eman ta zabal zazu

Universidad del País Vasco

Euskal Herriko Unibertsitatea

Trabajo de transformaciones en OpenGL

Ignacio Belzunegui y Aitor Domec

December 23, 2018

1 Objetivos de nuestra aplicación

Nuestra aplicación tiene que ser capaz de realizar transformaciones respecto a objetos cargados en OpenGL. Estas transformaciones tienen que poder ser realizadas respecto al objeto y respecto al mundo, las transformaciones permitidas son las siguientes: escalado, rotacion y traslación. Ademas de esas transformaciones también puede aumentar o disminuir nuestro punto de vista del mundo y deshacer los cambios realizados.
hemos dividido los cambios que hemos realizado en 5 pasos.

1. **Aumentar zoom.**
2. **Crear la estructura de lista de matrices.**
3. **Inicializar correctamente los objetos.**
4. **Crear/completar el teclado.**
5. **Crear las funciones necesarias para las transformaciones.**

2 Manual de usuario

Para compilar el programa debemos de compilar main.c, display.c, io.c y load_obj_joseba.c por supuesto teniendo en cuenta que usamos OpenGL, el comando para compilar es el siguiente:
main.c -lGL -lGLU -lglut -lm display.c io.c load_obj_joseba.c
Este comando generará el ejecutable **a.out**.
Una vez ejecutado el programa el usuario verá en el terminal la guía de ayuda que explica los comandos existentes y sus utilidades. También verá una ventana emergente donde se visualizarán todos los objetos y transformaciones pertinentes.
Primero se deberá de cargar un objeto, de lo contrario no nos permitirá realizar transformaciones. Para cargar un objeto deberá de escribir el path al objeto, dandole a la "f" antes, nosotros recomendamos el uso del siguiente comando para ahorrarnos el tener que escribir el path cada vez que cargamos un objeto: **cp objects/abioia.obj ./a** donde objects/abioia.obj es el path al objeto deseado. De esta forma para cargar un objeto bastará con escribir únicamente "a".
Una vez cargado el objeto tenemos acceso completo a las transformaciones, de forma predefinida los cambios se realizarán respecto al objeto, pero esto podrá ser cambiado a voluntad en cualquier momento. Los comandos permitidos son los siguientes:

1. "?" Visualizar ayuda.
2. "ESC" Salir del programa.
3. "f / F" Cargar un objeto.
4. "TAB" Cambiar la seleccion de un objeto cargado.
5. "DEL" Borrar el objeto seleccionado.
6. "CTRL + -" Aumentar el zoom.
7. "CTRL + +" Reducir el zoom.
8. "l / L" Activar transformaciones respecto al objeto.
9. "g / G" Activar transformaciones respecto al mundo.
10. "m / M" Activar la traslacion.
11. "b / B" Activar la rotacion.

12. ”t / T” Activar el escalado.

Para realizar las tranformaciones deseadas debemos de pulsar las flechas del teclado.

3 Cambios realizados

1. **Aumentar zoom.**
   El código para reducir el zoom ya lo tenemos, el único cambio necesario es cambiar una división por una multiplicación. Sabemos que KG_STEP_ZOOM tiene un valor de 0.75, lo podemos ver en 'definitions.io'. Sabiendo esto, si divides entre KG_STEP_ZOOM vamos a aumentar los valores correspondientes, ya que es un número menor a 1. Por ello si queremos aumentar el zoom, deberemos de reducir los puntos, esto se hace a través de la multiplicación.

```
#include "definitions.h"
#include "load_obj.h"
#include <GL/glut.h>
#include <stdio.h>
...
...
...

case '+':
          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;
            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;
            _ortho_y_min = midy - he/2;
        }
        break;
...
...
...
```

2. **Crear la estructura de lista de matrices.**
   Vamos a crear una estructura para contener una lista de matrices donde guardaremos todos los cambios que hemos realizado sobre el objeto. Operaremos con esta lista como si fuera una pila.
   La estructura de la lista contiene un array (que actua como una matriz) y un puntero o otra estructura igual a si misma que contendrá la siguiente matriz. A esta estructura la llamamos 'matrix_l'. Después de esto solo queda actualizar el objeto 'object3d' para que también contenga

esta lista de matrices. Cabe destacar que estos cambios son realizados en el fichero 'definitions.h'. Tras realizar estos cambios deberemos de actualizar la forma de inicializar los objetos, que explicaremos en el siguiente apartado.

```
typedef struct matrix_l{
    GLdouble matrix[16];  /*Matriz de 4x4 que guarda la matriz activa*/
    struct matrix_l *next;/*Lista de matrices del objeto*/
} matrix_l;

/***************************
 * Structure to store a     *
 * pile of 3D objects       *
 ***************************/
typedef struct object3d{
    GLint num_vertices;        /* number of vertices in the object*/
    vertex *vertex_table;      /* table of vertices */
    GLint num_faces;           /* number of faces in the object */
    face *face_table;          /* table of faces */
    point3 min;                /* coordinates' lower bounds */
    point3 max;                /* coordinates' bigger bounds */
    matrix_l *matrix;          /* Lista de matrices de estado */
    struct object3d *next;     /* next element in the pile of objects */
}object3d;
```

3. **Inicializar correctamente los objetos.**
   Tras realizar los cambios necesarios para tener la pila de matrices deberemos de inicializar el objeto correctamente, tenemos que actualizarlo porque ahora tiene un nuevo parámetro, la lista de matrices llamada *matrix*.
   Para ello, en el archivo 'load_obj_joseba.c' deberemos de crear una lista de matrices y asignarle un espacio de memoria a traves de la función 'malloc'.

```
matrix_l *matrix;
...
matrix = (matrix_l *) malloc(sizeof (matrix_l));
```

   Despues de asignarle el espacio de memoria correspondiente, debemos de inicializar la lista de matrices de forma correcta, la matriz inicial será la matriz de identidad y apuntará a 0 porque al actuar como una pila no meteremos elementos por debajo de la misma. Para guardar la matriz de identidad vamos a cargarla en OpenGL a través de la función 'glLoadIdentity()' y luego vamos a obtener la matriz de identidad que acabamos de cargar y la vamos a guardar a través de la función 'glGetDoublev(GL_MODELVIEW_MATRIX, X)' que obtiene la matriz cargada en el GL_MODELVIEW y la guarda en la variable 'X'.

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
```

```
glGetDoublev(GL_MODELVIEW_MATRIX,matrix->matrix);
matrix->next=0;
```

4. **Crear/completar el teclado.**
   El teclado del que ya disponemos lee de forma incorrecta las flechas necesarias para realizar las transformaciones, por ello vamos a crear una función que lea esas teclas especiales. La función encargada de leerlas es 'void special_keyboard(int key, int x, int y)', para que esta función las lea deberemos de especificarlo en el archivo 'main.c'. Esto lo hacemos a traves de la siguiente funcion:'glutSpecialFunc(special_keyboard)' que avisa que la función 'special_keyboard' va a leer teclas especiales.

   Cabe destacar la existencia de las variables 'modo' y 'modo2'. La primera es la encargada de decidir si se quiere trasladar (modo=1), rotar (modo=2) o escalar (modo=3). La segunda por otro lado es la encargada de decir si se hacen los cambios respecto al mundo (modo2=1) o respecto al objeto (modo2=0), pero esta variable es usada en las funciones encargadas de realizar las transformaciones que se explicarán en el siguiente apartado.

```
void special_keyboard(unsigned char key, int x, int y) {
  switch(key) {
      case GLUT_KEY_UP:
          if(modo==1) {
              printf("Usted acaba de pulsar Flecha Arriba con
              el modo de Traslacion activado\n");
              traslate(0,KG_STEP_MOVE,0);
          }
          if(modo==2) {
                printf("Usted acaba de pulsar Flecha Arriba con
                el modo de Rotacion activado\n");
                rotate(KG_STEP_ROTATE,KG_STEP_MOVE, 0, 0);
            }

            if(modo==3) {
                printf("Usted acaba de pulsar Flecha Arriba con
                el modo de Escalado activado\n");
                scale(1, KG_STEP_SCALE, 1);
            }
        break;
        case GLUT_KEY_DOWN:
            if(modo==1) {
                printf("Usted acaba de pulsar Flecha Abajo con
                el modo de Traslacion activado\n");
                traslate(0,-KG_STEP_MOVE,0);
            }
            if(modo==2) {
                printf("Usted acaba de pulsar Flecha Abajo con
                el modo de Rotacion activado\n");
                rotate(KG_STEP_ROTATE,-KG_STEP_MOVE, 0, 0);
            }
```

```
            if(modo==3) {
                printf("Usted acaba de pulsar Flecha Abajo con
                el modo de Escalado activado\n");
                scale(1, 1/KG_STEP_SCALE, 1);
            }
        break;
        case GLUT_KEY_LEFT:
            if(modo==1) {
                printf("Usted acaba de pulsar Flecha Izquierda con
                el modo de Traslacion activado\n");
                traslate(-KG_STEP_MOVE,0,0);
            }
            if(modo==2) {
                printf("Usted acaba de pulsar Flecha Izquierda con
                el modo de Rotacion activado\n");
                rotate(KG_STEP_ROTATE,0,-KG_STEP_MOVE, 0);
            }

            if(modo==3) {
                printf("Usted acaba de pulsar Flecha Izquierda con
                el modo de Escalado activado\n");
                scale(1/KG_STEP_SCALE,1, 1);
            }
        break;
        case GLUT_KEY_RIGHT:
            if(modo==1) {
                printf("Usted acaba de pulsar Flecha Derecha con
                el modo de Traslacion activado\n");
                traslate(KG_STEP_MOVE,0,0);
            }
            if(modo==2) {
                printf("Usted acaba de pulsar Flecha Derecha con
                el modo de Rotacion activado\n");
                rotate(KG_STEP_ROTATE, 0, KG_STEP_MOVE, 0);
            }

            if(modo==3) {
                printf("Usted acaba de pulsar Flecha Derecha con
                el modo de Escalado activado\n");
                scale(KG_STEP_SCALE,1, 1);
            }
        break;

    }
    glutPostRedisplay();
}
```

En la función de teclado normal simplemente hemos hecho que lea las teclas que el enunciado menciona, cambiando los valores de 'modo' y 'modo2' de forma correspondiente e imprimiendo la acción que realiza la tecla que se acaba de pulsar.
Cabe destacar la existencia de la función *deshacer()*, usada a la hora de pulsar *control + z*, esta es la función encargada de deshacer un cambio volviendo a la matriz de estado anterior, esta función será explicada en el siguiente apartado.

```
void keyboard(unsigned char key, int x, int y) {
```

```
    ...
    ...
    ...
    case 'g':
    case 'G':
        printf("Usted acaba de activar el sistema de
        referencia al del mundo\n");
        modo2=1;
        break;
    break;
    case 'l':
    case 'L':
        printf("Usted acaba de activar el sistema de
        referencia al del objeto\n");
        modo2=0;
        break;
    break;

    case 'b':
    case 'B':
        printf("Usted acaba de activar el modo de rotacion\n");
        modo=2;
        break;
    break;

    case 't':
    case 'T':
        printf("Usted acaba de activar el modo de escalado\n");
        modo=3;
        break;
    break;

    case 'm':
    case 'M':
        printf("Usted acaba de activar el modo de traslacion\n");
        modo=1;
        break;
    break;

    case 26: //Control + z Tenemos que volver al estado anterior
        printf("Deshacer cambios\n");
        deshacer();
    break;
    ...
    ...
    ...
}
```

5. **Crear las funciones necesarias para las transformaciones.**
   En este apartado vamos a explicar las funciones que hemos creado para el correcto funcionamiento de las transformaciones. Tenemos que realizar tres transformaciones, escalado, traslación y rotación. Estos cambios serán realizados a traves de matrices de transformación; es importante saber que las operaciones con matrices no cumplen la propiedad distributiva (excepto las operaciones con la matriz de identidad) de manera que no es lo mismo 'A*B' que 'B*A'. Esto es algo importante, ya que si multiplicamos nuestra matriz de estado actual por la nueva

matriz de cambios por la izquierda, realizaremos el cambio respecto al objeto y si lo multiplicamos por la derecha realizaremos el cambio respecto al mundo.

Para realizar el **cambio respecto al objeto**, basta con cargar la matriz de estado actual, usar la función correspondiente (*glTraslatef(float a, float b, float c)*, *glScalef(float a, float b, float c)* o *glRotatef(float angle,float a, float b, float c)*) y guardar la nueva matriz en la pila a través de la función 'guardar_estado()' que hemos creado.

```
void guardar_estado()
{
    if(modog==0) { //Guardamos los cambios realizados sobre el objeto
        matrix_l *matrizaux;
        matrizaux = (matrix_l *) malloc(sizeof (matrix_l));
        matrizaux->next=_selected_object->matrix;
        //Guardamos la matriz cargada dentro de la auxiliar
        glGetDoublev(GL_MODELVIEW_MATRIX,matrizaux->matrix);
        //Hacemos que el objeto apunte a la matriz auxiliar
        _selected_object->matrix=matrizaux;
    }
    if(modog==1) { //Guardamos los cambios realizados sobre la camara
        matrix_l *matrizaux;
        matrizaux = (matrix_l *) malloc(sizeof (matrix_l));
        matrizaux->next=camaraG->matrix;
        glGetDoublev(GL_MODELVIEW_MATRIX,matrizaux->matrix);
        camaraG->matrix=matrizaux;
    }
}
```

Creamos la lista de matrices matrizaux, donde guardaremos tanto la matriz actual de la cámara global (*camaraG*) como la matriz activa de transformaciones de OpenGL (*GL_MODELVIEW_MATRIX*). Después guardamos la matriz de cámaraG en la siguiente matriz a la actual de matrizaux (*matrizaux*). A continuación guardamos la matriz actual de transformaciones de OpenGL en matrizaux (*glGetDoublev(GL_MODELVIEW_MATRIX , matrizaux)*).

Finalmente hacemos que camaraG apunte a matrizaux, la cual apunta a la instancia anterior de camaraG (es decir, a camaraG sin estar actualizado) ($camaraG = matrizaux$)

Dicho de otra manera: creamos una nueva lista auxiliar de matrices y le asignamos un espacio de memoria a través de un *malloc*. Luego hacemos que apunte a la matriz actual y guardamos la matriz cargada en estos momentos (a través de la función *glGetDoublev()*) en la auxiliar. Finalmente asignamos esta nueva lista a la del objeto seleccionado

y ya habriamos terminado.

Por otro lado, para realizar los **cambios respecto al mundo** deberemos de conseguir multiplicar por la derecha. Esto lo vamos a conseguir cargando primero la matriz de identidad, operando sobre ella con la función correspondiente (scale(float a, float b, float c), rotate(float angle,float a, float b, float c) o traslate(float a, float b, float c)), y multiplicando la matriz de estado actual de la pila por la matriz cargada (la mencionada la matriz actual de transformaciones de OpenGL), esto lo haremos a traves de la función *glMultMatrixd(double *m)*, donde 'm' es la matriz correspondiente al estado actual del objeto. Tras esto guardamos el cambio realizado con la función *guardar_estado()*.
Cabe destacar que si no hay un objeto seleccionado (esto solo es posible si no hay objetos cargados) el programa nos avisará de ello y no realizará ninguna transformación, ya que de lo contrario nos daría un error de segmentación.

```
void traslate(float a, float b, float c)
{
        if(_selected_object!=NULL) {
            //Hacemos el cambio respecto al objeto
            if(modo2==0){
                printf("Traslacion respecto al objeto\n");
                glMatrixMode(GL_MODELVIEW);
                glLoadMatrixd(_selected_object->matrix->matrix);
                glTranslatef(a, b, c);
                guardar_estado();
            }
            //Hacemos el cambio respecto al mundo
            else {
                printf("Traslacion respecto al mundo\n");
                glMatrixMode(GL_MODELVIEW);
                glLoadIdentity();
                glTranslatef(a, b, c);
                glMatrixMode(GL_MODELVIEW);
                glMultMatrixd(_selected_object->matrix->matrix);
                guardar_estado();
            }
        }
        else {
            printf("ERROR: No hay ningun objeto cargado!\n");
        }
}

void scale(float a, float b, float c)
{
    if(_selected_object!=NULL) {
    //Hacemos el cambio respecto al objeto
        if(modo2==0){
            printf("Escalado respecto al objeto\n");
            glMatrixMode(GL_MODELVIEW);
```

```
            glLoadMatrixd(_selected_object->matrix->matrix);
            glScalef(a, b, c);
            guardar_estado();
        }
        //Hacemos el cambio respecto al mundo
        else {
            printf("Escalado respecto al mundo\n");
            glMatrixMode(GL_MODELVIEW);
            glLoadIdentity();
            glScalef(a, b, c);
            glMatrixMode(GL_MODELVIEW);
            glMultMatrixd(_selected_object->matrix->matrix);
            guardar_estado();
        }
    }
    else {
        printf("ERROR: No hay ningun objeto cargado!\n");
    }
}

void rotate(float angle,float a, float b, float c)
{
    //Hacemos el cambio respecto al objeto
    if(_selected_object!=NULL) {
        if(modo2==0){
            printf("Rotacion respecto al objeto\n");
            glMatrixMode(GL_MODELVIEW);
            glLoadMatrixd(_selected_object->matrix->matrix);
            glRotatef(angle,a, b, c);
            guardar_estado();
        }
    //Hacemos el cambio respecto al mundo
    else {
            printf("Rotacion respecto al mundo\n");
            glMatrixMode(GL_MODELVIEW);
            glLoadIdentity();
            glRotatef(angle,a, b, c);
            glMatrixMode(GL_MODELVIEW);
            glMultMatrixd(_selected_object->matrix->matrix);
            guardar_estado();
        }
    }
    else {
        printf("ERROR: No hay ningun objeto cargado!\n");
    }
}
```

Para **deshacer un cambio** vamos a crear una lista auxiliar de matrices, *borrar*, que vamos a igualar a la lista del objeto seleccionado. Tras esto vamos a hacer que el objeto seleccionado apunte a la siguiente matriz de la lista y luego borraremos la matriz auxiliar creada para que no ocupe espacio de memoria de forma innecesaria.
Cabe destacar que no se realizarán cambios si no hay un objeto seleccionado o si hemos vuelto a la matriz de estado inicial; sabremos que es la matriz de estado inicial porque su puntero apuntará a un '0' ya que así hemos decidido que se inicialicen los objetos anteriormente.

```
void deshacer() {
    if(_selected_object!=NULL) {
    if(_selected_object->matrix->next!=0) {
      glMatrixMode(GL_MODELVIEW);
      matrix_l *borrar = _selected_object->matrix;
      _selected_object->matrix=_selected_object->matrix->next;
      free(borrar);
      glLoadMatrixd(_selected_object->matrix->matrix);
    }
    else {
      printf("ERROR: El objeto no tiene mas estados anteriores\n");
    }
  }
    else {
        printf("ERROR: No hay ningun objeto cargado!\n");
    }
}
```