UG

Lenguaje de Programación: C++
GLUT
Transformaciones

José Luis Alonzo Velázquez

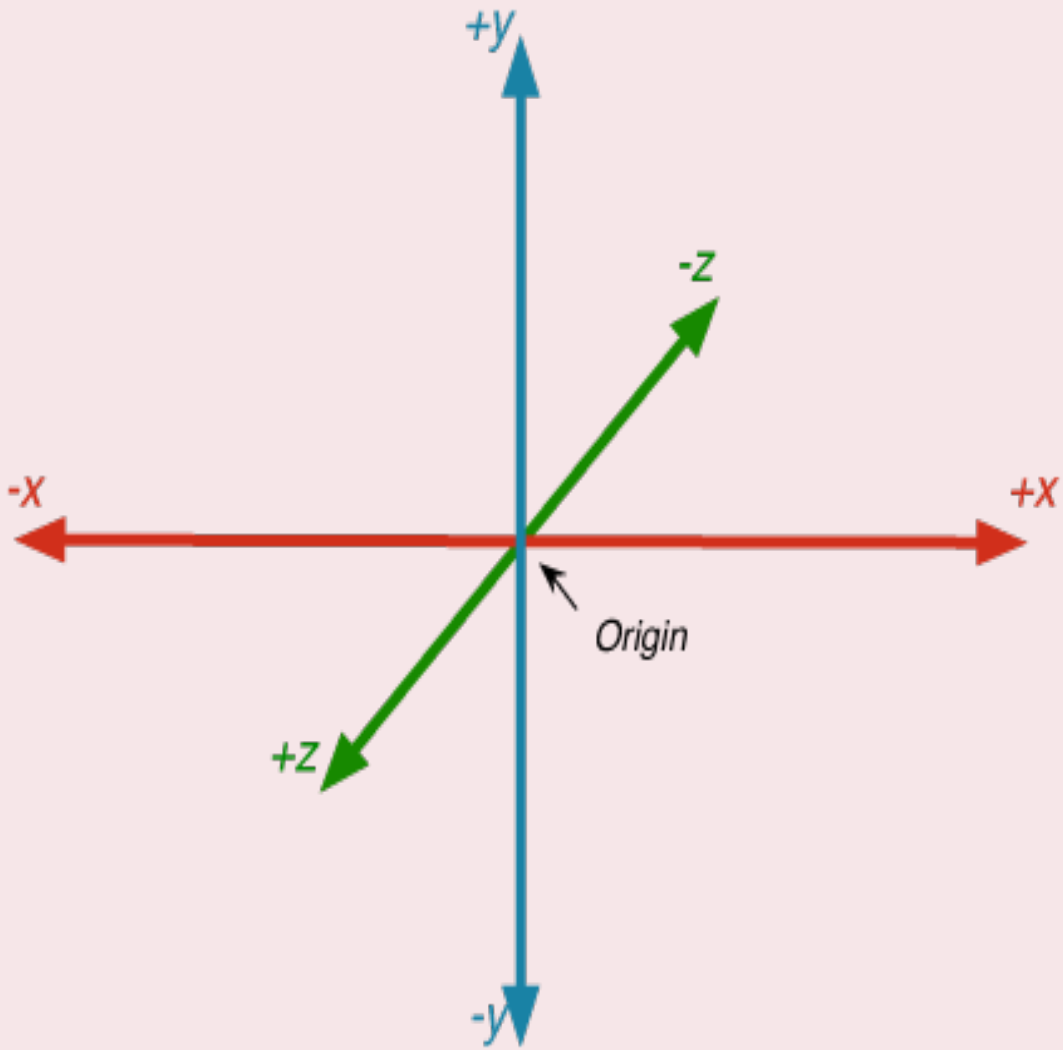
Universidad de Guanajuato

Noviembre 2010

Como se convierte un vértice en un pixel

La cámara desde que visualizaremos el mundo siempre esta colocada en el centro de coordenadas (0, 0, 0) y en vez de mover la cámara moveremos el mundo, es lo mismo mover la cámara hacia atrás que todo el mundo hacia adelante. Otra cuestión importante a tener en cuenta es que la zona que enfoca la cámara es el eje negativo de las **z**. Al estar enfocando de esta manera nos queda que el eje z positivo va de la pantalla hacia nosotros, el eje positivo de las **x** va de izquierda a derecha de la pantalla y el eje **y** positivo va de abajo a arriba en la pantalla.

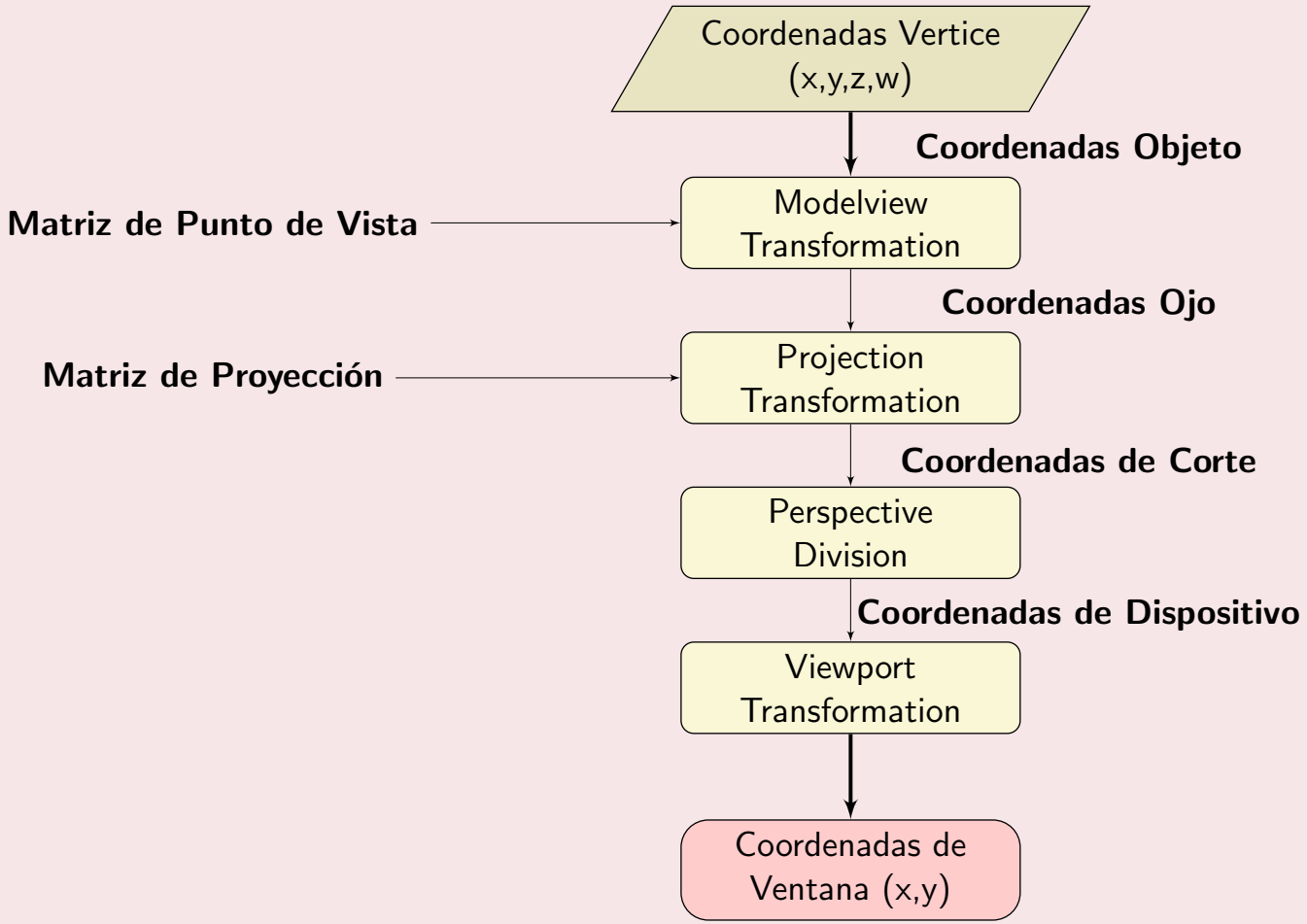
Right Hand



Proceso de Visualización

Un vértice especificado, en el sistema de coordenadas de OpenGL, se le aplican un serie de transformaciones hasta que se convierte en un pixel. Primero de todo se multiplica por la matriz de modelo vista (Modelview Matrix), a continuación se calcula la iluminación sobre el vértice si trabajamos con luces, después se le aplica la matriz de proyección (Projection Matrix) que permite realizar el clipping de los vértices que no son visibles. Una vez tenemos solo los vértices visibles se le aplica la división por la perspectiva (se utiliza el parámetro w para coordenadas homogéneas), por último se transforma a coordenadas ventana mediante la transformación de la ventana (Viewport Transformation).

Proceso de Visualización



3D$\longrightarrow$2D

Una vez hemos visto que transformaciones se aplican a un vértice para pasarlo desde coordenadas 3D a un punto en la pantalla veremos que funciones están relacionadas con este proceso, veremos tres apartados:

- Transformaciones para modificar el punto de vista.
- Modificar la proyección y el tipo de proyección.
- Funciones para modificar la zona de la ventana donde debe ir la escena renderizada.

Transformaciones para el Punto de Vista

Cada vértice antes de renderizarse se multiplica por una matriz guardada por OpenGL , MODELVIEW_MATRIX, esta matriz no obstante la podemos modificar, cargar la identidad o obtener los valores actuales:
Funciones existentes:

- glLoadIdentity(): Carga la matriz identidad.
- glMultMatrix{fd}(const TYPE *m): Multiplica por la matriz especificada por el puntero m, hay que tener en cuenta que los valores se encuentran en el siguiente orden:

$$M = \begin{bmatrix} m[0] & m[4] & m[8] & m[12] \\ m[1] & m[5] & m[9] & m[13] \\ m[2] & m[6] & m[10] & m[14] \\ m[3] & m[7] & m[11] & m[15] \end{bmatrix}$$

Funciones existentes

- glTranslate{fd}(TYPE x, TYPE y, TYPE z): Multiplica por una matriz que traslada el objeto por los valores x, y, z (o mueve el sistema de coordenadas local por los mismos valores).
- glRotate{fd}(TYPE angle, TYPE x, TYPE y, TYPE z): Multiplica la matriz activa por una matriz que rota un objeto en el sentido de las agujas del reloj sobre el eje que pasa por el origen y a través de el punto (x, y, z). El ángulo se especifica en grados.
- glScalefd(TYPE x, TYPE y, TYPE z): Multiplica la matriz activa por una matriz que escala por los valores x, y, z.

Funciones existentes

- gluLookAt(GLdouble eyex, GLdouble eyey, GLdouble eyez,GLdouble centerx, GLdouble centery, GLdouble centerz,GLdouble upx, GLdouble upy, GLdouble upz): Define una matriz de visualización y multiplica por la derecha la matriz activa. El punto desde el que se mira se especifica mediante las variables ejex, ejey, ejez. El punto al que se mira mediante las variables centerx, centery, centerz, y por ultimo upx, upy, upz define cual es la dirección hacia arriba de la cámara.

Transformaciones para el Punto de Vista

Cómo hemos visto siempre hacemos referencia a la matriz activa, ¿por qué? Porque como hemos dicho en el punto anterior tenemos dos matrices GL_MODELVIEW_MATRIX y GL_PROJECTION_MATRIX, para modificar una u otra debemos seleccionarla como matriz activa, realmente tenemos tres matrices, la tercera es la matriz para modificar las coordenadas de las texturas que veremos en la sesión correspondiente. Para cambiar la matriz activa lo hacemos con la función:

- glMatrixMode(GLenum mode): Especifica que matriz se modificará con las operaciones de matrices, usando el argumento GL_MODELVIEW, GL_PROJECTION, o GL_TEXTURE). Las siguientes transformaciones se aplicarán a la matriz que hayamos seleccionada, como se deduce solo se puede modificar una matriz al mismo tiempo. Por defecto se modifica la matriz de punto de vista (MODELVIEW) y todas

Transformaciones para la Proyección

Las transformaciones del apartado anterior se aplican en la mayoría de los casos a la matriz MODELVIEW (excepto glLoadIdentity que también se aplica a menudo), ahora veremos las transformaciones que llevan a cabo una proyección de la imagen (se pasa de tener un punto 3D a un punto 2D), estas transformaciones irán casi siempre acompañadas de glLoadIdentity y en la matriz PROJECTION. La matriz de proyección define el volumen de visualización, este volumen determina como se proyecta un objeto en la pantalla (proyección ortográfica o perspectiva) y define que objetos o partes de estos se eliminan de la imagen final al no ser visibles.

Proyecciones Ortográficas

En una proyección ortográfica el volumen de visualización es un paralepípedo rectangular, o de una manera mas informal una caja, en este tipo de proyección la distancia a la cámara no afecta como de grande se ve el objeto.
glOrtho(GLdouble left, GLdouble right, GLdouble bottom,GLdouble top, GLdouble near, GLdouble far): Multiplica por una matriz de proyección. El paralepípedo viene definido por los puntos (left, bottom, -near) y (right, top, -far), tanto near como far pueden ser valores positivos o negativos. Sin ninguna otra transformación la dirección de proyección es paralela al eje z, y el punto de vista mira hacia la parte negativa del eje z.

Proyecciones Perspectiva

El volumen definido por este tipo de proyección es una pirámide truncada, esto hace que los objetos aparezcan más pequeños a medida que se alejan del punto de vista. Para definir este tipo tenemos la siguiente función:

- glFrustum(GLdouble left, GLdouble right, GLdouble bottom,GLdouble top, GLdouble near, GLdouble far): Define un volumen de visión tal y como se presenta en la figura, todos los objetos que están mas cerca del plano near o más alejados que far no aparecen en la escena.

Proyecciones Perspectiva

- gluPerspective(GLdouble fovy, GLdouble aspect, GLdouble zNear,GLdouble zFar): Esta función aunque de menor posibilidad que la anterior resulta más intuitiva. El volumen viene definido por el ángulo de visión, variable fovy, en el plano x-z y la relación de aspecto entre ancho y alto (x/y) mediante la variable aspect. Las variables zNear y zFar como se deduce define la distancia a la que se encuentran los planos de corte perpendiculares al vector director de la cámara. Una vez más la parte que visualiza es la parte negativa del eje z.

Definición de la ventana de salida

El sistema operativo, no OpenGL, se encarga de gestionar las ventanas, una vez aclarado esto veremos como definir la región de la ventana en la que renderizaremos la escena. Por defecto la porción de ventana de salida esta ajustada a toda la region de la ventana de renderización, mediante el comando glViewPort se puede reducir a un área de dibujado menor.
glViewport(GLint x, GLint y, GLsizei width, GLsizei height): Los dos primeros parámetros especifican el punto 2D de la ventana donde empieza el área en que aparecerá la imagen renderizada, los otros dos especifican el tamaño.
Esto nos permite renderizar cuatro escenas en una misma ventana. Para ello renderizamos cuatro veces, y en cada una de las renderizaciones especificamos una region de viewport diferente.

Multiplicación de Matrices. Transformaciones consecutivas

Como sabemos disponemos de dos matrices de transformaciones que afectan al pixel, la matriz de MODELVIEW y PROJECTION. A cada una de ellas le podemos aplicar las transformaciones que hemos visto anteriormente (cargar identidad, traslación, rotación, escalado, proyección perspectiva, ...), estas transformaciones se aplican siempre sobre la matriz activa, que seleccionamos con la instrucción glMatrixMode. El orden con que se aplican estas transformaciones es importante, no es lo mismo aplicar una traslación y posteriormente un rotación que el orden inverso rotación y traslación.
El orden en que se aplican las transformaciones en OpenGL es al contrario de lo que parece intuitivo a primera vista.

Ejemplo

```
glLoadIdentity();
glTranslatef(0.5, 0.5, 1.0);
glRotatef(45.0, 1.0, 1.0, 1.0)

glBegin(GL_TRIANGLE);
glVertex3f(0.0, 0.0, 0.0);
glVertex3f(1.0, 0.0, 0.0);
glVertex3f(0.0, 1.0, 0.0);
glEnd();
```

Primero carga la matriz identidad, luego multiplica por una matriz de traslación y por ultimo una de rotación. El resultado es que primero rota el triángulo y posteriormente lo traslada, esto es debido a que las multiplicaciones las realiza por el lado derecho:
$I \cdot R \cdot T \cdot \text{vértice} = (I \cdot (R \cdot (T \cdot \text{vértice})))$ El vértice primero lo multiplica por la matriz T (de traslación) luego por R y finalmente por I.

Lectura

La lectura se realiza de abajo arriba, hasta encontrar una función glLoadIdentity o glLoadMatrix. Las transformaciones que haya después de la primitiva de dibujado glBegin - glEnd no se tienen en cuenta porque la primitiva ya ha sido renderizada.
Si deseamos realizar multiplicaciones por la izquierda de una matriz N podemos aplicar el siguiente método: leer la matriz actual M cargar la identidad multiplicar por la nueva transformación N multiplicar por la matriz M
En OpenGL seria:

```
GLFloat fMatrizM[16];
glLoadMatrixf(fMatrizM);
glLoadIdentity();
glMultMatrixf(fMatrizN), glTranslatef(x, y, z),
glScalef(sx, sy, sz) o glRotatef(ang, x, y, z);
glMultMatrixf(fMatrizM);
```

Pila de Matrices

En algunos casos nos puede interesar guardar la transformación que tenemos en la matriz activa para posteriormente recuperarla, por ejemplo si deseamos renderizar un coche formado por la carrocería y cuatro ruedas. El coche entero tendrá una transformación para colocarlo en el lugar (traslación + rotación) y cada una de las ruedas tendrá una transformación de rotación y transformación adicional que la colocara en relación al sistema de coordenadas del coche, para no tener que aplicar la misma transformación del coche a cada rueda podemos almacenarla en una pila y posteriormente recuperarla.

Nota

Disponemos de una pila de matrices para cada una de las matrices existentes: MODELVIEW, PROJECTION y TEXTURE.

Ejemplo

```
glPushMatrix(); // Guardamos la transformacion del mundo
glTranslatef(cocheX, cocheY, cocheZ);
glRotatef(angulo, ejeX, ejeY, ejeZ);
RenderCarroceria();

    glPushMatrix(); // Guardamos la transformacion del coche
    glTranslatef(rueda1X, rueda1Y, rueda1Z);
    glRotatef(anguloRueda1, eje1X, eje1Y, eje1Z);
    RenderRueda();
    glPopMatrix(); // Recuperamos la transformacion del coche

    glPushMatrix();
    glTranslatef(rueda2X, rueda2Y, rueda2Z);
    glRotatef(anguloRueda2, eje2X, eje2Y, eje2Z);
    RenderRueda();
    glPopMatrix();

    glPushMatrix();
    glTranslatef(rueda3X, rueda3Y, rueda3Z);
    glRotatef(anguloRueda3, eje3X, eje4Y, eje3Z);
    RenderRueda();
    glPopMatrix();

    glPushMatrix();
    glTranslatef(rueda4X, rueda4Y, rueda4Z);
    glRotatef(anguloRueda4, eje4X, eje4Y, eje4Z);
    RenderRueda();
    glPopMatrix();

glPopMatrix(); // Recuperamos la transformación del mundo
```

Objetos 3D. Z-Buffer

La librería GLUT incorpora unas funciones para generar solidos 3D. Estas funciones siempre renderizan el objeto en el punto (0, 0, 0), será pues trabajo del programador aplicar las transformaciones para colocarlo en la posición y orientación deseado.

```
glutSolidSphere, glutWireSphere
glutSolidCube, glutWireCube
glutSolidCone, glutWireCone
glutSolidTorus, glutWireTorus
glutSolidDodecahedron, glutWireDodecahedron
glutSolidOctahedron, glutWireOctahedron
glutSolidTetrahedron, glutWireTetrahedron
glutSolidIcosahedron, glutWireIcosahedron
glutSolidTeapot, glutWireTeapot
```

Objetos 3D. Z-Buffer

Para cada objeto tenemos dos posibles funciones, solid y wire:

```
Solid: renderización solida del objeto
Wire: renderización alambrica del objeto
```

Cuando renderizamos un objeto 3D, ya sea de la librería GLUT o nuestro, tenemos un problema, si renderizamos primero los polígonos próximos a la cámara y posteriormente los más alejados, los últimos en renderizarse se superponen a los anteriores. Necesitamos un algoritmo que haga visibles los polígonos más cercanos y oculte los más alejados. El algoritmo que utiliza OpenGL es Z-Buffer.

Objetos 3D. Z-Buffer

Mediante este algoritmo tenemos un buffer adicional que guardamos para cada pixel la distancia entre la cámara y el pixel 3D. Para cada pixel que renderiza comprueba que la distancia del nuevo pixel sea menor que la distancia almacenada en el Z-Buffer, si es así, pasa el test de profundidad y el pixel se renderiza, de lo contrario se rechaza. El pixel que pasa el test de profundidad modifica el buffer de color con su color y el Z-Buffer con la profundidad del mismo. Para que el algoritmo funcione además necesitamos una inicialización del Z-Buffer a la profundidad máxima que se consigue con la llamada glClear(GL_DEPTH_BUFFER_BIT).

Objetos 3D. Z-Buffer

El Z-Buffer tiene una precisión 8, 16, 24... bits que nos da menor o mayor precisión cuando tengamos polígonos a distancias similares. Para habilitar y deshabilitar el test de profundidad (Depth Test) tenemos:

`glEnable(GL_DEPTH_TEST)`: Habilita el test de profundidad
`glDisable(GL_DEPTH_TEST)`: Deshabilita el test de profundidad

Anteriormente hemos descrito la operación de comprobación del test de profundidad como un operación de menor que, el pixel entrante ha de tener menor distancia que el pixel ya existente, esta operación se puede cambiar:

`glDepthFunc(GLenum func)`: Permite cambiar la operación del test de profundidad entre `GL_NEVER`, `GL_LESS`, `GL_LEQUAL`, `GL_EQUAL`, `GL_GREATER`, `GL_NOTEQUAL`, `GL_GEQUAL`, `GL_ALWAYS`.

 Programming Principles and Practice Using C++, Bjarne Stroustrup.

 http://www.codeblocks.org

 http://www.wxwidgets.org

 (O'Reilly) Practical C Programming (3rd Edition)

 http://www.cplusplus.com

 http://es.wikipedia.org/wiki/GLUT