplicándole una rotación sobre uno de los ejes de coordenadas, por ejemplo el z, con un ángulo α. Este ángulo α no tendrá un valor fijo si no que será una función de la variable del eje sobre el que se gira, en este caso $\alpha = f(z)$.

Para doblar un objeto a lo largo de un eje se ha de dividir el mismo en dos regiones, aplicando en cada una transformaciones diferentes.

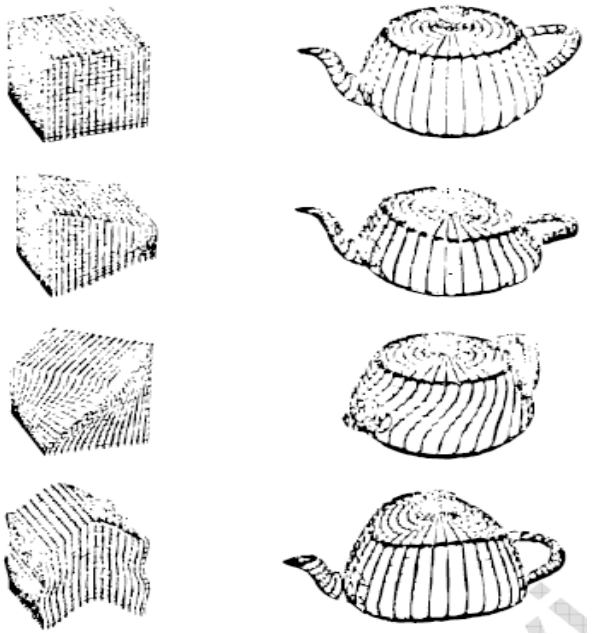

Figura 17. Ejemplo de deformaciones.

Al aplicar una transformación deformante a un objeto, por no ser esta lineal puede afectar a las características topológicas del objeto, es decir, a la relación entre vértices, aristas y caras, perdiéndose información. Esto podría dar lugar a fallos en la renderización, por lo que la resolución del objeto posiblemente sería inferior, o bien, si el número de polígonos que forman el objeto es pequeño, puede que no fuese posible visualizar el objeto resultante.

Por lo general, las transformaciones deformantes no son procesos fáciles, por lo que para implantarse se aconseja acudir a libros especializados.

6.2.3 Importancia del orden de aplicación de las transformaciones

Cuando se encadenan varias transformaciones, es especialmente importante el orden en el que éstas se aplican, ya que la escena resultante será diferente en muchos casos, especialmente cuando tienen lugar rotaciones y traslaciones. Esta diferencia se puede observar contrastando las dos partes de la Figura. Por un lado, la siguiente figura muestra el resultado de aplicar una transformación compuesta por una rotación y una traslación sobre un objeto en ese orden. En dicha figura, al rotar el objeto, se rota el sistema de coordenadas y la traslación siguiente se efectúa sobre el sistema girado. El efecto resultante es distinto, como se puede apreciar a continuación en la figura primero se traslada el objeto, con lo que se traslada el sistema de coordenadas y a continuación se rota, produciendo una rotación del objeto.

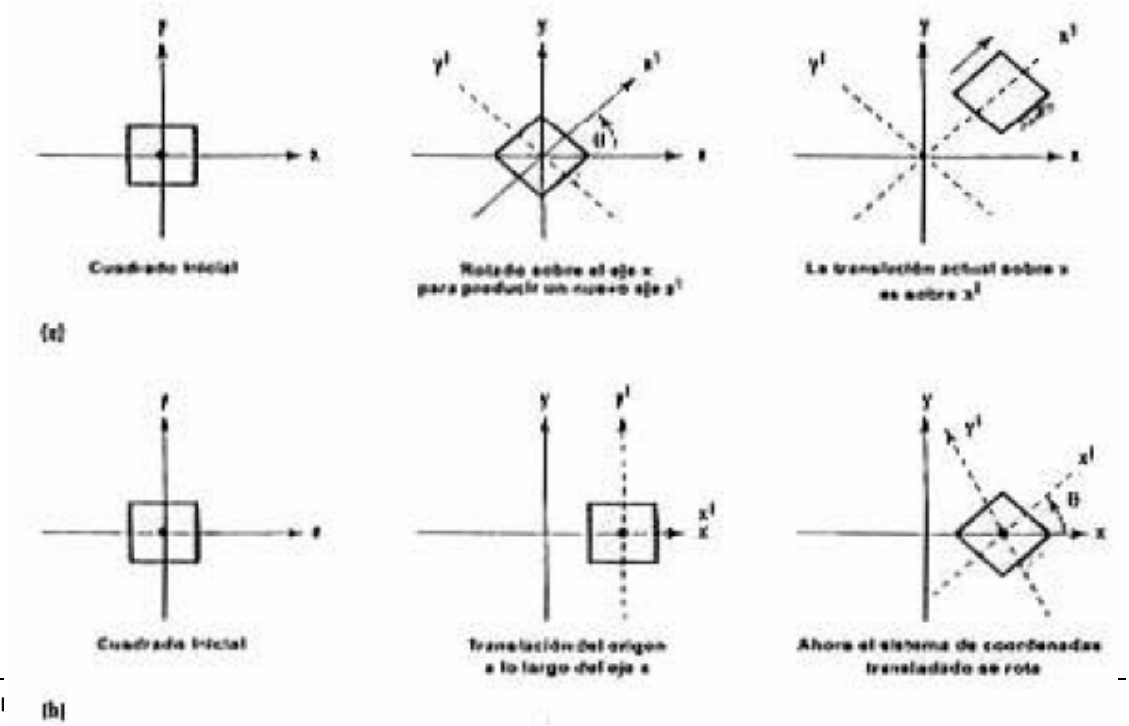

Figura 18. Efectos rotación-translación y translación - rotación.

6.2.4 Utilización de matrices para representar las transformaciones en OpenGL

En la introducción de los diversos conceptos matemáticos para realizar transformaciones se ha visto la importancia del manejo de matrices y de diferentes operaciones, destacando sobre todas ellas la multiplicación.

La librería OpenGL intenta facilitar esta labor y conseguir el efecto deseado, OpenGL dispone de un conjunto de primitivas que simplifican este proceso.

En concreto, para aplicar una transformación a un elemento, por ejemplo, un cubo, será necesario aplicar una transformación a cada uno de los vértices que lo definen, multiplicando las coordenadas de dichos vértices por la matriz de transformación adecuada.

Para conseguir los efectos perseguidos con la utilización de las transformaciones, se utilizan dos matrices particulares, sobre las que se realizan modificaciones: la matriz del modelador y la matriz de la proyección.

La librería OpenGL implementa un conjunto de primitivas de alto nivel que facilitan la realización de estas transformaciones sin necesidad de acceder a los elementos individuales de cada una de las matrices implicadas y simplificando, en gran medida, todo el proceso.

El proceso de transformación de un vértice y su posterior visualización en pantalla comprende una serie de pasos que se detallan en la figura que se encuentra a continuación. Dicho vértice tendrá unas coordenadas iniciales *x, y, z* que describen su posición en los ejes, y una cuarta coordenada *w* llamada factor de escala, que por defecto vale 1, y cuya modificación produce el mismo efecto que una transformación de escalado del vértice.

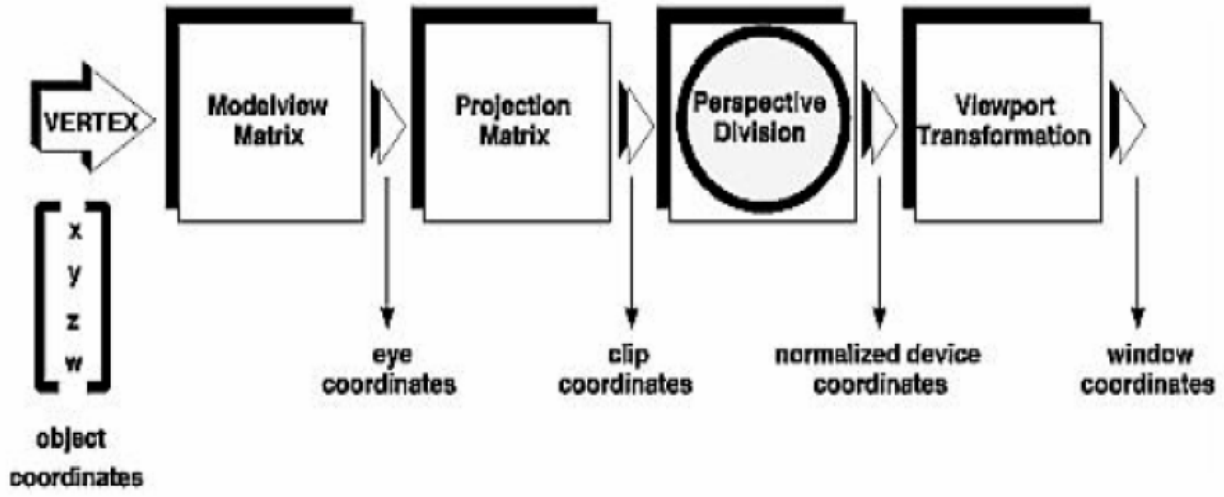

Figura 19. Proceso de transformación de un vértice en coordenadas 2D .

El primer paso del proceso indicado anteriormente, consiste en obtener las coordenadas oculares del vértice transformado. Para ello, se multiplican las coordenadas iniciales del vértice por la matriz del modelador, que como se explica posteriormente, representa al sistema de coordenadas transformado que se utiliza en cada momento para colocar los objetos que componen una escena.

A continuación, dichas coordenadas oculares se multiplican por la matriz de proyección, obteniéndose las llamadas coordenadas de trabajo, eliminándose de esta forma los elementos que quedan fuera del volumen de visualización.

Posteriormente, se dividen las coordenadas *x, y, z* obtenidas entre el factor de escala *w*, cuyo valor dependerá de las transformaciones aplicadas con anterioridad. De esta forma se reduce a tres el número de coordenadas del vértice, que definen el mismo en tres dimensiones.
Por último, y para hacer posible la representación del vértice en un espacio bidimensional, se realiza la transformación de la vista, obteniéndose las coordenadas x, y que definen la posición del vértice en la pantalla. Este último paso, es realizado internamente por OpenGL, a partir de los parámetros de la primitiva *glViewPort ().* La sintaxis de dicha función es la siguiente:

```
glViewPort (GLint x, GLint y, GLsizei ancho, GLsizei alto)
```

6.2.4.1 Matriz de modelador

Como se ha descrito en apartados anteriores, el proceso para transformar un vértice no consiste en modificar sus coordenadas con respecto al sistema de coordenadas utilizado, sino en transformar dicho sistema de coordenadas, manteniendo los valores originales que definen el punto. Por ejemplo, si se desea trasladar el punto (1, 1, 1) diez unidades en el eje X, no se modifican sus coordenadas a (11, 1, 1), sino que es el sistema de coordenadas el que se traslada 10 unidades en dicho eje, manteniéndose los valores (1, 1, 1) para el punto.

La principal ventaja que se consigue con la utilización de este método es la reducción del número de operaciones necesarias para realizar una transformación. Esto se puede ver claramente con el siguiente ejemplo:
Supongamos un sistema definido por 26 vértices. En caso de trasladar cada uno de los vértices, sería necesario realizar un total de 26 multiplicaciones, mientras

que utilizando este método, tan sólo se requeriría la traslación del sistema de coordenadas, manteniéndose constante la posición de los 26 vértices de la figura.

En este sentido, la matriz del modelador será una matriz cuadrada (4x4) que se encargará de definir el nuevo sistema de coordenadas, que se utilizará para representar los objetos que han sufrido una transformación. Al multiplicar dicha matriz por la matriz columna que representa el vértice, se obtiene otra matriz columna, que representa las nuevas coordenadas del vértice transformado.
La ecuación siguiente muestra este proceso, en el cual, además de las coordenadas x, y, z se tiene en cuenta un factor de escala w.

$$\begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} (M) = \begin{pmatrix} x^e \\ y^e \\ z^e \\ w^e \end{pmatrix}$$

La utilización de la matriz del modelador se justifica porque a medida que se van realizando transformaciones, dicha matriz se va modificando, de forma que va acumulando los efectos de las sucesivas transformaciones. Por ejemplo, si deseamos colocar dos elementos en las posiciones (0, 1, 0) y (1, 0, 0) respectivamente, a primera vista se podría pensar en trasladar el sistema de coordenadas una unidad hacia arriba en el eje Y, dibujar el primer elemento y, a continuación, trasladar el sistema una unidad hacia la derecha en el eje X y dibujar el segundo elemento. Sin embargo, como puede comprobarse en la figura dibujada a continuación, ésta no sería la solución acertada, ya que al acumularse los efectos de las transformaciones, el segundo elemento quedaría dibujado en las coordenadas (1, 1, 0). Una posible solución a este problema consistiría en trasladar el sistema de coordenadas una unidad hacia abajo en el eje Y, después de dibujar el primer elemento. Con ello se consigue que el sistema de coordenadas quede en la posición inicial, con lo cual sería necesario trasladar el eje de coordenadas una unidad hacia la derecha en el eje X y dibujar finalmente el segundo elemento.

Sin embargo, esta solución puede resultar extremadamente complicada e inviable en el caso de que el número de transformaciones a realizar sea elevado, ya que habría que tener presente, en todo momento, la posición del sistema de coordenadas.

Una solución más factible sería almacenar en la matriz del modelador un estado conocido del sistema de coordenadas. En el caso del ejemplo anterior, se reiniciaría la matriz al origen del sistema de coordenadas. Para ello, se utiliza la

matriz identidad (matriz cuyos elementos son ceros excepto los que ocupan su diagonal principal, que son unos), con la cual se carga la matriz del modelador. Al multiplicar esta matriz por cualquier vértice, se obtiene el mismo vértice, y como las coordenadas del mismo se mantienen siempre (ya que lo que se traslada es el sistema de coordenadas), el resultado es la representación del vértice con respecto al sistema de coordenadas inicial. Las primitivas utilizadas para cargar la matriz identidad en la matriz del modelador son `glMatrixMode()` y `glLoadIdentity()`, cuya utilidad se puede comprobar en el siguiente fragmento de código del programa planets.c:

```
/*
Funcion de actualizacion en caso del cambio de tamaño de la ventana. La primera vez que
se llama ademas define el volumen de visualizacion
*/
void reshape ( int w, int h )
{
        // se ajusta la vista a las dimensiones de la ventana
        glViewport ( 0, 0, w, h );
         // se actua sobre la matriz de la proyeccion
        glMatrixMode ( GL_PROJECTION );
         // se carga en ella la matriz identidad (estado inicial)
         glLoadIdentity ( );
         // se define una proyección perspectiva
         if ( h==0 )
                gluPerspective ( 80, ( float ) w, 1.0, 5000.0 );
        else
                gluPerspective ( 80, ( float ) w / ( float ) h, 1.0, 5000.0 );
        // se actua sobre la matriz del modelador
        glMatrixMode ( GL_MODELVIEW );
         // se carga en la matriz del modelador la identidad (estado inicial)
         glLoadIdentity ( );
         vp_centre_x = w / 2;
         vp_centre_y = h / 2;
 }
```

La salida generada por el programa anterior se puede observar en la Figura

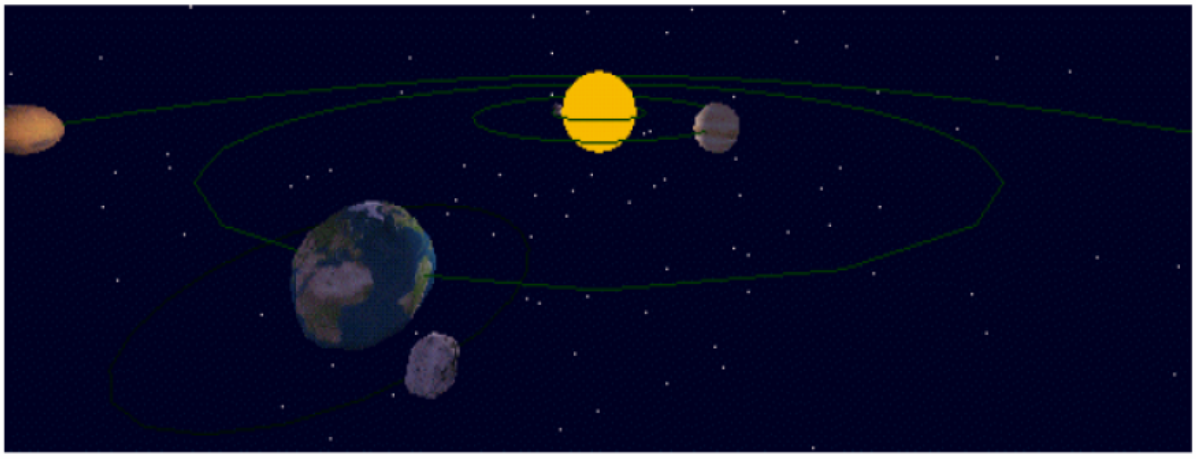

Figura 20. Salida del programa planets.c.

6.2.4.2 Utilización de primitivas de OpenGL para realizar transformaciones

A continuación se describen, de forma detallada, los pasos que se deben dar para generar las transformaciones explicadas anteriormente haciendo uso de la librería OpenGL. Para cada una de ellas se presentarán las primitivas que pueden resultar útiles y se acompañará de ejemplos.

6.2.4.2.1 Traslación

Una traslación provoca un cambio en la posición de un elemento con respecto al sistema de coordenadas utilizado. Se podría pensar que para trasladar un objeto hasta una posición deseada, sería necesario construir una matriz como las vistas anteriormente que representase dicha traslación, y aplicarla sobre la matriz del modelador.

Para facilitar este proceso y hacer el código más legible, OpenGL contiene una primitiva que realiza todo este proceso internamente. En concreto se trata de la primitiva `glTranslatef()`, cuya sintaxis es la siguiente:

```
glTranslatef (GLfloat x, GLfloat y, GLfloat z)
```

La función recibe como parámetros la medida en que el sistema de coordenadas actual se trasladará sobre cada uno de los ejes (recordemos que lo que realmente se traslada es el sistema de coordenadas y no el objeto).

Así por ejemplo, para dibujar un triángulo desde el punto (-20, 0, 0) sería necesario mover primero el sistema de coordenadas a dicho punto, esto es 20 puntos en el sentido negativo del eje X, y a continuación dibujar el triángulo. Utilizando la primitiva anterior, tendríamos:

```
// se mueve el sistema de coordenadas
glTranslatef (-20.0, 0.0, 0.0);
// se dibuja el triángulo
draw_triangle();
```

6.2.4.2.2 Rotación

La rotación de un elemento es un cambio en la orientación de un elemento con respecto al sistema de coordenadas utilizado. Para llevar a cabo una rotación de un elemento, ya hemos visto que sería necesario construir una matriz que caracterizara dicha transformación y multiplicarla por la matriz del modelador.

Con ayuda de la primitiva `glRotatef()` proporcionada por la librería OpenGL, este proceso se hace de forma transparente al programador, que no se tiene que encargar de mantener ningún tipo de matriz ni de realizar a mano una engorrosa multiplicación. La sintaxis para esta primitiva es bastante similar a la primitiva utilizada en la traslación, concretamente:

```
glRotatef(GLfloat angulo, GLfloat x, GLfloat y, GLfloat z)
```

En este caso los parámetros *x,y,z* representan las coordenadas que definen el eje de rotación. En el caso de que se quiera girar el objeto respecto a uno de los ejes de coordenadas, el valor correspondiente a los otros dos ejes será cero. Además, se puede apreciar que en este caso se incluye un parámetro adicional que representa el número de grados que se girará el sistema alrededor del eje especificado. Como ya se ha descrito, los valores positivos para este parámetroproducen giros en sentido anti-horario.

Por ejemplo, para girar el triángulo anterior 90 grados con respecto al eje *Z* necesitaríamos escribir el siguiente código:

```
// se rota el sistema de coordenas 90 grados
glRotatef (90.0, 0,0, 0.0, 1.0);
// se dibuja el triángulo.
```

```
draw_triangle();
```

Pero no es preciso asociar el eje de rotación con uno de los ejes de coordenadas, sino que se puede especificar cualquier línea como eje de rotación. Dicho eje vendrá definido por dos puntos: uno el origen de coordenadas y otro, el definido por los parámetros *x,y,z*.

Concretamente, la llamada siguiente rota una escena 45 grados alrededor del eje que une el (0,0,0) con el punto (1,2,3):

```
glRotatef (45.0f, 1.0f, 2.0f, 3.0f);
```

6.2.4.2.3 Escalado

El escalado modifica las dimensiones de un objeto, de forma que se aumentan o disminuyen dichas dimensiones en función de uno o varios parámetros de escala. Una vez más, la librería OpenGL cuenta con una primitiva que nos ahorra tener que realizar complicadas operaciones matriciales como vimos anteriormente. En concreto, esta primitiva es `glScalef()` y su sintaxis es la siguiente:

```
glScalef (GLfloat a, GLfloat b, GLfloat c)
```

Esta función multiplica las coordenadas iniciales *x,y,z* del objeto representado por los factores de escala *(a,b,c)* que recibe como parámetros, de forma que las coordenadas resultantes de cada uno de los puntos que definen el objeto serán *(x*a, y*b, z*c).*

Si los valores utilizados para *a, b y c* son iguales, se dice que el escalado es uniforme y el resultado es una ampliación o disminución del tamaño del objeto que se representa, manteniéndose su estructura. En caso contrario, se habla de escalado no uniforme, y el resultado es un objeto deformado en función de los factores de escala para cada uno de los ejes.

Por ejemplo, el siguiente código muestra nuestro triángulo deformado con la mitad de su altura y el doble de su ancho originales gracias a una escalación no uniforme en los ejes X e Y:

```
// se efectúa el cambio de escala
glScalef (1.5, 0.5, 1.0);
// se dibuja el triángulo
draw_triangle();
```

6.2.4.3 Pilas de matrices

Como se planteo en apartados anteriores, los efectos de las transformaciones son acumulativos, con lo que, a menudo, será necesario reiniciar la matriz del modelador al estado inicial para facilitar la comprensión de las mismas.

Sin embargo, no es necesario realizar siempre este proceso, ya que es posible mantener cada uno de los efectos que las transformaciones tienen en el sistema de coordenadas. Esto es especialmente útil en el caso de que se quieran recolocar objetos con respecto a estados anteriores y volver al estado actual. Para ello se utiliza una pila de matrices como la que se muestra en la figura, en la que se van almacenando de manera sucesiva las matrices que definen el sistema de coordenadas en cada momento. De esta forma, se puede insertar la matriz actual en la pila y aplicarle modificaciones, pudiendo recuperarla posteriormente, tal y como se muestra:

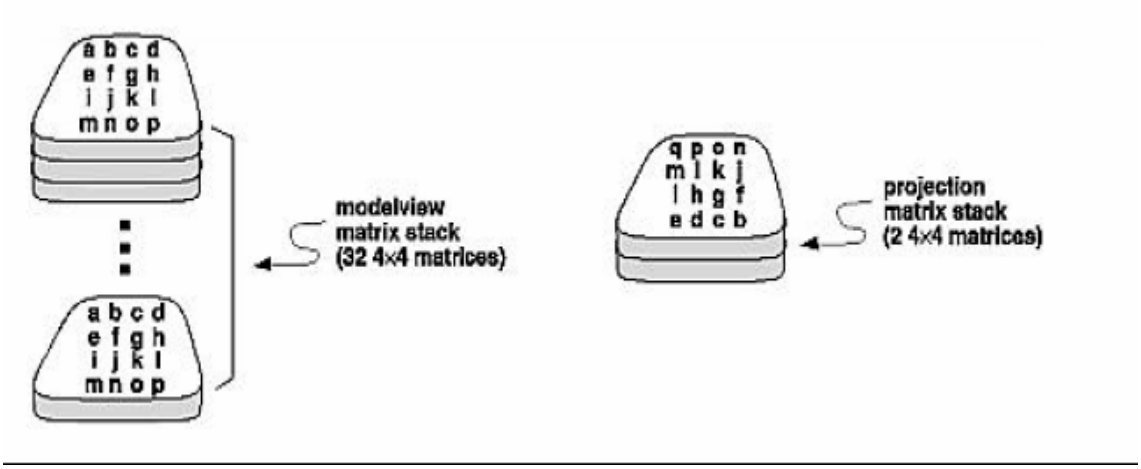

Figura 21. Pila de matrices.

En realidad, OpenGL dispone de una pila para matrices del modelador y de otra para matrices de proyección, junto con una pila adicional para matrices de textura.

Las operaciones básicas que se pueden realizar sobre la pila de matrices son las siguientes:

- Se puede extraer un elemento (matriz) de la pila mediante una llamada a la primitiva `glPopMatrix()`. Esta primitiva, que no contiene parámetros, devolverá un valor `GL_STACK_UNDERFLOW`, en el caso de que no sea posible la extracción al no existir elementos en la pila.

- Se puede insertar un elemento (matriz) en la pila mediante una llamada a la rutina `glPushMatrix()`, que tampoco recibe ningún parámetro. Devolverá un valor de error, concretamente `GL_STACK_OVERFLOW`, en el caso de que la pila haya alcanzado su número máximo de elementos permitidos, no pudiéndose realizar la última inserción.

El número de elementos que se puede almacenar en la pila varía en función del sistema utilizado. Concretamente, la versión de Microsoft permite un máximo de 32 elementos para la pila de matrices del modelador y 2 elementos para la pila de matrices de proyección. En cualquier caso, se puede obtener dicho valor con las primitivas

- `glGet(GL_MAX_MODELVIEW_STACK_DEPTH)` y
- `glGet(GL_MAX_PROJECTION_STACK_DEPTH)`.

Un ejemplo de la utilización de pilas de matrices para la representación de modelos compuestos, se puede observar en el siguiente fragmento de código procedente del programa rotate.c

```
/* Dibuja las esferas en la pantalla. Una rota sobre el eje
que se ha seleccionado. La otra rota sobre su propio centro.
*/
void drawSphere (void)
{
/* Se guarda la matriz del modelador actual en la pila*/
glPushMatrix ();
/* Se aplican las transformaciones al modelo: girar la
primera esfera */
glRotatef (rotate, axis[0], axis[1], axis[2]);
glTranslatef (-15.0, 10.0, 5.0);
glColor3f (1.0, 0.0, 0.0);
glutWireSphere (5.0, 6.0, 6.0);
/* Se recupera la matriz del modelador original de la pila
para que las transformaciones ya realizadas no afecten a las
posteriores */
glPopMatrix ();
/* Se repite el proceso con la esfera que rota sobre si misma
*/
glPushMatrix ();
glTranslatef (5.0, 10.0, 0.0);
glRotatef (rotate, axis[0], axis[1], axis[2]);
glColor3f (0.0, 1.0, 0.0);
glutWireSphere (2.0, 6.0, 6.0);
glPopMatrix ();
}
```

La Figura muestra la escena de salida del ejemplo anterior:

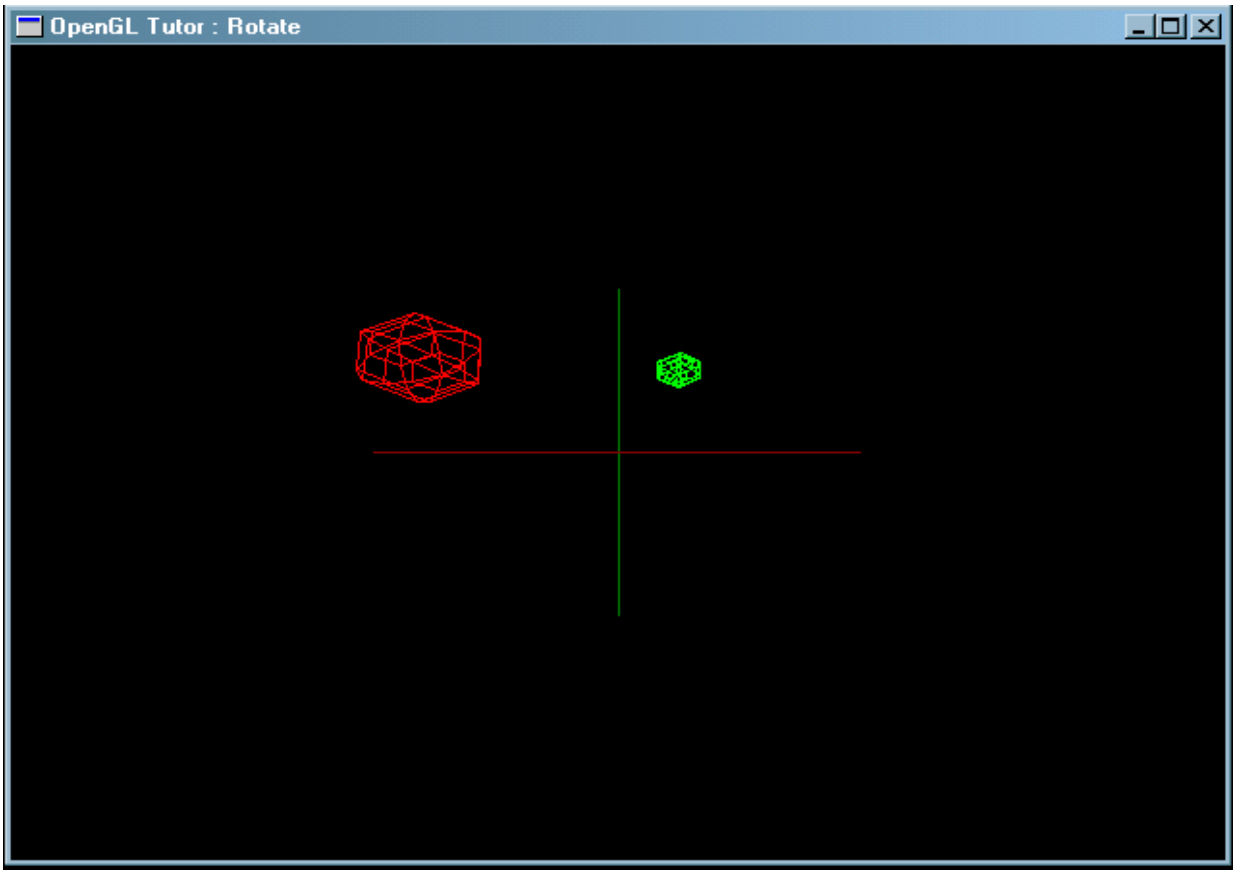

Figura 22. Salida del programa rotate.c.

6.3 Proyecciones

El proceso de visión en tres dimensiones es mas complejo que en dos dimensiones, en dos dimensiones solamente seria necesario definir una ventana y un marco, si embargo en tres se debe realizar una transformación de tres a dos dimensiones llamada proyección.

Existen dos métodos principales de proyección capaces de generar vistas de los objetos sólidos (tridimensionales):

- Proyección paralela
- Proyección perspectiva

En la siguiente figura se muestran las relaciones entre los distintos tipos de proyecciones.

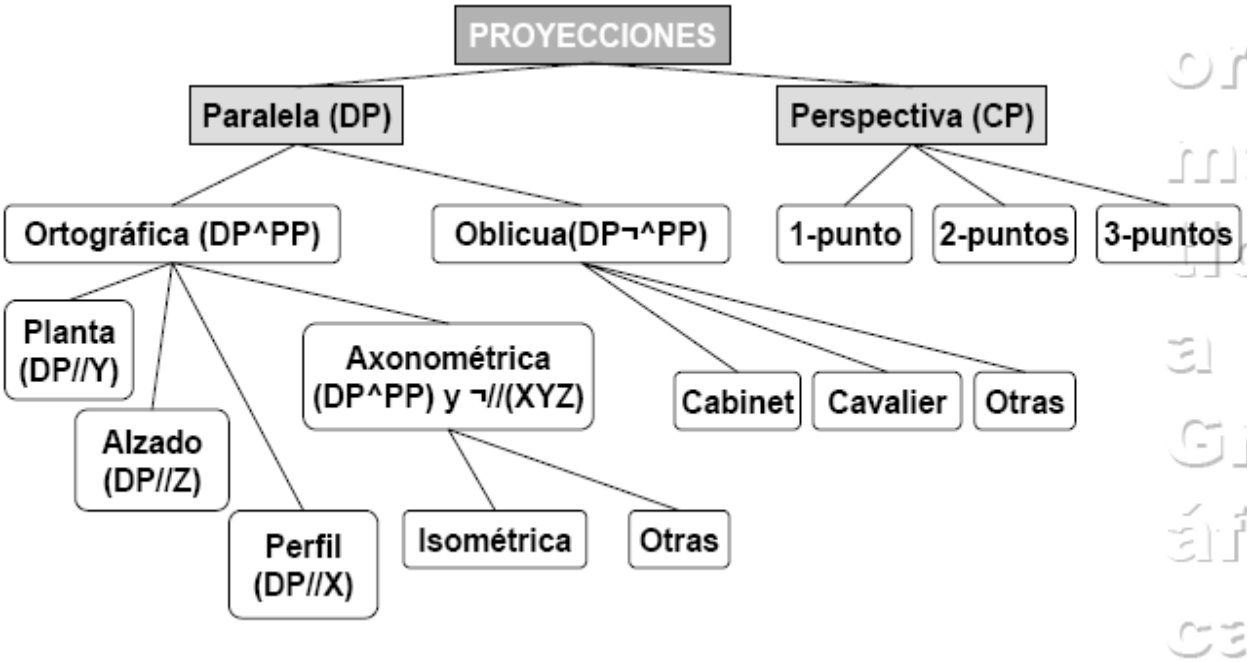

Figura 23. Relación entre los diferentes tipos de proyecciones.

En los siguientes apartados se hará un estudio exhaustivo de cada una de ellas, aunque se puede dar una pequeña noción con la siguiente figura donde en la primera figura se muestra una proyección perspectiva y en b una paralela.

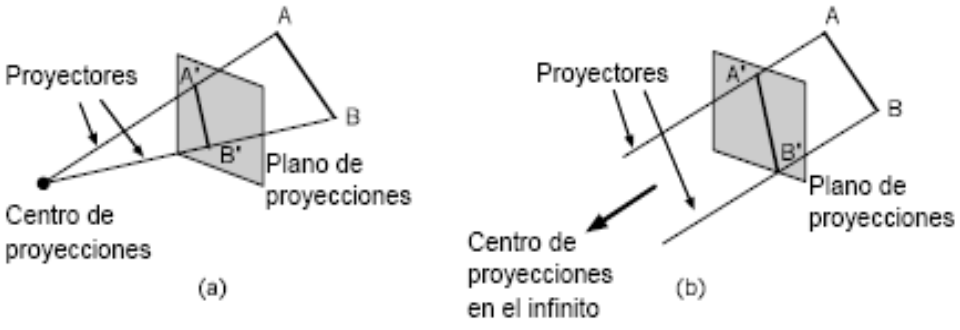

Figura 24. A) Proyección perspectiva, B) Proyección paralela.

6.3.1 Proyección paralela

Una proyección paralela define un volumen de visualización rectangular, concretamente un paralelepípedo de tamaño infinito. A diferencia de la proyección perspectiva, en este caso, el tamaño de dicho volumen de visualización no cambia desde un extremo al otro y, por lo tanto, la distancia a la que se encuentra la cámara no afecta a las dimensiones en la pantalla que presenta un objeto.

En la proyección paralela, como se puede observar en la Figura, se proyectan puntos de la superficie de los objetos a lo largo de líneas paralelas sobre el plano de despliegue.

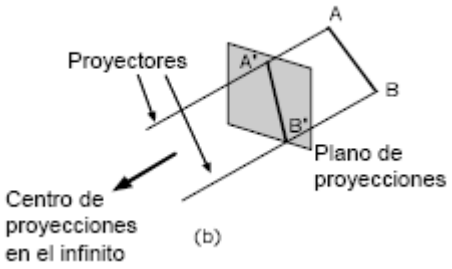

Figura 25. Proyección paralela.

Las posiciones de coordenadas en el plano de visión se transforman a lo largo de líneas paralelas, es decir, las líneas que son paralelas en la escena en coordenadas normales, se proyectan en las líneas paralelas del plano bidimensional.

Se pueden generar tantas proyecciones como puntos de vista del objeto se tengan, como queda expuesto en la siguiente figura.

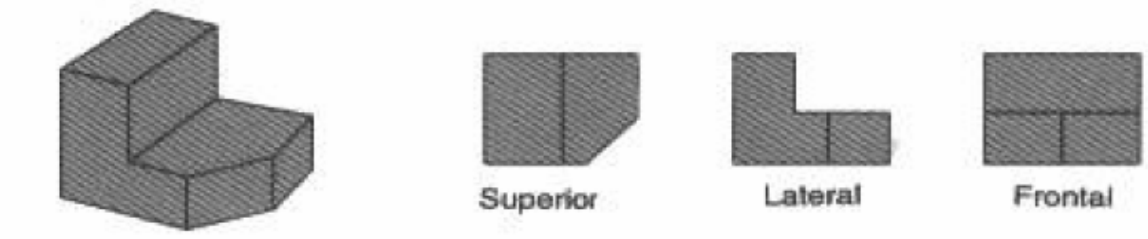

Figura 26. Diferentes tipos de proyecciones.

Este tipo de proyecciones presentan las siguientes características:

- Se pueden especificar a partir de un vector de proyección que define la dirección de las líneas de proyección.

- No proporciona una representación realista del aspecto del objeto tridimensional, sino que da lugar a las vistas exactas de los lados del mismo.

- La proyección es cuadrada en todas sus caras, da lugar a una proyección paralela.

- Esta proyección se utiliza en aplicaciones en las que no es crucial mantener el tamaño de los objetos y los ángulos entre ellos tal y como están proyectados, sino que se pretende representar las dimensiones exactas de tales objetos.

- Este método se utiliza para la generación de trazos a escala de objetos tridimensionales, ya que se mantienen las proporciones relativas de los mismos.

- Por la misma razón, este tipo de proyección se suele utilizar en diseños arquitectónicos y de ingeniería para la representación de objetos a partir de vistas que conservan las proporciones mencionadas. De esta forma, dadas las vistas principales, se puede reconstruir la apariencia de los objetos.

Existen dos tipos de proyecciones paralelas en función de su dirección respecto al plano de visión:

- Proyección paralela ortográfica, en caso de que la proyección sea perpendicular al plano de visión.
- Proyección paralela oblicua, en el caso de proyectar puntos a lo largo de líneas paralelas que no son perpendiculares al plano de visión.

Ambos tipos de proyecciones pueden observarse en la figura:

Figura 27. Proyección ortogonal y oblicua.

6.3.1.1 Proyección ortográfica

Las proyecciones ortogonales son utilizadas para generar diferentes vistas de los objetos, mas concretamente sirven para obtener una vista superior, frontal y lateral, como se observa en la siguiente figura.

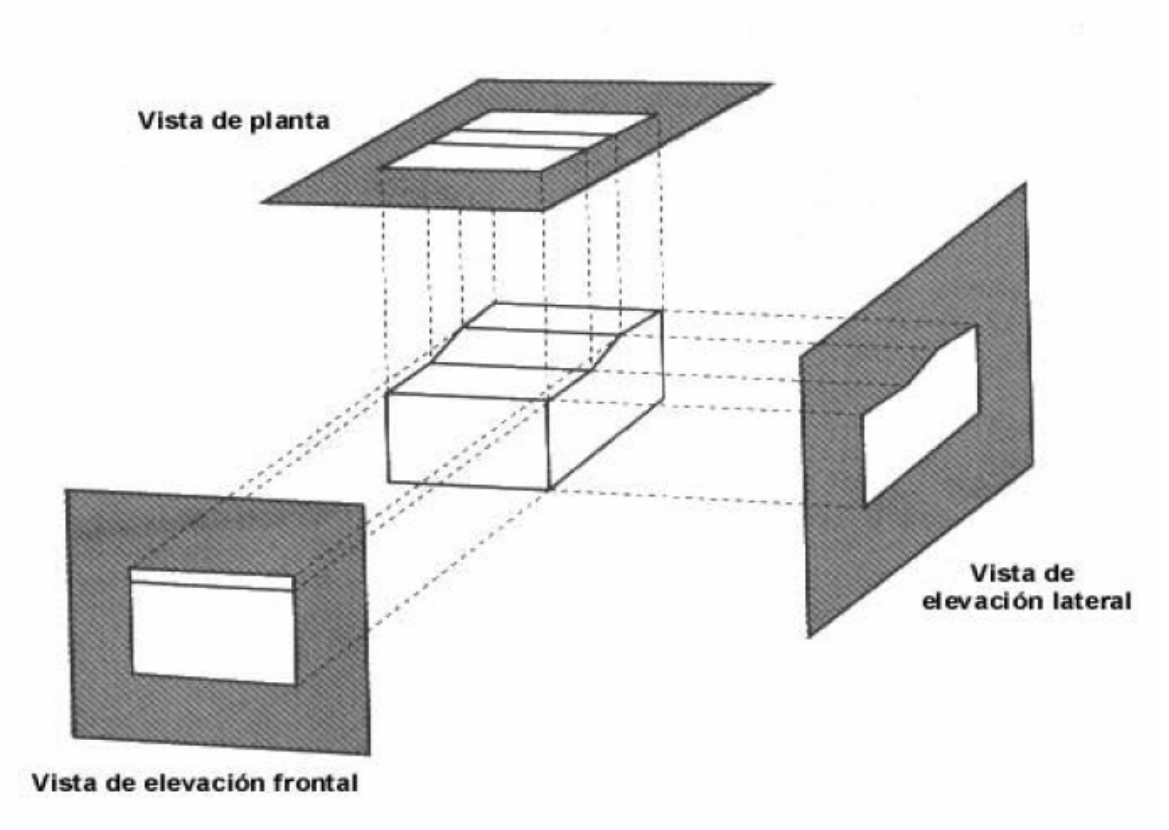

Figura 28. Proyección ortográfica.

Una proyección ortogonal superior se denomina vista de planta, mientras que las proyecciones ortogonales frontal, lateral y posterior reciben el nombre de elevaciones. Los ángulos y segmentos se dibujan con exactitud, y ésta es la razón por la que se suelen utilizar en dibujos de ingeniería y arquitectura.

También es posible crear proyecciones ortogonales que desplieguen varias caras de un objeto. Dichas vistas se conocen como proyecciones ortogonales axonométricas. De todas ellas, la más usual es la proyección isométrica, generada al alinear el plano de proyección de manera que intersecte cada eje de coordenadas en que se define el objeto a igual distancia del origen.

Como ejemplo, puede observarse en la siguiente figura, la proyección isométrica de un cubo, generada al alinear el vector de proyección con la diagonal del mismo.

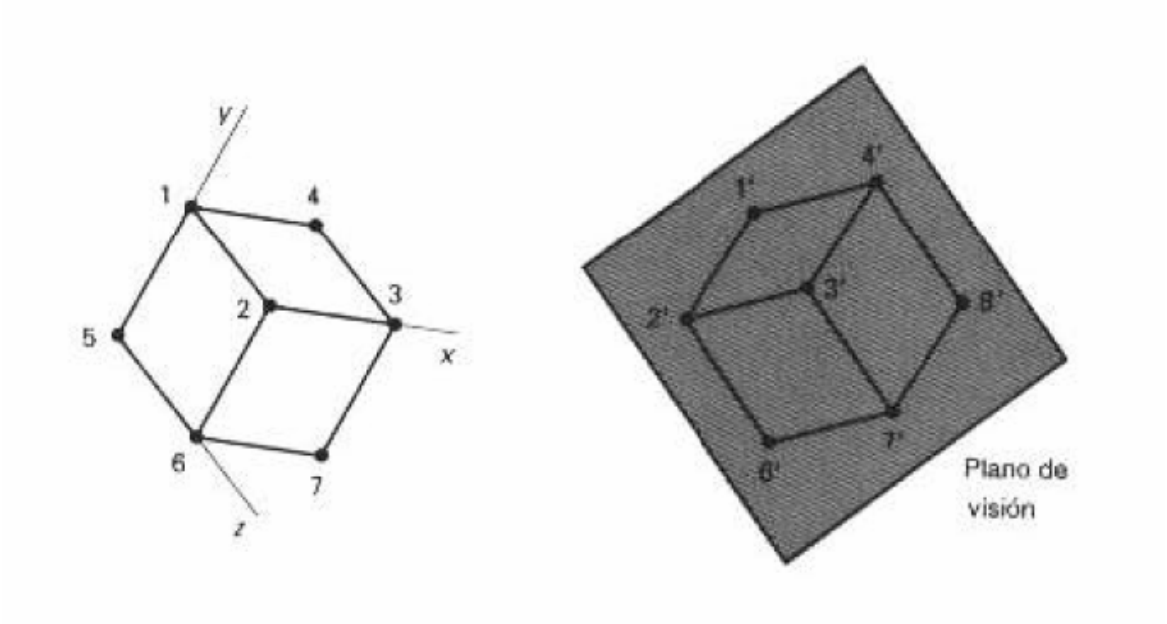

Figura 29. Proyección isométrica de un cubo.

Por lo general, para las proyecciones axonométricas generales, los factores de escala pueden cambiar para las tres direcciones. Supongamos que se sitúa el plano de visión en el punto z_p del eje z_v. Para obtener las coordenadas de proyección (x_p, y_p, z_p) de un punto cuyas coordenadas de vista son (x, y, z), se utilizan las siguientes ecuaciones:

$$x_p = x$$
$$y_p = y$$

Se puede observar un ejemplo de proyección ortogonal en la siguiente figura:

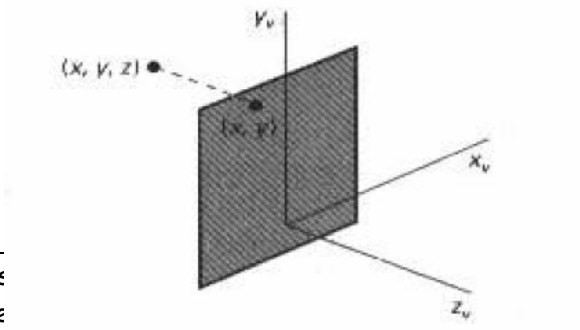

Figura 30. Proyección ortogonal.

6.3.1.2 Proyecciones oblicuas

En una proyección oblicua, el vector de proyección se suele especificar a partir de dos ángulos, φ y α. Se proyecta el punto (x, y, z) a la posición (x_p, y_p) en el plano de visión. Las coordenadas de proyección ortogonal en el plano son (x, y). Se genera un ángulo φ entre la línea de proyección oblicua y la línea contenida en el plano de proyección. Esta última línea forma un ángulo α con la horizontal del plano de proyección, y su longitud es L. Con estos parámetros se tienen las siguientes ecuaciones:

$$x_p = x + L\cos\phi$$
$$y_p = y + Lsen\phi$$

En función del valor que tome α y la coordenada z del punto que se va a proyectar, el valor de L se puede calcular como:

$$L = \frac{z}{\tan\alpha} + z\,L_1$$

donde L_1 es el inverso de tan α.

De esta forma, las ecuaciones anteriores se pueden poner como:

$$x_p = x + z(L_1\cos\phi)$$
$$y_p = y + z(L_1\cos\phi)$$

En conclusión, para generar una proyección paralela en el plano x_v y_v se utiliza la siguiente matriz:

$$\begin{pmatrix} 1 & 0 & L_1 \cos\phi & 0 \\ 0 & 1 & L_1 sen\phi & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Las proyecciones oblicuas se obtienen para valores de L_1 distintos de cero, mientras que las proyecciones ortogonales se generan cuando $L_1 = 0$.

A continuación, veamos el siguiente ejemplo:

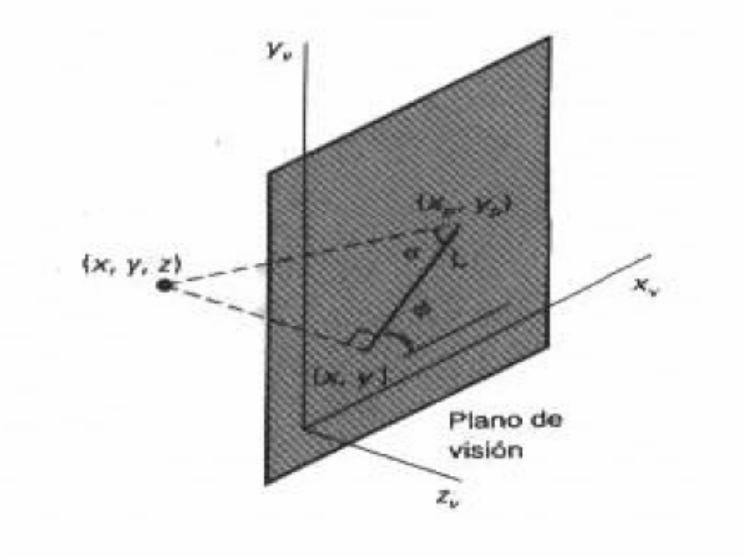

Figura 31. Proyección oblicua.

El ángulo φ suele ser de 30º o de 45º, desplegando una vista de combinación de las partes frontal, lateral y superior (o inferior) de un objeto.

El ángulo α suele tomar aquellos valores para los cuales su tangente vale 1 ó 2.

- En el primer caso, α = 45º, obteniéndose las vistas denominadas proyecciones caballeras, en las cuales todas las líneas perpendiculares al plano de proyección se proyectan sin alterar su longitud. Este efecto puede apreciarse en la siguiente figura:

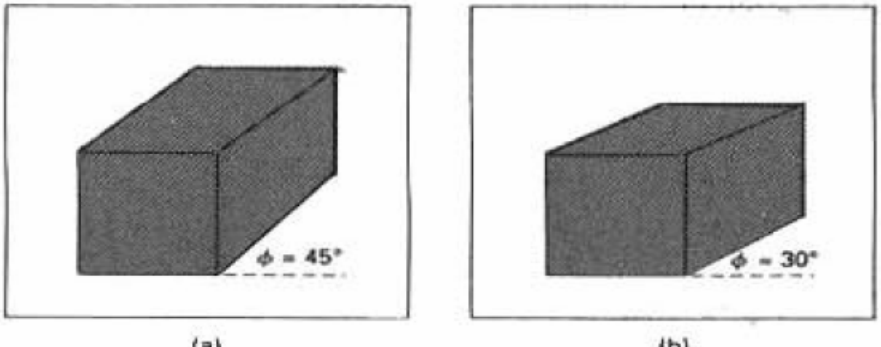

Figura 32. Proyecciones caballeras.

- En el segundo caso, α ≈ 63.4º, obteniéndose las vistas denominadas proyecciones de gabinete, en las cuales las líneas perpendiculares a la superficie de vista se proyectan alterando su longitud a la mitad. Debido a esta reducción, este tipo de proyecciones ofrecen un aspecto más realista que las proyecciones caballera. Puede observarse este efecto a continuación:

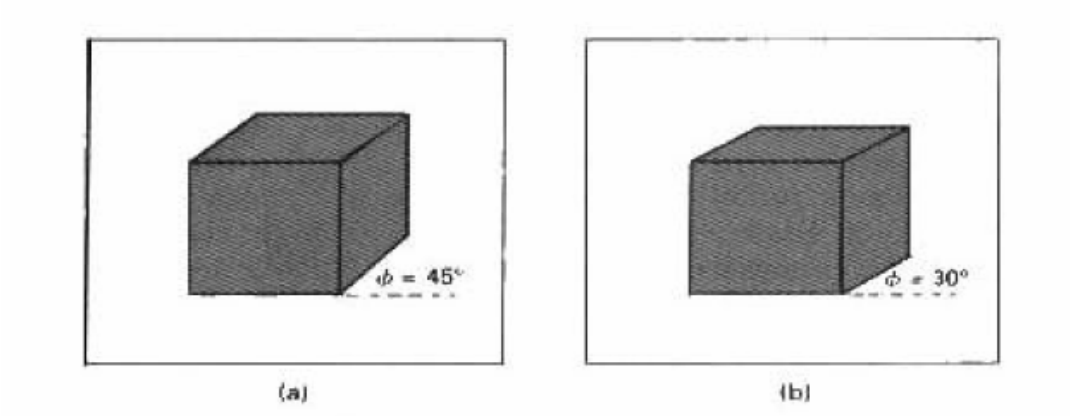

Figura 33. Proyecciones de gabinete.

6.3.1.3 Primitivas para generar proyecciones ortográficas

Una proyección ortográfica en OpenGL se especifica con la primitiva glOrtho(), cuya sintaxis es la siguiente:

```
void glOrtho (Gldouble izquierda, Gldouble derecha,
Gldouble abajo, Gldouble arriba, Gldouble cerca,
Gldouble lejos)
```

Esta primitiva crea una matriz para una ventana de visualización paralela ortográfica y la multiplica por la matriz actual. Sus seis argumentos definen la ventana de visualización y los planos de corte, tanto cercano como lejano:

- Con las coordenadas de dos esquinas de la ventana, quedaría definida por completo la ventana de visualización.

- Los valores de cerca y lejos representan el plano cercano y el plano lejano. En caso de que el objeto a visualizar no se encuentre dentro de ambos planos, dicho objeto se recortará automáticamente.

El objeto se visualizará entre los dos planos de recorte. En caso de que sobrepase dichos planos, el objeto se recortará. En caso de que el objeto sea tan grande que la ventana de visualización esté dentro de él, la pantalla quedará en negro y el objeto no se visualizará. Si el plano de corte es negativo, éste se encontrará detrás de la ventana de visualización. Sin ninguna otra transformación, la dirección de la proyección es paralela al eje Z y el punto de visión baja a lo largo del eje Z.

Como ejemplo de la utilización de proyecciones ortográficas, a continuación se presenta parte del código del programa ortho.c :

```
// Previene la división entre cero
if(h == 0)
      h = 1;
// Ajusta la vista a las dimensiones de la ventana
glViewport(0, 0, w, h);
// Reinicia el sistema de coordenadas.
glMatrixMode(GL_PROJECTION);
// Reinicia el sistema de coordenadas a coordenadas oculares abriendo la matriz
// identidad en la matriz actual.
glLoadIdentity();
// Establece el volumen de recorte (izquierda, derecha, abajo, arriba, cerca,
// lejos)
if (w <= h)
      glOrtho   (-nRange,   nRange,   -nRange*h/w,   nRange*h/w,   -nRange*2.0f,
      nRange*2.0f);
else
      glOrtho   (-nRange*w/h,   nRange*w/h,   -nRange,   nRange,   -nRange*2.0f,
      nRange*2.0f);

// Reinicia la matriz del modelador.
glMatrixMode(GL_MODELVIEW);
// Reinicia el sistema de coordenadas a coordenadas oculares
glLoadIdentity();
```

En las siguientes figuras se puede observar la salida proporcionada por el programa anterior:

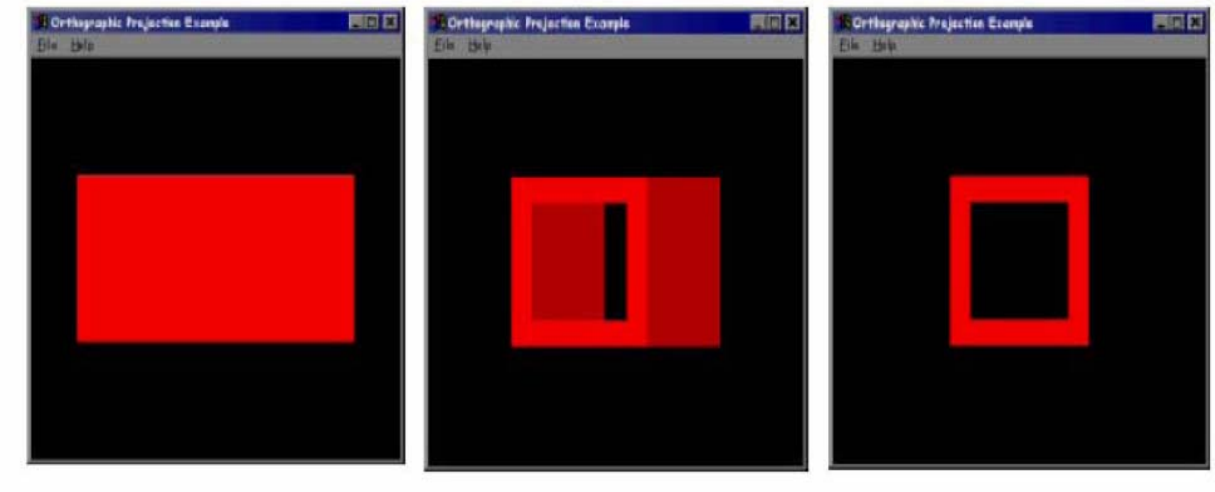

Figura 34. Salida por pantalla de ortho.c.

En el caso especial de proyectar una imagen bidimensional a una pantalla también bidimensional, se utiliza la rutina `gluOrtho2D()`. Esta rutina es similar a la versión tridimensional, `glOrtho()`, excepto en que se supone que todas las coordenadas z de los objetos están entre –1.0 y 1.0. Al dibujar objetos bidimensionales utilizando comandos bidimensionales, todas las coordenadas z son cero.

Al final del apartado se verá un ejemplo de aplicación de las primitivas para generar perspectivas paralelas y perspectivas, como veremos a continuación.

6.3.2 Proyección perspectiva

La proyección perspectiva, los objetos en el plano de visión se transforman a lo largo de líneas que convergen en un punto denominado centro de referencia de proyección o simplemente punto de proyección. En la siguiente figura se puede observar este tipo de proyecciones.

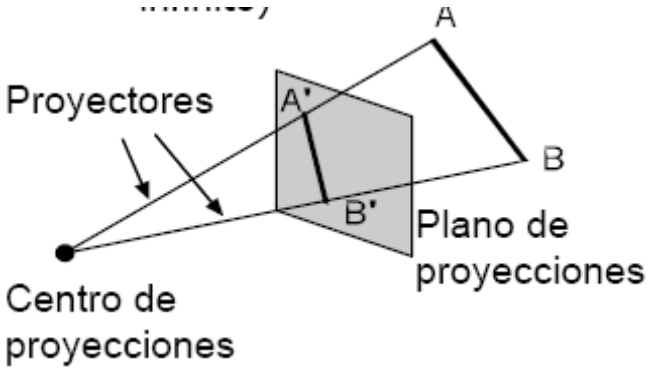

Figura 35.Proyección perspectiva.

Se proyectan los puntos del objeto hacia el plano de despliegue a lo largo de trayectorias convergentes. Calculando la intersección de las líneas de proyección con el plano de visión, queda definida la vista que se proyecta de un objeto.

Al proyectar un objeto tridimensional en un plano de visión utilizando las ecuaciones de transformación de perspectiva, cualquier conjunto de líneas paralelas en el objeto que no sean paralelas al plano se proyectan en líneas convergentes. Así las líneas paralelas parece que convergen hacia un punto lejano en el fondo. Las líneas paralelas que también lo son al plano de visión se proyectan como líneas paralelas.

El punto en el cual converge un conjunto de líneas paralelas que se proyecta recibe el nombre de *punto de fuga*. Cada conjunto de líneas paralelas que se proyecta tiene un punto de fuga separado. Dependiendo del número de conjuntos de líneas paralelas que hay en una escena, ésta tendrá un determinado número de puntos de fuga.

Se denomina punto de fuga principal al punto de fuga para cualquier conjunto de líneas que son paralelas a uno de los ejes principales de un objeto. A partir de la orientación del plano de proyección se controla el número de puntos de fuga principales (uno, dos o tres).

Las proyecciones en perspectiva se clasifican como proyecciones de uno, dos o tres puntos. El número de ejes principales que intersectan el plano de visión

determina el número de puntos de fuga principales en una proyección. El siguiente ejemplo del cubo presenta gráficamente esto.

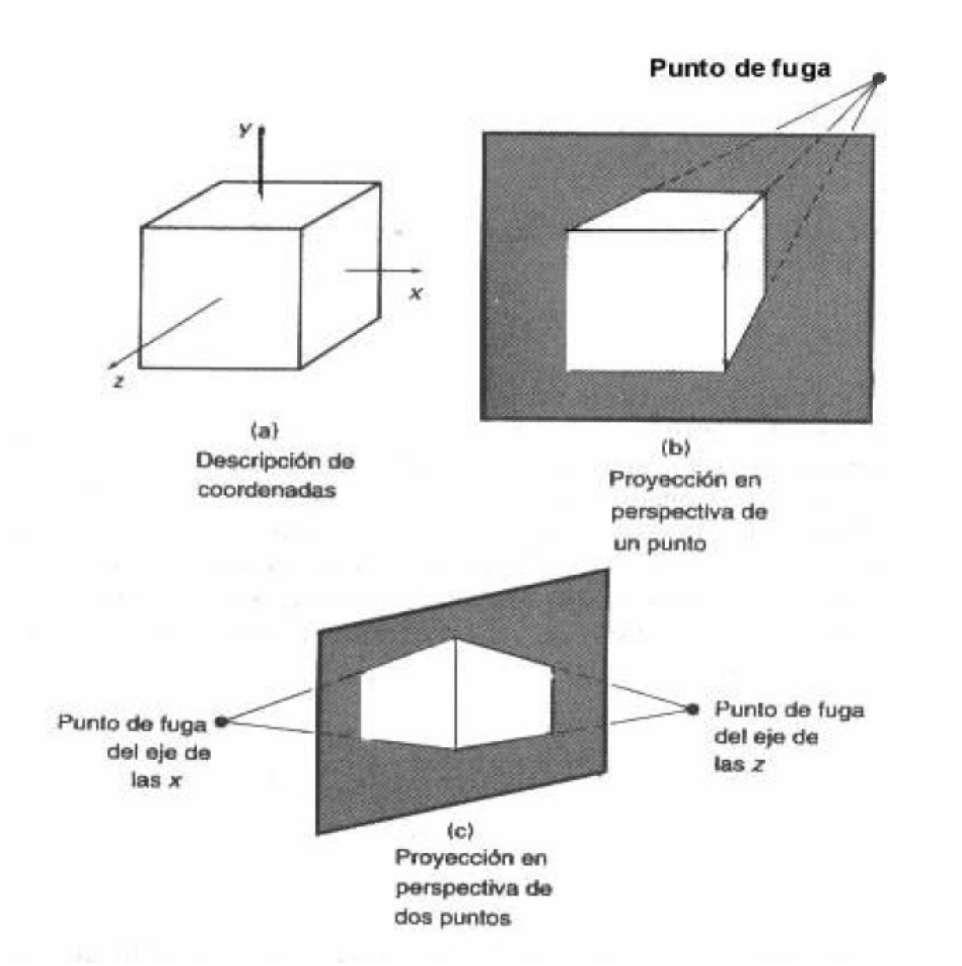

Figura 36. Puntos de fuga en la proyección perspectiva.

Este tipo de perspectivas no conserva las proporciones relativas pero la proyección perspectiva da lugar a vistas realistas. Un mismo objeto aparece más pequeño en la parte posterior del volumen de visualización que si estuviera dibujado en el frente del mismo, ya que el frontal y el ancho posterior del volumen de visualización no tienen las mismas dimensiones.

Es decir, aquellos objetos que están más próximos a la posición de vista, se despliegan más grandes que los objetos de igual tamaño que están más alejados de la posición de vista. Como puede apreciarse en la siguiente figura, las

proyecciones de objetos del mismo tamaño, los cuales están más próximos al plano de proyección, son más grandes que las proyecciones de objetos alejados.

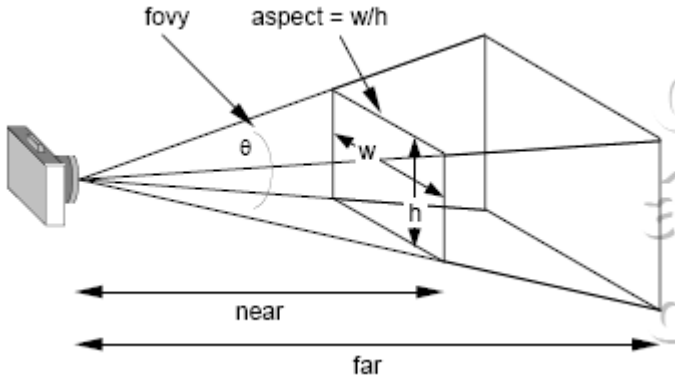

Figura 37. Efecto realista de la proyección perspectiva.

Esto es debido a que el volumen de visualización para la proyección perspectiva es un *frustum* de una pirámide, es decir, una pirámide truncada cuya parte superior ha sido cortada por un plano paralelo a su base. Los objetos que caen dentro del volumen de visualización se proyectan hacia el vértice de la pirámide, lugar donde se encuentra la cámara o punto de visión. Los objetos más próximos a la cámara se ven más grandes porque ocupan una parte del volumen de visualización proporcionalmente más grande que los más alejados, situados en la parte más grande del frustum.
Este tipo de proyección se suele utilizar en animaciones y aplicaciones que requieren algún grado de realismo, puesto que es similar a la forma en que trabajan el ojo humano y la lente de una cámara al formar las imágenes.

6.3.2.1 Primitivas para generar proyecciones en perspectiva

Para especificar una proyección perspectiva se utiliza el comando `glFrustum()`, cuya sintaxis es la siguiente:

```
void glFrustum (Gldouble izquierda, Gldouble derecha,
Gldouble abajo, Gldouble arriba, Gldouble cerca,
Gldouble lejos)
```

Dicha rutina calcula una matriz que representa la proyección perspectiva y multiplica por ella la matriz de proyección actual (generalmente la matriz identidad). Como argumentos, se especifican las coordenadas y distancias entre los planos de trabajo frontal y posterior, es decir, los planos de corte para limitar el volumen de vista, que define un tronco piramidal. De esta forma, el volumen de visualización del frustum se define a partir de los parámetros: (izquierda, abajo, -cerca) y (derecha, arriba, -cerca) especifican las coordenadas (x, y, z) de las esquinas inferior izquierda y superior derecha del plano más cercano; cerca y lejos representan la distancia desde el punto de visión a los planos más cercano y más alejado. Deberían tomar siempre un valor positivo. El volumen de visualización se utiliza para recortar los objetos que se encuentran fuera de él.

Como se puede observar en la siguiente figura, los cuatro lados del frustum, su techo y su base corresponden a los seis planos del volumen de visualización.

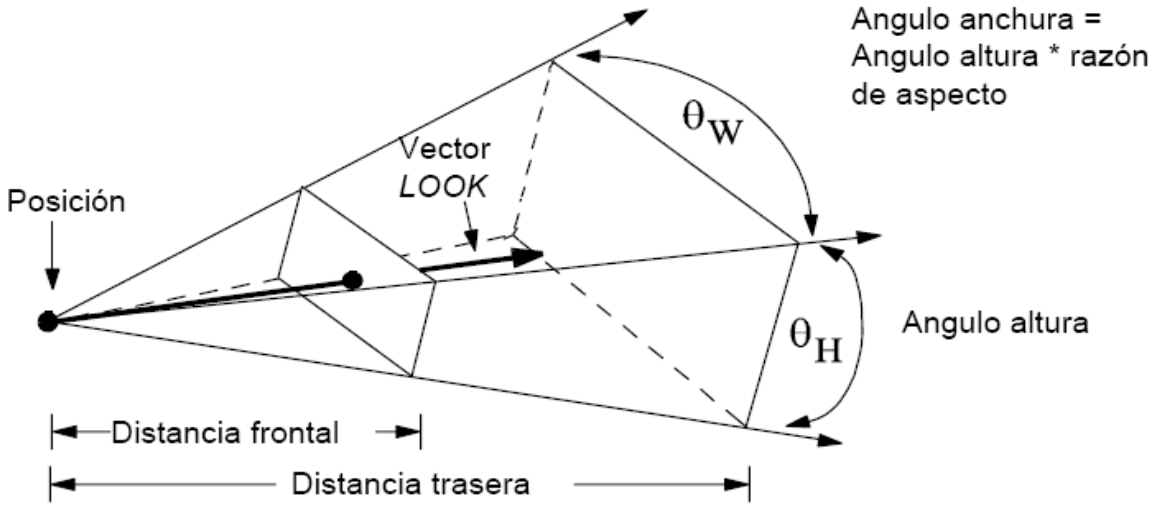

Figura 38. Volumen de perspectiva útil con glfrustum

Las partes de los objetos que caen fuera de estos planos son recortados de la imagen final. `glFrustum()` no necesita definir un volumen de visualización simétrico.

El frustum tiene una orientación por defecto en el espacio tridimensional. Para cambiar esta orientación, se pueden definir rotaciones o traslaciones sobre la

matriz de proyección. Además, el frustum no tiene que ser simétrico, y sus ejes no tienen por qué estar alineados necesariamente con el eje Z.

Aunque conceptualmente es fácil de entender, `glFrustum()` es difícil de utilizar. En su lugar, se puede recurrir a la rutina `gluPerspective()`, cuya sintaxis es la siguiente:

```
void gluPerspective (GLdouble angulo, GLdouble aspecto,
GLdouble zFrontal, GLdouble zPosterior)
```

Esta rutina se diferencia de `glFrustum()` en la manera de definir la ventana de visualización: en lugar de especificar los vértices necesarios de la ventana, en este caso sólamente se define el ángulo de apertura de la cámara virtual (ángulo), que puede tomar valores comprendidos entre 0º y 180º, y la relación entre el largo y ancho del plano cercano de corte (aspecto). Estos dos parámetros determinan una pirámide no truncada a lo largo de la línea de visión. Para truncar la pirámide, se especifica la distancia entre el punto de visión y los planos de recorte más cercano y más alejado.

`gluPerspective()` está limitado a crear frustums que sean simétricos con respecto a los ejes X e Y a lo largo de la línea de visión.

Al igual que con `glFrustum()`, se pueden aplicar rotaciones y traslaciones para cambiar la orientación por defecto del volumen de visualización creado por `gluPerspective()`. Sin tales transformaciones, el punto de visión permanece en el origen y la línea de visión baja a lo largo del eje Z.

Con `gluPerspective()`, es necesario dar valores apropiados para el campo de visión, o la imagen puede parecer distorsionada.

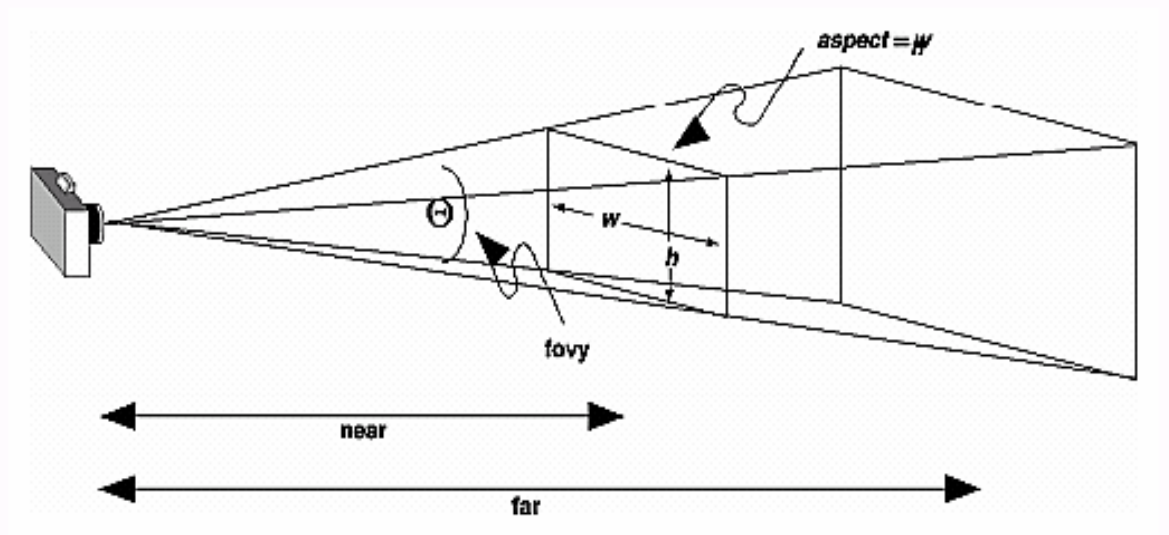

Figura 39. Volumen de perspectiva útil con gluPerspective

Como ejemplo de utilización de proyecciones perspectivas, a continuación se presenta parte del código del programa perspect.c.

```
// Previene la división entre cero
if(h == 0)
      h = 1;
// Ajusta la vista a las dimensiones de la ventana.
glViewport(0, 0, w, h);
// Calcula la relación de aspecto de la ventana.
fAspect = (GLfloat)w/(GLfloat)h;
// Reinicia el sistema de coordenadas.
glMatrixMode(GL_PROJECTION);
// Reinicia el sistema de coordenadas a coordenadas oculares abriendo la matriz
identidad en la matriz actual.
glLoadIdentity();

/**********************/
/* Proyección perspectiva */
/**********************/
/**
Define una matriz de proyección en perspectiva para la visualización.
Parámetros:
- El campo de visión en la dirección y es de 60 grados.
- La relación de aspecto es fAspect, calculada previamente.
- La distancia desde el observador al plano de trabajo próximo es 1.
- La distancia desde el observador al plano de trabajo lejano es 400.
*/
gluPerspective(60.0f, fAspect, 1.0, 400.0);

// Reinicia la matriz del modelador.
glMatrixMode(GL_MODELVIEW);
// Reinicia el sistema de coordenadas a coordenadas oculares
glLoadIdentity();
```

En las siguientes figuras se puede observar la salida proporcionada por el programa anterior:

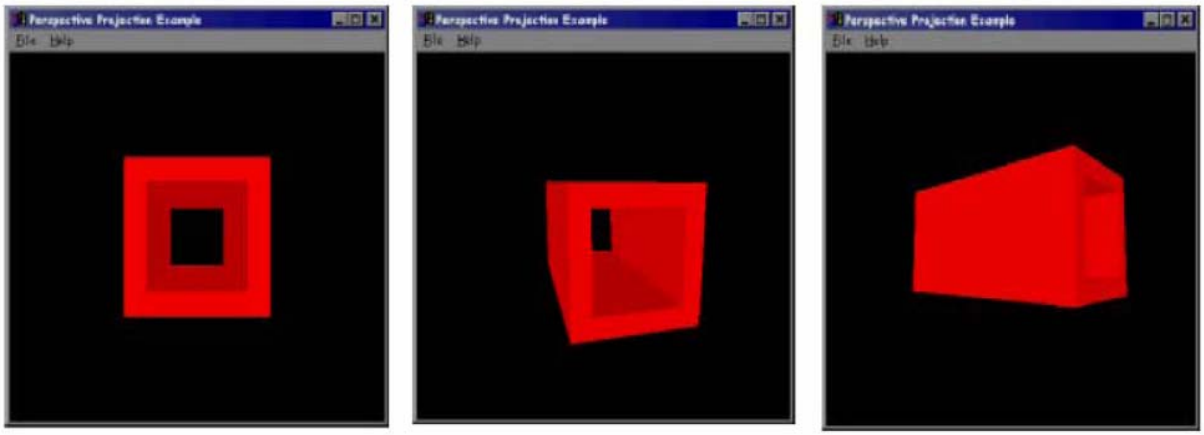

Figura 40. Salida por pantalla del programa perspect.c.

Como colofón, se presenta un ejemplo donde se pueden diferenciar las dos tipos de proyecciones presentadas anteriormente, el código, prueba.cpp, se encuentra en el anexo de ejemplos. Su salida por pantalla es la siguiente.

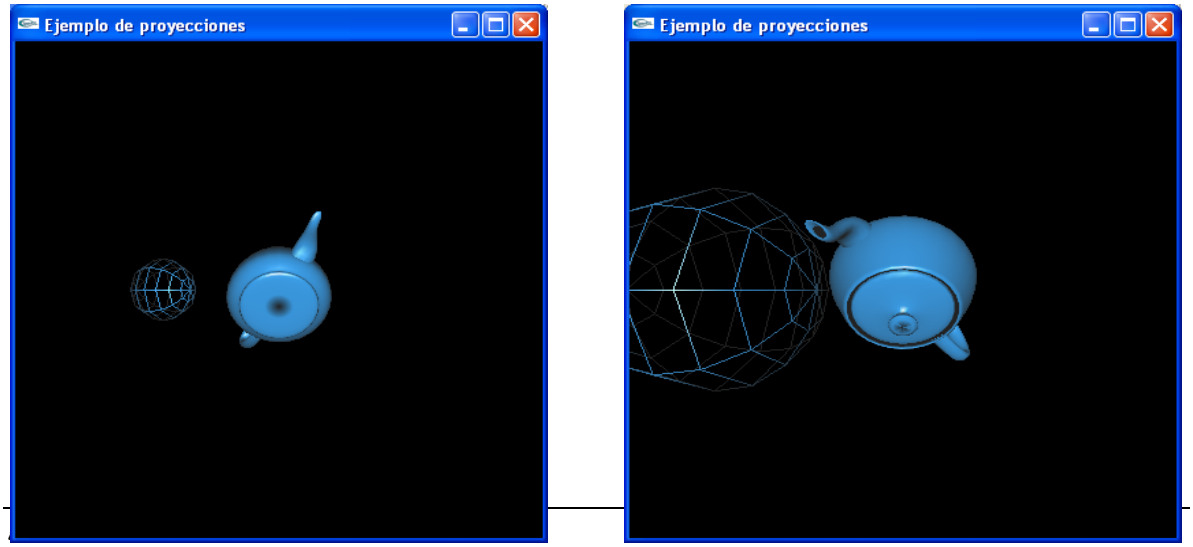

Figura 41. Salida por pantalla del programa prueba.cpp.

En la figura anterior, aparecen respectivamente la salida por pantalla para una proyección ortográfica y una perspectiva en el mismo instante de rotación de la esfera, como se puede observar, las proyecciones ortográfica mantienen las dimensiones de los objetos pero no dan un representación realista de la situación, como se puede observar en la captura de la proyección perspectiva.

6.3.3 Planos de recorte adicionales

Se ha visto anteriormente que seis planos que determinan el volumen de visualización, además de estos, se pueden definir hasta seis planos adicionales de recorte para restringir aún más dicho volumen. Esto es útil para eliminar determinados objetos de una escena, por ejemplo, para representar una vista parcial de un objeto.

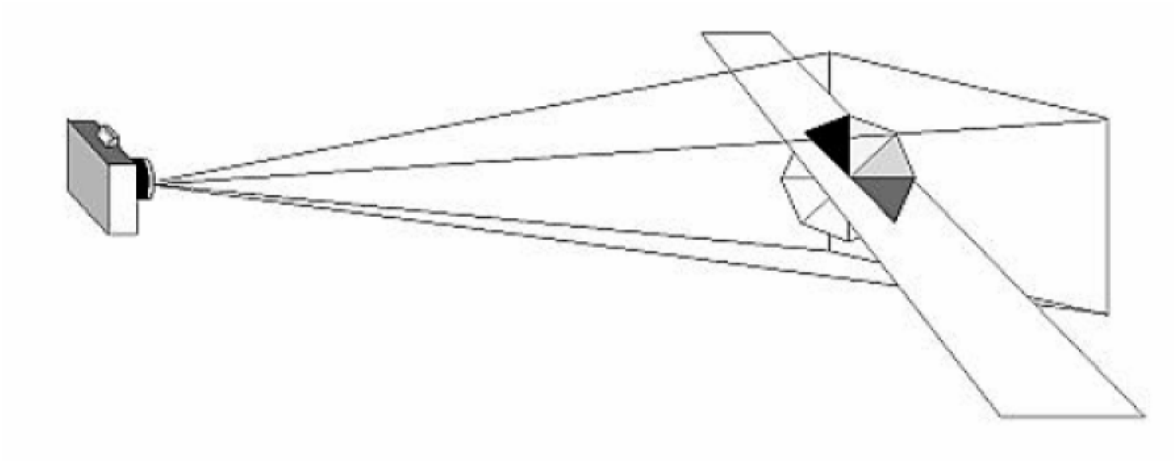

Figura 42. Utilización de planos de recorte.

Cada plano se especifica por los coeficientes de su ecuación (Ax + By + Cz + D = 0). Los planos de recorte cambian automáticamente al aplicar transformaciones de la vista y del modelo. El volumen de vista resultante será la intersección entre el volumen de vista inicial y cada uno de los espacios intermedios definidos por los planos de recorte especificados. OpenGL reconstruye automáticamente los bordes de los objetos que quedan recortados.

Para definir un plano de recorte se utiliza la primitiva `glClipPlane()`, cuya sintaxis es la siguiente:

```
glClipPlane (GLenum numero_plano, const GLdouble *ecuacion)
```

Los parámetros de dicha función son:

- ecuacion: matriz con las coordenadas que definen el plano de recorte Ax + By + Cz + D = 0.
- numero_plano: número entero que especifica qué plano de recorte se está definiendo. Puede tomar el valor `GL_CLIP_PLANEi`, donde i es un número entre 0 y 5 que especifica cada uno de los seis planos de recorte que se pueden definir.

Además, es necesario habilitar cada uno de los planos de recorte que se definen. Para ello, se utiliza la primitiva `glEnable (GL_CLIP_PLANEi).` Si se desea prescindir del efecto que proporciona un plano de recorte, se puede utilizar la función `glDisable (GL_CLIP_PLANEi).`

En algunas implementaciones, se pueden definir más de seis planos de recorte. Utilizando la primitiva `glGetIntegerv (GL_MAX_CLIP_PLANES),` se determina el número de planos de recorte que se soportan.
Como ejemplo de la utilización de planos de recorte, a continuación se presenta parte del código del programa clip.c .

```
void display(void)
{
/* se declaran las ecuaciones de los planos de recorte */
GLdouble eqn[4] = {0.0, 1.0, 0.0, 0.0}; /* plano de recorte y=0 */
GLdouble eqn2[4] = {1.0, 0.0, 0.0, 0.0};/* plano de recorte x=0 */
glClear(GL_COLOR_BUFFER_BIT);
glColor3f (1.0, 1.0, 1.0);
/* se almacena el estado inicial y se traslada el sistema de coordenadas hacia
dentro de la pantalla */
glPushMatrix();
glTranslatef (0.0, 0.0, -5.0);

/* se definen y habilitan los planos de recorte */
glClipPlane (GL_CLIP_PLANE0, eqn);
glEnable (GL_CLIP_PLANE0);

glClipPlane (GL_CLIP_PLANE1, eqn2);
glEnable (GL_CLIP_PLANE1);

// se rota el sistema de coordenadas ya que glut dibuja la esfera en el eje Z
glRotatef (90.0, 1.0, 0.0, 0.0);
glutWireSphere(1.0, 20, 16);
glPopMatrix();
glFlush ();
}
```

El resultado se puede observar en la siguiente figura:

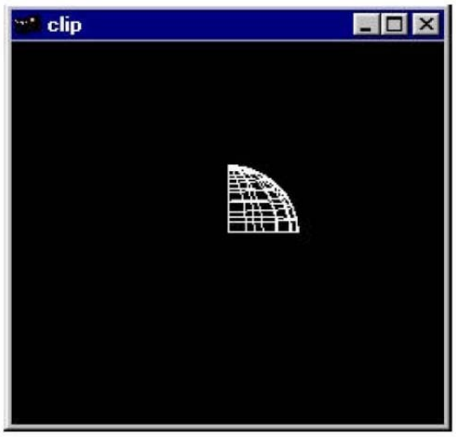

Figura 43. Vista parcial de una esfera utilizando planos de recorte.

6.4 Manual de referencia

glClipPlane

Propósito: Define un plano de recorte adicional.
Fichero de inclusión: `<gl.h>`
Sintaxis:

```
void glClipPlane (Glenum numero_plano, const Gldouble *ecuacion)
```

Descripción: Esta función define un plano que se utilizará para crear una vista parcial de
una escena. Dicho plano permitirá ocultar partes visibles de la escena.
Parámetros:
numero_plano
`GLenum`: número entero que especifica qué plano de recorte se está definiendo. Puede tomar
el valor `GL_CLIP_PLANEi`, donde i es un número entre 0 y 5 que especifica cada uno de
los seis planos de recorte que se pueden definir.
ecuación

`GLdouble*` : matriz con las coordenadas que definen el plano de recorte, concretamente,
contiene los coeficientes de la ecuación Ax + By + Cz + D = 0.

Retornos: Ninguno.

glFrustum

Propósito: Multiplica la matriz activa por una matriz de perspectiva.

Fichero de inclusión: <gl.h>

Sintaxis:

```
void glFrustum( GLdouble izquierda, GLdouble derecha, GLdouble abajo, GLdouble arriba, GLdouble cerca, GLdouble lejos);
```

Descripción: Esta función crea una matriz de perspectiva que produce una proyección en perspectiva.

Parámetros: *izquierda, derecha*

GLdouble: coordenada de los planos de trabajo izquierdo y derecho.

abajo, arriba

GLdouble: coordenada de los planos de trabajo inferior y superior.

cerca, lejos

GLdouble: coordenada de los planos de trabajo próximo y lejano. Los dos valores deben ser
positivos.

Retornos: ninguno.

glLoadIdentity

Propósito: Abre en la matriz actual la de identidad.

Fichero de inclusión: <gl.h>

Sintaxis: *void glLoadIdentity(void);*

Descripción: Reemplaza la matriz de transformación actual con la matriz identidad.

Retornos*:* Ninguno

glLoadMatrix

Propósito: Abre la matriz especificada como la actual.

Fichero de inclusión: <gl.h>

Sintaxis:

*void glLoadMatrix(constoadMatrix(const GLdouble *m); GLfloat *m); void glL*

Descripción: Reemplaza la matriz de transformación actual con una matriz proporcionada.

Parámetros:

*m

GLdouble o GLsformación float: esta matriz representa una matriz de 4x4 que se usará en la matriz de transformación actual.

Retornos: Ninguno.

glMatrixMode

Propósito: Especifica la matriz actual (PROJECTION, MODELVIEW, TEXTURE).

Fichero de inclusión: <gl.h>

Sintaxis: void glMatrixMode(GLenum modo);

Descripción: esta función se usa para determinar que pila de matrices(GL_PROJECTION, GL_MODELVIEW o GL_TEXTURE) se usara con las operaciones de matrices.

Parámetros:

modo

Glenum: identifica que pila de matrices se usara con las operaciones de matrices.

Puede ser: GL_PROJECTION, GL_MODELVIEW o GL_TEXTURE.

Retornos: Ninguno.

glMultMatrix

Propósito: Multiplica la matriz actual por la especificada.

Fichero de inclusión: <gl.h>

Sintaxis:

```
void MultMatrix( const GLdouble *m);
void MultMatrix(const Glfloat *m);
```

Descripción: Esta función multiplica la pila de matrices seleccionada actualmente con la especificada, La matriz resultante es almacenada como la matriz actual en la parte superior de la pila de matrices.

Parámetros:

*m

GLdouble o GLfloat: esta matriz representa una matriz de 4x4 que se usará en la matriz de transformación actual. La matriz se almacena en orden de mayor columna como una sucesión de 16 valores.

Retornos: Ninguno.

glOrtoh

Propósito: Define una proyección ortográfica
Fichero de inclusión: `<gl.h>`
Sintaxis:

```
void glOrtho (Gldouble izquierda, Gldouble arriba,
Gldouble derecha, Gldouble abajo, Gldouble cerca,
Gldouble lejos);
```

Descripción: Esta función define una matriz de proyección ortográfica. Concretamente, crea una matriz para una ventana de visualización paralela ortográfica y la multiplica por la matriz actual.
Parámetros:
izda, dcha, arriba, abajo
`GLdouble` : Coordenadas de las dos esquinas de la ventana de visualización.
cerca, lejos
`GLdouble` : Especifican los planos de trabajo cercano y lejano.
Retornos: Ninguno.

glPop Matrix

Propósito: Saca la matriz actual de la pila de matrices.
Fichero de inclusión: `<gl.h>`
Sintaxis: *`void glPopMatrix(void);`*
Descripción: Esta función se usa para sacar la última matriz de la pila de matrices.
Retornos: Ninguno.

glPushMatrix

Propósito: Coloca la matriz actual en la pila de matrices.
Fichero de inclusión: `<gl.h>`
Sintaxis: *`void glPushMatrix(void);`*
Descripción: Esta función se emplea para colocar la matriz actual en la pila de matrices actual. Esto se emplea a menudo para almacenar la matriz de transformación actual de manera que pueda recuperarse más tarde con una llamada a `glPopMatrix`
Retornos: Ninguno.

glRotate

Propósito: Rota la matriz actual con una matriz de rotación.

Fichero de inclusión: <gl.h>

Sintaxis:

void glRotated(GLdouble angulo, GLdouble x, GLdouble y, GLdouble z);
void glRotatef(GLfloat angulo, GLfloat x, Glfloat y, GLfloat z);

Descripción: Esta función multiplica la matriz actual por una matriz de rotación que ejecuta una rotación en sentido horario negativo, sobre un vector direccional que pasa desde el origen a través del punto (x,y,z). La nueva matriz rotada se convierte en la matriz de transformación.

Parámetros:

ángulo
El ángulo de rotación en grados.
x, y, z
Un vector de dirección desde el origen empleado como eje de rotación.

Retornos: Ninguno.

GlScale

Propósito: Multiplica la matriz actual por una matriz de escala.

Fichero de inclusión: <gl.h>

Sintaxis:

```
void glScaled(GLdouble x, GLdouble y, GLdouble z);
void glScalef(GLfloat x, Glfloat y, GLfloat z);
```

Descripción: Esta función multiplica la matriz actual por una matriz de escala. La nueva matriz escalada se convierte en la matriz de transformación actual.

Parámetros:

x, y, z
factores de escala a lo largo de los ejes x,y,z.

Retornos: Ninguno.

glTranslate

Propósito: Multiplica la matriz actual por una matriz de traslación.

Fichero de inclusión: <gl.h>

Sintaxis:

```
void glTranslated(GLdouble x, GLdouble y, GLdouble z);
void glTranslatef(GLfloat x, GLfloat y, GLfloat z);
```

Descripción: Esta función multiplica la matriz actual por una matriz de traslación. La nueva matriz trasladada se convierte en la matriz de transformación actual.

Parámetros:

x, y, z
Las coordenadas x,y,z del vector de traslación.

Retornos: Ninguno.

gluLookAt

Propósito: Define una transformación de visualización.
Fichero de inclusión: <glu.h>
Sintaxis:

```
void gluLookAt(GLdouble ojox, GLdouble ojoy,
GLdouble ojoz, GLdouble centrox, GLdouble centroy,
GLdouble centroz, GLdouble arribax, GLdouble arribay,
GLdouble arribaz);
```

Descripción: Esta función define una transformación de visualización basada en la posición del observador, la posición del centro de la escena y un vector que apunta hacia arriba desde la perspectiva del observador.
Parámetros:

ojox, ojoy, ojoz
`GLdouble` : Coordenadas x,y,z del punto de vista.
centrox, centroy, centroz
`GLdouble` : Coordenadas x,y,z de*l centro de la escena a la que* se mira.
arribax, arribay, arribaz
`GLdouble` : Coordenadas x,y,z del vector de dirección ascendente.

Retornos: Ninguno.

gluOtho2D

Propósito: Define una proyección ortográfica bidimensional.
Fichero de inclusión: `<glu.h>`
Sintaxis:

```
gluOrtho2D(GLdouble izda, GLdouble dcha, GLdouble
abajo, void GLdouble arriba);
```

Descripción: Esta función define una matriz de proyección ortográfica 2D. Esta matriz de proyección es equivalente a llamar a glOrtho con cero y uno para lo s valores *cerca* y *lejos* respectivamente
Parámetros:

izda, dcha
`GLdouble` : Especifican los planos de trabajo izquierdo y derecho.
arriba, abajo
`GLdouble` : Especifican los planos de trabajo superior e inferior.

Retornos: Ninguno

gluPerspective

Propósito: Define una matriz de proyección en perspectiva para la visualización.

Fichero de inclusión: `<glu.h>`

Sintaxis:

void gluPerspective(GLdouble angulo, GLdouble aspecto, GLdouble zcerca, GLdouble zlejos);

Descripción: Esta función crea una matriz que describe un frustum de visualización en coordenadas globales. La relación de aspecto debe ajustarse a la relación de aspecto de la vista (especificada con glViewPort). La división de perspectiva está basada en el ángulo del la campo de visión y la distancia a los planos de trabajo próximo y lejano.

Parámetros:

angulo

`GLdouble` : Especifica el campo de visión en grados en la dirección Y.

aspecto

`GLdouble` : Especifica la relacion de aspecto. Se emplea para determinar el campo de visión en la dirección X. La relación de aspecto es X / Y.

zcerca, zlejos

`Gldouble`: Especifica la distancia desde el observador a los planos de trabajo próximo y lejano.

Retornos: Ninguno:

6.5 Resumen

El presente documento ha sido una exposición detallada de los diferentes tipos de transformaciones que un objeto puede experimentar sobre un sistema de coordenadas, mas concretamente se han visto transformaciones sobre el sistema de coordenadas y proyecciones de objetos sobre el espacio. Todas estas transformaciones se han estudiado bajo un punto de vista matemático e informático, ya que se ha centrado el estudio para poder ser implementado mediante la librería OpenGL.

Este estudio servirá para que el lector tenga una referencia de cómo construir aplicaciones con referencias espaciales.

Se han estudiado las diferentes transformaciones que pueden sufrir los objetos como rotaciones, escala y traslaciones, también se han visto brevemente diferentes transformaciones de deformación.

Con respecto a las proyecciones, se han estudiado los diferentes tipos, como pueden ser proyecciones paralelas y perspectivas y sus importantes aplicaciones

no solamente en el mundo de la informática gráfica, sino en ingeniería y arquitectura.

Para apostillar el documento se ha incluido un glosario de las diferentes funciones que se han comentado en el presente documento y una serie de ejemplos para mostrar gráficamente los diferentes conceptos estudiados.

Bibliografía

Redbook –Ejemplos del libro "OpenGL Programming Guide".
http://trant.sgi.com/opengl/examples/redbook/redbook.html
OpenGL Simple Samples.
http://trant.sgi.com/opengl/examples/samples/samples.html
More sophisticated OpenGL Samples
http://trant.sgi.com/opengl/examples/more_samples/more_samples.html
OpenGL Tutorials (excelentes tutoriales para diversas plataformas)
http://nehe.gamedev.net
GLUT (Graphics Library Utility Toolkit Samples
http://trant.sgi.com/opengl/toolkits/glut-3.5/progs/progs.html
Nate Miller's OpenGL Tutorials & Code (great code examples)
http://nate.scuzzy.net/gltut/
Nate Robins' OpenGL Chronicles
http://www.xmission.com/~nate/opengl.html
Page Michael Gold's OpenGL
http://www.berkelium.com/OpenGL/
TutorialWin 32
http://trant.sgi.com/opengl/examples/win32_tutorial/win32_tutorial.html
OpenGL Challenge Handy Coding Routines
http://oglchallenge.otri.net/challengepack/
OpenGL Components for Delphi & C++ Builder
http://www.signsoft.com
Fun with OpenGL (utilizand o lenguaje C para la API Win32)
http://home.att.net/~bighesh
Beginner OpenG L Code Samples
http://www.uic.edu/~jayvee/opengl/
matumot's OpenGL Paradise
http://www.nk-ex a.co.jp/~matumot/index-e.shtml
Fatal FX OpenGL (GLUT) sources
http://www.fatalfx.com/opengl/
Sample Programs for Mark Kilgard's Articles in the X Journal Magazine
ftp://sgigate.sgi.com/pub/opengl/xjournal/
Brian Paul's Mesa 3D, OpenG L-like API
http://www.mesa3d.org
Getting Started with OpenGL
http://members.home.com/borealis/gldemo.c
druid's GL Journal
http://www.gameprog.com/opengl
MFC Programmers' Sourcebook OpenGL code samples |
http://www.codeguru.com/opengl/index.htm
Paul´s OpenGL page
http://home.clara.net/paulyg/ogl.htm

Buen Documento de Transformaciones de la Universidad de Oviedo
http://di002.edv.uniovi.es/~rr/PIG06-07.pdf
Documento sobre proyecciones
www3.uji.es/~ribelles/Docencia/E26-F27/Docs/4-Proyecciones.pdf