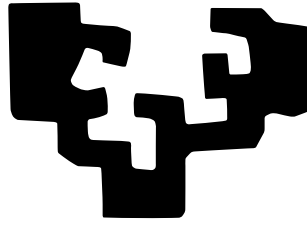


eman ta zabal zazu



Universidad
del País Vasco

Euskal Herriko
Unibertsitatea

Gráficos por Computador : Transformaciones geométricas

Finalizado el Domingo, Octubre 23, 2016

Cristina Mayor y Sergio Tobal

Contents

1.- Objetivos de esta práctica	3
2.- Métodos de resolución	3
3.- Solución elegida	4
4.- Explicación del código	5
5.- Ampliación a lo pedido	12
6.- Conclusiones	13
Referencias	14
Anexos	15
Anexo I: Manual del Usuario	15

1.- Objetivos de esta práctica

En esta práctica aprenderemos cómo utiliza OpenGL las matrices internamente para poder hacer transformaciones a los objetos.

Durante este proceso, también veremos la pila que usa OpenGL internamente, y como la usa para empilar y desempilar las distintas matrices de transformaciones que hemos ido realizando.

2.- Métodos de resolución

Hemos visto 3 posibles maneras de realizar esta practica, que detallamos a continuación:

1. **Calculando nosotros las matrices**, y las transformaciones. Este método es el mas complicado de los tres pero es el que mas versatilidad nos ofrece, ya que podremos aplicar algoritmos o mejoras que no estén en OpenGL. Por otro lado, es el mas proclive a errores debido a que somos nosotros los que deberemos calcular todas las matrices y sus transformaciones. La manera de crear la matriz sería la siguiente:

```
char[16] identityMatrix = {1.0f,0.0f,0.0f,0.0f,0.0f,1.0f,0.0f,0.0f,0.0f,0.0f,1.0f,0.0f,0.0f,0.0f,0.0f,1.0f}
```

2. **Usando la manera de OpenGL 4** en adelante, es decir, usando una librería matemática como GLM, usada debido a que en esas versiones se entiende que se van a usar shaders para programar las transformaciones. En caso contrario, se debe usar una librería externa a OpenGL para hacer los cálculos. De este modo, nos olvidamos de definir las matrices pero nos sigue permitiendo versatilidad en cuanto a las maneras de utilizarlas. El problema reside al tener que acordarnos de cómo funcionan las matrices en OpenGL, que se guarda siempre la traspuesta de la matriz usada.

```
glm::mat4 myIdentityMatrix = glm::mat4(1.0f);
```

3. **Depender de OpenGL** para generar las transformaciones. Esta es la manera mas cómoda ya que no debemos saber el código que se efectúa detrás de las funciones a las que llamamos. Nosotros conoceremos qué devuelven las funciones dependiendo de los parámetros de entrada pero no necesitamos conocer la matemática que hay detrás de los mismos. En este caso utilizamos los métodos que OpenGL nos ofrece para efectuar el cálculo matricial, es la forma mas sencilla pero que esta en estado *deprecated* ya que en las versiones mas nuevas no existen estos métodos.

```
glLoadIdentity();
```

Los ejemplos vistos en los 3 casos eran para crear una matriz identidad, es decir, la matriz que está formada por unos en la diagonal principal y por ceros en el resto de la matriz. Como podemos ver, cada caso es mas simple que el anterior, siendo el más sencillo el depender de OpenGL.

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} 1 * x + 0 * y + 0 * z + 0 * w \\ 0 * x + 1 * y + 0 * z + 0 * w \\ 0 * x + 0 * y + 1 * z + 0 * w \\ 0 * x + 0 * y + 0 * z + 1 * w \end{bmatrix} = \begin{bmatrix} x + 0 + 0 + 0 \\ 0 + y + 0 + 0 \\ 0 + 0 + z + 0 \\ 0 + 0 + 0 + w \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

3.- Solución elegida

Para efectuar esta práctica hemos decidido depender de las funciones que OpenGL nos ofrece para el cálculo matricial que hay que hacer para las transformaciones.

Al inicio del proyecto se nos ha entregado un programa a medio realizar (un código fuente) en el que podemos cargar distintos objetos 3D. Para realizar las pruebas necesarias y asegurarnos de que nuestro programa funcionaba correctamente hemos utilizado los ficheros abioia.obj y logoehu_ona.obj ya que estos dos tienen una dimensión similar y están bien definidos. Con este programa también podemos hacer zoom al entorno virtual donde cargamos los distintos objetos.

Se nos pide aplicar las transformaciones de escalado, rotación y traslación, la tecla TAB para movernos entre los distintos objetos, las tecla CTRL Z para deshacer las transformaciones realizadas sobre el objeto y, como suplemento, hemos añadido el CTRL Y rehacer los cambios de nuevo. Con el uso de las teclas de dirección AVPAG, REPAG, '+' y '-' podremos hacer las transformaciones de los objetos.

Para empezar, tenemos que saber como funciona OpenGL; tiene 3 matrices, una para modelos, otra para proyecciones, y dependiendo de la versión otra para la vista, aunque en algunas versiones es la misma que la de proyección.

En esta practica usaremos la matriz del modelo, cuyo nombre es GL_MODELVIEW_MATRIX, en ella se guardan todos los cambios que se han hecho a un objeto, como por ejemplo un escalado que se calcula usando la función glScale().

Internamente glScale() se encarga de multiplicar los valores que le pasemos, por la matriz usada por ultima vez. Así que es recomendable seleccionar la matriz del modelview para no tener problemas, esto lo haremos usando glMatrixMode(GL_MODELVIEW). De esta manera cuando se haga la multiplicación y se llame otra vez al dibujado, la matriz del modelview o modelado tendrá ya las coordenadas transformadas.

De esta manera, se genera una nueva matriz con los resultados de la multiplicación, que se coloca en la cima de la pila interna que tiene OpenGL, y que se encarga de guardar todos los cambios realizados a los objetos.

Tenemos también una pila propia dentro de cada objeto. En esta pila se almacena los cambios que se han ido realizando sobre el objeto seleccionado. Esta pila tiene una estructura de lista doblemente enlazada (DoubleLinkedList) , permitiendo así que el usuario pueda deshacer y rehacer sus cambios, cargando la matriz de un paso concreto y devolviendo el objeto al aspecto en ese paso.

4.- Explicación del código

Con el primer código que se nos dio hemos construido nuestro código. A lo largo de la práctica hemos ido cambiando el código hasta llegar al código final que presentamos en esta entrega. De este modo podemos ver los siguientes cambios:

1. Dado que utilizamos sistemas operativos diferentes las librerías que cada uno de nosotros utiliza son diferentes, por lo que necesitamos imports condicionales, de manera que dependiendo del sistema operativo del que se quiera ejecutar se llamen a unas librerías o a otras. Estos imports los podemos encontrar en la mayoría de nuestros archivos y tienen la forma de:

```
#ifndef __linux__
    #include <GL/glut.h>
    #include <GL/gl.h>
    #include <GL/glu.h>
5 #elif __APPLE__
    #include <GLUT/glut.h>
    #include <OpenGL/gl.h>
    #include <OpenGL/glu.h>
#endif
```

Este ejemplo está tomado del archivo `display.c`. Podemos observar que cuando estamos con sistema operativo linux solo se importará la librería `<GL/glut.h>`, `<GL/gl.h>` y `<GL/glu.h>`, en cambio, cuando estamos con el sistema operativo OS X necesitamos las librerías `<GLUT/glut.h>`, `<OpenGL/glu.h>` y `<OpenGL/gl.h>`.

Del mismo modo que las librerías son diferentes en los sistemas operativos, el comando en la terminal para compilar los archivos también son diferentes:

- Para linux: `gcc -lGL -lGL -lGLU -lglut *.c -I./ -o test.out`
- Para OS X: `gcc -w *.c -I./ -framework OpenGL -framework GLUT -o test`

2. En el archivo **definitions.h** hemos creado una nueva estructura para poder tener en el objeto una pila de las matrices de transformaciones por las que ha pasado. Esta pila será una lista doblemente enlazada que tiene la siguiente estructura:

```
struct typeNode {
    float m[16];
    struct typeNode *next;
    struct typeNode *prev;
5 };
typedef struct typeNode typeNode;
```

En esta nueva estructura, como podemos observar, tenemos una matriz 4x4 (16 valores), y dos apun­tadores, uno al siguiente (next) typeNode y otro al anterior (prev). De este modo, podremos tener guardadas todas las matrices de transformación de los objetos. En el momento de crear el struct se carga la identidad en el atributo m y tanto prev como next serán NULL.

3. Dentro de **io.h** hemos incluido la definición del método `void SpecialKeys(int key, int x, int y)`; ya que este método lo hemos incluido en `io.c`.

4. En **main.c**, dada la utilización de las teclas especiales, hemos incluido la definición del método que gestiona las teclas especiales *glutSpecialFunc(SpecialKeys)* y ampliado la ayuda que venia por defecto incluyendo las nuevas funciones.
5. Para poder gestionar la pila de matrices de un objeto hemos creado la clase **stack2.c**. Esta clase se encarga de gestionar las opciones de la pila; insertar al comienzo, borrar un elemento, liberar la memoria, etc.
 - *isEmpty()*: nos dice si la pila del objeto está vacía o no.
 - *length()*: devuelve la longitud de la pila.
 - *insertFirst(typeNode *v)*: este método tiene dos funciones dado que utilizamos también el ctrl + y. Por un lado libera todos los elementos tipo typeNode que quede antes de lo que nosotros consideramos el primer elemento de la pila (*_selected_object->pila*), y por el otro inserta en la primera posición de la pila el elemento que pasamos por parámetro.
 - *deleteFirst()*: se usa para poner en el atributo pila del objeto seleccionado el siguiente elemento que hubiera en la lista, es decir, la transformación anterior.
 - *deleteMiddle()*: se encarga de poner en el atributo pila del objeto seleccionado el elemento anterior que hubiera en la lista, es decir, rehacer la transformación que anteriormente se hubiera deshecho.
6. Hemos creado el archivo cabecera **stack2.h** y en él hemos incluido el nombre de los métodos que forma *stack2.c*.
7. En la clase **display.c** hemos hecho las siguientes modificaciones:
 - *Eliminación de loadIdentity()*: inicialmente al cargar cada objeto se iniciaba con una matriz identidad, para que al hacer multiplicaciones sobre ella se fueran produciendo las distintas transformaciones.

Ahora realizamos la carga de la matriz identidad en la pila de cada objeto, en el momento de la carga del fichero, y el resto de transformaciones según se van pulsando las teclas de dirección.

 - *Carga de la matriz del objeto*: relacionado a lo anterior hemos definido la matriz que se va dibujar, la cual sera inicialmente la matriz identidad, aquella que tiene 1 en la diagonal principal y 0 en el resto de elementos. Pero ahora tenemos una pila interna con todos los cambios, que usaremos para mostrar los distintos cambios que le ocurran al objeto.
8. En la clase **io.c** hemos hecho los siguientes cambios y hemos añadido el siguiente código:
 - En primer lugar, tuvimos que modificar las funciones principales del archivo fuente que nos dieron para que en ejecución se ajustaran a lo que se supone que hacían.

- *TAB*: para movernos entre los distintos objetos que hemos cargado utilizamos la tecla TAB. Con ella pasaremos al siguiente objeto cargado. En un principio (en el código fuente) si nos encontrábamos en el último elemento de la lista de objetos cargados, no podíamos pasar al primero, por eso incluimos una condición. Esta condición consiste en que si nos encontramos en el último elemento, el siguiente elemento sería el primer elemento de la lista. El código sería el siguiente:

```

if (_first_object != 0){
    _selected_object = _selected_object->next;
    if (_selected_object == 0)_selected_object = _first_object;
}

```

- *F, f*: Se nos indica que al presionar la tecla f o F se debería cargar un fichero *.obj. Este fichero va a estar dado mediante la ruta que especifiquemos nosotros, en el caso de que la ruta sea correcta y se proceda a dibujar el objeto, el código resultante sería:

```

/*Read OK*/
case 0:
    auxiliar_object->next = _first_object;
    _first_object = auxiliar_object;
5    _selected_object = _first_object;
    _selected_object->pila = (typeNode *) malloc(sizeof (typeNode));
    /* insertar la identidad en la pila */
    glGetFloatv(GL_MODELVIEW_MATRIX, _selected_object->pila->m);
    _selected_object->pila->next = NULL;
10    _selected_object->pila->prev = NULL;
    printf("%s\n", KG_MSSG_FILEREAD);
break;

```

Con esto inicializamos la pila de nuestro objeto, y creamos espacio en la memoria para almacenarlo, lo cual haremos usando la función malloc, cuyo parámetro será el indicador del espacio que deseamos reservar; una vez hemos creado el espacio, procedemos a insertar en el atributo m de la pila la matriz identidad; cuando ya hemos completado este paso, procedemos a poner tanto el puntero prev como el puntero next a NULL.

- *SUPR*: Al presionar esta tecla se nos pide que el objeto seleccionado sea eliminado. En este código hemos tenido que meter la condición de que el primer objeto sea distinto de 0, es decir, que solo se pueda eliminar un objeto en el caso de que haya al menos un objeto cargado.

```

if (_first_object != 0)
    if (_selected_object == _first_object) {
        _first_object = _first_object->next;
        free(_selected_object);
5        _selected_object = _first_object;
    } else {
        auxiliar_object = _first_object;
        while (auxiliar_object->next != _selected_object)
            auxiliar_object = auxiliar_object->next;
10        auxiliar_object->next = _selected_object->next;
        free(_selected_object);
        _selected_object = auxiliar_object;
    }
}

```

- *CTRL + +*: Con la utilización de estas dos teclas conjuntamente, deberemos reducir el volumen de visualización. De este modo, tomando como referencia el código que se nos daba de *CTRL + -* hemos logrado escribir el siguiente código, el cual hace lo que se nos pide.

```

if (glutGetModifiers() == GLUT_ACTIVE_CTRL) {
    wd=(_ortho_x_max-_ortho_x_min)*KG_STEP_ZOOM;
    he=(_ortho_y_max-_ortho_y_min)*KG_STEP_ZOOM;

5    midx = (_ortho_x_max+_ortho_x_min)/2;
    midy = (_ortho_y_max+_ortho_y_min)/2;
    _ortho_x_max = midx + wd/2;
    _ortho_x_min = midx - wd/2;
    _ortho_y_max = midy + he/2;
10   _ortho_y_min = midy - he/2;
}

```

- Una vez modificadas las funciones principales tuvimos que hacer las transformaciones. En este apartado se nos pedía que pudiéramos elegir una de las tres transformaciones para que se pudiera proceder a hacerla, elegir si se iba a hacer una transformación global o local y, dado que este proyecto va a ser un proyecto por fases, que se pudiera también elegir transformar el objeto, la cámara o la luz, aunque, para esta primera fase, solo se nos pedían las traslaciones del objeto.

Para poder gestionar que traslación se iba a realizar creamos la variable *estado*. Por defecto inicializada a 0, es decir, no hay ninguna transformación elegida. Tal como se nos plantea, la transformación se puede elegir:

- *M, m*: presionando esta tecla, lograremos cambiar el estado a 1, es decir, que se aplique la traslación.
- *B, b*: presionando esta tecla, lograremos cambiar el estado a 2, es decir, que se aplique la rotación.
- *T, t*: presionando esta tecla, lograremos cambiar el estado a 3, es decir, que se aplique el escalado.

Para poder gestionar si la transformación que se va a aplicar va a ser local (en el eje de coordenadas del objeto) o global (en el eje de coordenadas global) vamos a utilizar una variable llamada *glob_loc* que inicialmente va a estar a 0, es decir, que las transformaciones que se van a realizar son locales.

Para poder cambiar el valor de esta variable tenemos las siguientes posibilidades:

- *G, g*: Presionando esta tecla haremos que la variable *glob_loc* tome como valor 1 y que, por tanto, las transformaciones que sufra el objeto seleccionado sean globales.
- *L, l*: Presionando esta tecla haremos que la variable *glob_loc* tome como valor 0 y que, por tanto, las transformaciones que sufra el objeto seleccionado sean locales.

Para poder gestionar si las transformaciones se van a efectuar sobre el objeto, la cámara o la luz hemos creado la variable *mov*, inicialmente vale 0 y en esta primera fase del código está en desuso ya que no tenemos otro elemento a transformar más que el objeto. Aun así, hemos definido cómo va a cambiar el valor de la variable. Esta variable va a cambiar de la siguiente manera:

- *O, o*: Presionando esta tecla, haremos que la variable *mov* tome como valor 1, esto nos indicará que las transformaciones se han de efectuar en el objeto.

- K, k : Con esta tecla, la variable mov sera 2, es decir, haremos transformaciones en la cámara.
- A, a : Por ultimo, con a o A, asignaremos a mov 3, lo que establecerá el hacer transformaciones de luz.
- Otro de los puntos que se nos pedían era utilizar las teclas + y - para *escalar todos los ejes a la vez*. Para hacer este paso hemos utilizado las funciones que OpenGL nos ofrece y hemos acabado escribiendo el siguiente código:
 - A la hora de hacer escalado (ampliar) los tres ejes:

```
case +:
    struct typeNode *matriz = (typeNode *) malloc(sizeof (typeNode));
    glLoadMatrixf(_selected_object->pila->m);
    glScalef(2.0f, 2.0f, 2.0f);
5    glGetFloatv(GL_MODELVIEW_MATRIX, matriz->m);
    insertFirst(matriz);
    glPopMatrix();
    break;
```

- A la hora de hacer escalado (disminuir) los tres ejes:

```
case -:
    struct typeNode *matriz = (typeNode *) malloc(sizeof (typeNode));
    glLoadMatrixf(_selected_object->pila->m);
    glScalef(0.5f, 0.5f, 0.5f);
5    glGetFloatv(GL_MODELVIEW_MATRIX, matriz->m);
    insertFirst(matriz);
    glPopMatrix();
    break;
```

Podemos observar que ambos códigos son iguales salvo en la asignación que se hace en la función `glScalef()`. Inicialmente crearemos el espacio para la estructura `typeNode`, esta estructura será introducida en la pila de matrices de nuestro objeto. Una vez hemos creado el espacio, procedemos a cargar en la modelview la matriz a la que apunta nuestra pila. Haremos el escalado sobre esa matriz, y esa nueva matriz la vamos a cargar en la estructura para la que hemos guardado memoria. Una vez finalizado este paso, procederemos a insertar esta nueva matriz en nuestra pila y quitaremos la matriz que hemos puesto en el modeview.

Para el caso del -, el procedimiento será el mismo solo que el escalado será a la mitad en vez de al doble de las dimensiones del objeto.

- Otro de los puntos a tratar en esta primera fase del proyecto es poder hacer *transformaciones* (escalado, rotación y traslación) *globales y locales*.

En el caso de las traslaciones locales, haremos referencia al código insertado para el + y el -.

A la hora de hacer un escalado, la estructura que vemos en esos puntos es la que se va a llevar a cabo.

A la hora de hacer una transformación, se seguirán todos los puntos anteriormente citados con una leve modificación. En la línea 4 hemos definido `glScalef()` para hacer el escalado, ahora nos encontramos ante una traslación por lo que sustituiremos esa línea de código por: `glTranslatef(a.0f, b.0f, c.0f)`. En el caso de que queramos trasladar sobre el eje x (nosotros hemos decidido trasladar

de unidad en unidad, sin utilizar los decimales) el valor de 'a' sería 1/-1 y el valor de b y c sería 0, es decir, tomará por valor 1 aquel eje sobre el que queramos trasladar.

A la hora de hacer una rotación, seguirán todos los puntos citados en el subapartado anterior con una modificación. En vez de utilizar `glScalef()` utilizaremos `glRotatef(a.0f,b.0f, c.0f, d.0f)`. En esta función el valor de a será el número de grados que se quiera rotar el elemento y, al igual que en el caso de la traslación, tomarán como valor 1/-1 (es el estándar que nosotros hemos utilizado) aquellos ejes sobre los que se quiera hacer el giro siendo 'b' el referente al eje X, 'c' el referente al eje Y y 'd' el referente al eje Z.

En el escalado existen tres parámetros que se le pasan a la función, todos ellos tendrán que ser 1.0f exceptuando aquel que se quiera ampliar o reducir que en nuestro caso es 1.5f y 0.75f.

En el caso de las transformaciones globales el código cambia un poco y en el caso del escalado será el siguiente:

```
5 struct typeNode *matriz = (typeNode *) malloc(sizeof (typeNode));
   glMatrixMode(GL_MODELVIEW);
   glLoadIdentity();
   glScalef(1.0f,1.5f,1.0f);
   glMultMatrixf(_selected_object->pila->m);
   glGetFloatv(GL_MODELVIEW_MATRIX, matriz->m);
   insertFirst(matriz);
   glutPostRedisplay();
```

Como podemos observar, hay ciertos puntos que se repiten pero también hay variaciones en el código. Uno de los cambios más significativos es que la multiplicación de las matrices se hace por el lado contrario, de este modo el código cambia de la siguiente manera:

Inicialmente creamos el espacio para la estructura `typeNode` que contiene la matriz que vamos a generar. Cargamos la identidad en la `ModelviewMatriz` y multiplicamos la identidad por la matriz que hace la traslación (en nuestro caso el escalado sobre el eje Y). Procederemos a multiplicar esa matriz por la matriz de nuestra pila. El resultado obtenido lo guardaremos en la matriz de la estructura `typeNode` para la que hemos reservado memoria. Una vez guardada, la añadiremos a nuestra pila.

- Por ultimo tenemos el *deshacer* y *rehacer* de las *acciones* realizadas por el usuario.

Inicialmente estas acciones se suponía que iban a ser efectuadas mediante CTRL Z (deshacer) y CTRL Y (rehacer). Dados los problemas surgidos en el laboratorio para deshacer tendremos que presionar las teclas 'u' o 'U' y, para rehacer la teclas 'r' o 'R'.

Para efectuar estas acciones, haremos uso de las matrices alojadas en los nodos de la pila del objeto seleccionado, seleccionando el nodo siguiente en el caso de querer deshacer un cambio, o el nodo anterior en el caso de rehacer. De esta manera nos olvidamos de tener que usar multiplicaciones o revertir los cambios hechos, simplemente cogemos la forma que tenia previamente el objeto y lo cargamos. Al final se llamara a la función `glutPostRedisplay()` que se encargara de dibujar el estado actual de la matriz.

```
case 'U': // Deshacer
case 'u':
    if (_selected_object)
        if (_selected_object->pila->next != NULL)
            _selected_object->pila = _selected_object->pila->next;
5 break;

case 'R': // Rehacer
case 'r':
10    if (_selected_object)
        if (_selected_object->pila->prev != NULL)
            _selected_object->pila = _selected_object->pila->prev;
break;
```

5.- Ampliación a lo pedido

1. *Rehacer*: Usando las teclas 'R' o r' el usuario podrá rehacer los cambios deshechos.
2. *Conditional imports*: Al encontrarnos con el problema de que nuestro código variaba dependiendo del sistema operativo usado, hemos investigado como se hace un programa multiplataforma, ya que no estábamos por la labor de andar copiando y pegando las diferentes líneas todos los días, y hemos descubierto la existencia de `#ifdef` y `#elif` así como los descriptores `__linux__`, `__APPLE__` y `__WIN32__` o `__WIN64__` dependiendo de la versión de Windows que usemos.

6.- Conclusiones

Aunque el inicio ha sido un poco complicado, en especial porque no habíamos usado C hasta ahora y no terminábamos de entender como trabajar con punteros, con ayuda de nuestros compañeros hemos conseguido superar ese bache. Aunque luego nos hemos encontrado con algún otro problema, los hemos superado y nos ha gustado llegar al final del ejercicio, ver lo que hemos creado y sentirnos orgullosos de ello.

Hemos aprendido como se pueden utilizar funciones que existen en librerías que necesitábamos, y que nos ayudan a quebrarnos menos la cabeza logrando un resultado igual (o mejor) que en el caso de intentar programar nosotros esas mismas funciones (sobre todo en el caso de las transformaciones).

Referencias

[vertice-pixel] [Como pasar de vértice a pixel](#)

[espacio3D] [Moviendonos en un espacio 3D](#)

[teclado] [Teclado y funciones especiales](#)

Anexos

Anexo I: Manual del Usuario

El usuario podrá usar las siguientes teclas para manejar el programa:

- ?: Visualizar la ayuda
- ESC: Finalizar la ejecución de la aplicación
- F,f: Carga de objeto desde un fichero *.obj
- TAB: Seleccionar siguiente objeto (de entre los cargados)
- SUPR: Eliminar objeto seleccionado
- CTRL + +: Reducir el volumen de visualización
- CTRL + -: Incrementar volumen de visualización
- M,m: Activar traslación.
- B,b: Activar rotación.
- T,t: Activar escalado.
- G,g: Activar transformaciones en el sistema de referencia del mundo. (transformaciones globales)
- L,l: Activar transformaciones en el sistema de referencia local del objeto.
- U,u: Deshacer.
- R,r: Rehacer.
- UP: Trasladar +Y; Escalar + Y; Rotar +X.
- DOWN: Trasladar -Y; Escalar - Y; Rotar -X.
- RIGHT: Trasladar +X; Escalar + X; Rotar +Y.
- LEFT: Trasladar -X; Escalar - X; Rotar -Y.
- AVPAG: Trasladar +Z; Escalar + Z; Rotar +Z.
- REPAG: Trasladar -Z; Escalar - Z; Rotar -Z.
- +: Escalar + en todos los ejes. (solo objetos)
- -: Escalar - en todos los ejes. (solo objetos)

En primer lugar el usuario deberá conocer su sistema operativo para saber que comando utilizar a la hora de compilar el código.

Deberá abrir la terminal, colocarse en la carpeta que contiene los ficheros necesarios para la compilación y posterior ejecución del programa y, dependiendo de en que sistema operativo esté, deberá ejecutar los siguientes comandos:

- En Linux: `gcc -lGL -lGL -lGLU -lglut *.c -I./ -o test`
- En MAC OS X: `gcc -w *.c -I./ -framework OpenGL -framework GLUT -o test`

A la hora de ejecutar el programa habrá que ejecutar el comando `./test` y podremos observar la siguiente pantalla:



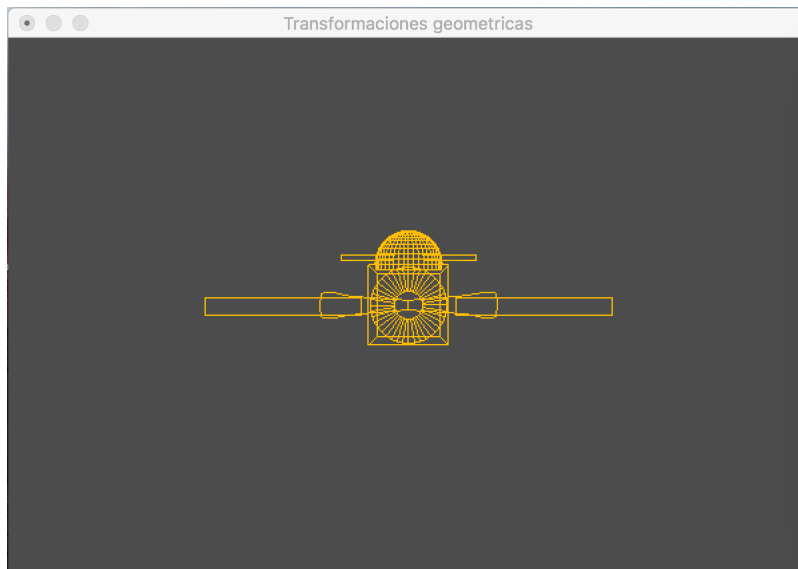
En la terminal observaremos un resumen de las funciones que tiene nuestro programa.

```
exercise2 — test — 80x24
Resumen de funciones
<?>      Lanza esta ayuda.
<ESC>     Salir del programa.
<F>       Cargar objeto.
<TAB>     Moverse entre los objetos cargados en escena.
<SUPR>    Borrar el objeto seleccionado.
<CTRL + -> Acercar zoom.
<CTRL + +> Alejar zoom.
<M,m>     Activar traslación.
<B,b>     Activar rotación.
<T,t>     Activar escalado.
<G,g>     Activar transformaciones globales.
<L,l>     Activar transformaciones locales.
<U,u>     Deshacer.
<R,r>     Rehacer.
<UP>      Trasladar +Y; Escalar + Y; Rotar +X.
<DOWN>    Trasladar -Y; Escalar - Y; Rotar -X.
<RIGHT>   Trasladar +X; Escalar + X; Rotar +Y.
<LEFT>    Trasladar -X; Escalar - X; Rotar -Y.
<AVPAG>   Trasladar +Z; Escalar + Z; Rotar +Z.
<REPAG>   Trasladar -Z; Escalar - Z; Rotar -Z.
<+>       Escalar + en todos los ejes del objeto.
<->       Escalar - en todos los ejes del objeto.
```

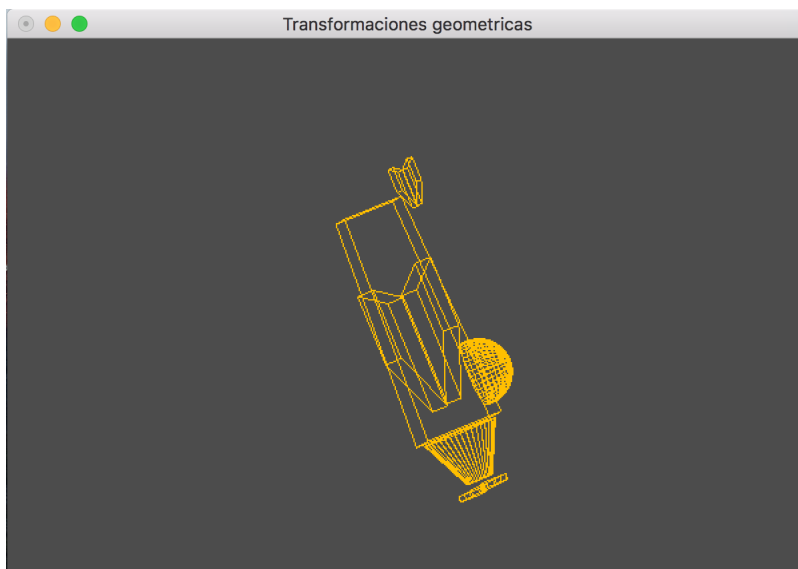
Para poder iniciar nuestro proceso de transformación de un objeto, es imprescindible insertar un objeto. La manera de insertar un objeto en nuestra pantalla es presionando la tecla `f` ó `F`.

Una vez presionada esta tecla, por la terminal se nos indicará que debemos introducir un path donde haya un objeto. Esta es la ruta que tiene el objeto.

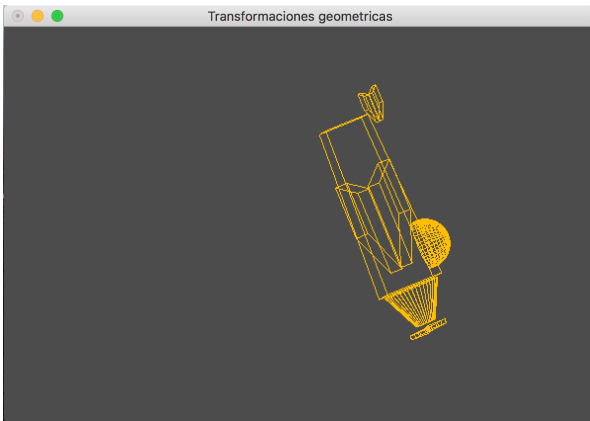
Para hacer el ejemplo nosotros vamos a utilizar el objeto abioia.obj. Al cargarlo podremos observar:



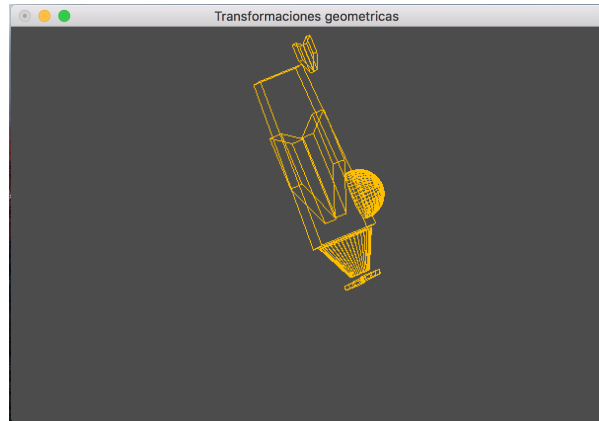
Por defecto solo se pueden utilizar las teclas '+' y '-' para escalar todos los ejes del objeto. Presionando '+' agrandaremos los objetos y presionando '-' decrementaremos los mismos. Para poder transformar el objeto de cualquier manera tendremos que presionar las teclas que hemos mostrado en las funciones anteriormente. Tal como se muestra podemos hacer transformaciones locales o globales. La diferencia de las mismas es la siguiente: Si tenemos el objeto en una posición



La transformación local hacia arriba sería lo que observamos en la figura de la izquierda y la transformación global hacia arriba lo que observamos en la figura de la derecha:



(a) Transformacion local



(b) Transformacion global

Moviéndose hacia arriba

A partir de ahora, y con el uso de las flechas de dirección del teclado, la tecla AVPAG y la tecla REPAG ya se podría mover los objetos cargados.

Del mismo modo que se ha cargado un objeto, volviendo a repetir el proceso de carga se pueden cargar tantos objetos como se deseen y, para moverse entre ellos solo habrá que presionar la tecla TAB. En el caso de que se quiera suprimir el objeto seleccionado (aquel que está en color naranja) habrá que presionar la tecla SUPR.