eman ta zabal zazu

Universidad del País Vasco
Euskal Herriko Unibertsitatea

Control de camara y iluminación en OpenGL

Ignacio Belzunegui y Aitor Domec

December 23, 2018

1 Objetivos de nuestra aplicación

Nuestra aplicación tiene que ser capaz de controlar una camara y la iluminación. Tienen que existir varias cámaras, debemos de poder ver lo que un objeto, entrar en modo análisis/vuelo, modificar el volumen de vision, trasladar/rotar la cámara y cambiar el tipo de proyección entre modo paralelo o perspectiva.
Hemos dividido este trabajo en dos partes, camara y iluminacion (cada una tendrá sus propios sub-apartados).

1. **Camara.**
   (a) **Camara / Estructura de la camara.**
      i. **Estructura.**
      ii. **Cambio de camara.**
      iii. **Camara del objeto.**
   (b) **Modificaciones sobre la camara.**
      i. **Transformaciones.**
         A. **Cambio volumen de vision.**
         B. **Rotacion.**

C. **Traslacion.**

ii. **Modo analisis / Modo vuelo.**

iii. **Cambio de tipo de proyeccion.**

2. **Iluminación.**

   (a) **Prueba.**

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 y librerias de lglu, el comando para compilar es el siguiente:
gcc main.c display.c io.c load_obj_joseba.c -lGL -lGLU -lglut -lm
Este comando generará el ejecutable **a.out**.
Para ejecutar el ejecut
./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 donde realizará las transformaciones pertinentes.
Primero se deberá de cargar un objeto, de lo contrario no nos permitirá realizar ninguna operación. Para cargar un objeto deberá de escribir el path al objeto, dándole 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 todos los comandos, de forma predefinida los cambios se realizarán respecto al objeto y comenzara con el modo de rotación activado, pero esto podrá ser cambiado a voluntad en cualquier momento. Si deseamos utilizar la camara podremos acceder a sus comandos si pulsamos la tecla 'k', del mismo modo si deaseamos activar/desactivar la iluminaciónn deberemos de pulsar la tecla 'f9'.
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 u las teclas 'PageUp' o 'PageDown'.

3 Cambios realizados

1. **Camara.**

   (a) **Estructura.** Nuestra camara va a ser una pila doble que contendrá, una lista de matrices, un identificador y varios parametros que nos serviran para modificar el volumen de visión de la misma. Cada cámara apunta a la camara anterior y a la siguiente, creando asi un circuito cerrado.

```
typedef struct camara_l{
    int numero;                 /*Identificador de la camara*/
    GLdouble left;              /**/
    GLdouble right;             /**/
    GLdouble bottom;            /**/
    GLdouble top;               /**/
    GLdouble near;              /*Desde como de cerca vemos*/
    GLdouble far;               /*Como de lejos llegamos a ver*/

    GLdouble ortho_x_min;       /* Parametros usados*/
    GLdouble ortho_x_max;       /* para el modo     */
    GLdouble ortho_y_min;       /* de proyeccion    */
    GLdouble ortho_y_max;       /* ortogonal        */

    struct matrix_l *matrix;    /*Lista de matrices*/
```

```
    struct camara_l *next;          /*Siguiente camara */
    struct camara_l *previous;      /*Camara anterior */
} camara_l;
```

(b) **Cambio de camara.** Existen 4 camaras creadas en nuestro proyecto, si activamos el modo camara (pulsando la tecla 'k') y pulsamos la 'c' cambiaremos de la camara actual a la siguiente en la lista, se nos indicará por terminal en cual nos encontramos.
Cabe destacar que la camara inicial es simplemente la de identidad y que comenzaremos viendo todo desde un modo de perspectica ortogonal, es decir, en paralelo.
Para conseguir el efecto de cambiar de cámara, simplemente guardaremos la cámara actual en 'camara_auxiliar' y haremos que la cámara principal ('camaraG') apunte a la siguiente en la lista.

```
case 'c'://Cambiamos de camara
  if(camaraG==_selected_object->camara){
    camaraG=camara_auxiliar;
  }
  else{
    camara_auxiliar=camaraG;
    camaraG=camaraG->next;
  }
    printf("Usted acaba de cambiar a
    la camara: %i\n",camaraG->numero);
break;
```

(c) **Camara del objeto.** Si pulsamos la tecla 'C' la camara actual se trasladará al objeto, viendo asi lo que el objeto ve. Una vez activado el modo de camara del objeto no seremos capaces de modificar la camara, si queremos cambiar nuestro punto de vista deberemos de transformar el objeto selecionado.
'modog' es una variable que dice si están activadas las transformaciones en los objetos (modog=0), camara (modog=1) o iluminación (modog=2). 'modoCamara' simplemente dice si la cámara se ha colocado encima del objeto o no (Activado modoCamara=1, desactivado modoCamara=0), ya que si está activado no podremos modificar la cámara.

```
case 'C':
  if(modog==1) {
    if(modoCamara==0){
        modoCamara=1;
        printf("Usted acaba de cambiar a la camara del objeto\n");
      }
      else {
        modoCamara=0;
        printf("Usted acaba de desactivar la camara del objeto\n");
      }
  }
```

```
  else {
    printf("No esta activado el modo de camara\n");
  }
break;
```

Una vez activado el modoCamara deberemos de tener en cuenta que cada vez que modifiquen el objeto deberemos de modificar también la cámara. Para ello hemos creado una función llamada 'void camera_update()' que, una vez activado el modoCamara, se asegurará de actualizar la matriz de la cámara cada vez que se realize una transformación en el objeto. Cada vez que llamamos a 'camera_update' realizaremos un gluLookAt con los siguientes valores: Eye=Posición del objeto, Center=Posición del objeto-Vector Z del objeto, Up= Vector Y del objeto

```
void camera_update() {
  if(modoCamara==1) {
    glLoadIdentity();
    gluLookAt(_selected_object->matrix->matrix[12],
    _selected_object->matrix->matrix[13],
    _selected_object->matrix->matrix[14],
    _selected_object->matrix->matrix[12]-_selected_object->matrix->matrix[8],
    _selected_object->matrix->matrix[13]-_selected_object->matrix->matrix[9],
    _selected_object->matrix->matrix[14]-_selected_object->matrix->matrix[10],
    _selected_object->matrix->matrix[4],
    _selected_object->matrix->matrix[5],
    _selected_object->matrix->matrix[6]);
    glGetDoublev(GL_MODELVIEW_MATRIX,camaraG->matrix->matrix);

  }
}
```

Esta función es llamada cada vez que pulsamos una tecla, por lo que va a aparecer al final de tanto la función de teclado normal como la especial (encargada de leer las flechas).

2. **Modificaciones sobre la cámara.**

   (a) **Transformaciones.**

      i. **Cambio volumen de visión.** Si la cámara está activada (tecla 'k') y pulsamos la 't/T' vamos a activar el cambio en volumen de visión.
      Dentro de cada cámara guardamos los valores left, right, top, bottom, near y far. Los dos últimos son los encargados de decir como de lejos vemos (far) y desde como de cerca empezamos a ver (near). Los otros cuatro valores son los encargados de especificar la 'ventana' desde la que vemos en el

modo perspectiva, si los alteramos veremos el escenario deformado. Es decir, nos muestran las "dimensiones" de nuestro volumen de visión.
Visualizaremos la implementación del cambio de volumen de visión mediante el código implementado en el botón UP.

```
        case GLUT_KEY_PAGE_UP:
          if(modoV == 0){//Modo vuelo
                if(modo==1) {
          (...)
                }
                if(modo==2) {
            (...)
                }
                if(modo==3) {
                  if(modog==0){
          (...)
                    }
                    if (modog==1)
                     {
                    camaraG->near=camaraG->near*1.1;
                    camaraG->far=camaraG->far*1.1;
                     }
                }
             }
             else {//Modo analisis
              if(modo==2){
                (...)
              }
              if(modo==1){
                (...)
              }
             }
                (...)
        break;
```

Los parámetros near y far aumentan en un 10% cuando le damos al botón UP y disminuyen un 10% cuando es pulsado el botón DOWN, y los parámetros right y left aumentan un 10% cuando RIGHT es pulsado y disminuyen un 10% cuando LEFT es pulsado.

ii. **Modo Análisis/Vuelo.** El modo vuelo nos permite "volar" por el escenario; la cámara podrá ser movida libremente por el mundo dependiendo de lo que queramos hacer (como trasladarnos o rotar).
El modo análisis nos permite "analizar" (como indica su nombre) un objeto, situándolo en el centro de la cámara y permitiéndonos verlo desde cualquier ángulo a cualquier distancia. La variable modoV define el modo en el que nos encontramos; modoV = 0 (modo vuelo) y modoV = 1 modo ANÁLISIS. Cambiaremos entre modo vuelo y modo análisis pulsando la tecla G/g. Por defecto está activado el modo

vuelo.

iii. **Rotación.**
La cámara puede rotar sobre si misma en el MODO VUELO (modoV = 0) y alrededor de un objeto concreto en el MODO ANÁLISIS (modoV=1).

```
void rotate(float angle,float a, float b, float c)
{
    //Hacemos el cambio respecto al objeto
    if(_selected_object!=NULL) {
        if(modog==0) {//Sobre el objeto
            if(modo2==0){
        (...)
            }
        //Hacemos el cambio respecto al mundo
            else {
        (...)
            }
        }
        else {//Cambio respecto a la camara
            if (modoV==0)//Modo vuelo
            {
              glMatrixMode(GL_MODELVIEW);
              glLoadIdentity();
              glRotatef(angle,a, b, c);
              glMatrixMode(GL_MODELVIEW);
              glMultMatrixd(camaraG->matrix->matrix);
              guardar_estado();
              representar_matriz();
            }
            else{//modo analisis

              matrix_l *matrizaux;
            matrizaux = (matrix_l *)
            malloc(sizeof (matrix_l));

            //Matriz inversa de la MCSR
            matrizaux->matrix[0] =
            camaraG->matrix->matrix[0];
        matrizaux->matrix[1] =
        camaraG->matrix->matrix[4];
        matrizaux->matrix[2] =
        camaraG->matrix->matrix[8];
        matrizaux->matrix[3] = 0.0;


        matrizaux->matrix[4] =
        camaraG->matrix->matrix[1];
        matrizaux->matrix[5] =
        camaraG->matrix->matrix[5];
        matrizaux->matrix[6] =
        camaraG->matrix->matrix[9];
        matrizaux->matrix[7] = 0.0;


        matrizaux->matrix[8] =
        camaraG->matrix->matrix[2];
        matrizaux->matrix[9] =
        camaraG->matrix->matrix[6];
        matrizaux->matrix[10] =
```

```
        camaraG->matrix->matrix[10];
        matrizaux->matrix[11] = 0.0;
        matrizaux->matrix[12] =
        camaraG->posicionC.x;
            matrizaux->matrix[13] =
            camaraG->posicionC.y;
            matrizaux->matrix[14] =
            camaraG->posicionC.z;
        matrizaux->matrix[15] = 1.0;


            glLoadIdentity();
            glMatrixMode(GL_MODELVIEW);
            glTranslatef(_selected_object-
            >matrix->matrix[12],
            _selected_object->matrix->matrix[13],
            _selected_object->matrix->matrix[14]);
          glRotatef(angle,a,b,c);
            glTranslatef(-_selected_object->
            matrix->matrix[12],
            -_selected_object->matrix->matrix[13],
            -_selected_object->matrix->matrix[14]);
            glMultMatrixd(matrizaux->matrix);
            glMatrixMode(GL_MODELVIEW);
            glGetDoublev
            (GL_MODELVIEW_MATRIX,matrizaux
            ->matrix);
            camaraG->posicionC.x =
            matrizaux->matrix[12];
            camaraG->posicionC.y =
             matrizaux->matrix[13];
            camaraG->posicionC.z =
            matrizaux->matrix[14];

        //Generamos la Mcsr a partir de la inversa
        camaraG->matrix->matrix[0] =
        matrizaux->matrix[0];
        camaraG->matrix->matrix[1] =
        matrizaux->matrix[4];
        camaraG->matrix->matrix[2] =
        matrizaux->matrix[8];
        camaraG->matrix->matrix[3] = 0.0;


        camaraG->matrix->matrix[4] =
        matrizaux->matrix[1];
        camaraG->matrix->matrix[5] =
        matrizaux->matrix[5];
        camaraG->matrix->matrix[6] =
        matrizaux->matrix[9];
        camaraG->matrix->matrix[7] = 0.0;


        camaraG->matrix->matrix[8] =
        matrizaux->matrix[2];
        camaraG->matrix->matrix[9] =
        matrizaux->matrix[6];
        camaraG->matrix->matrix[10] =
        matrizaux->matrix[10];
        camaraG->matrix->matrix[11] = 0.0;
```

```
        camaraG->matrix->matrix[12] =
        -((matrizaux->matrix[0]*matrizaux-
        >matrix[12])+
        (matrizaux->matrix[1]*matrizaux-
        >matrix[13]) + (matrizaux-
        >matrix[2]*matrizaux->matrix[14]));


        camaraG->matrix->matrix[13] =
        -((matrizaux->matrix[4]*matrizaux
        ->matrix[12])+(matrizaux-
        >matrix[5]*matrizaux->matrix[13]) +
        (matrizaux->matrix[6]*matrizaux-
        >matrix[14]));

        camaraG->matrix->matrix[14] = -
        ((matrizaux->matrix[8]*matrizaux-
        >matrix[12])+(matrizaux-
        >matrix[9]*matrizaux->matrix[13]) +
        (matrizaux->matrix[10]*matrizaux-
        >matrix[14]));
        camaraG->matrix->matrix[15] = 1.0;

        glLoadMatrixd(camaraG->matrix->matrix);
            guardar_estado();
            representar_matriz();
            }
        }
    }
    else {
      (...)
    }
}
```

Hemos implementado la rotación en modo vuelo y modo análisis en la función rotate. Si el modo cámara está activado y el modoVuelo está activado, entonces la cámara rotará en el ángulo definido por angle en el vector definido por a,b,c.
Si nos encontramos en modo vuelo (modoV = 0), se realizará la siguiente operación:

```
            glMatrixMode(GL_MODELVIEW);
            glLoadIdentity();
            glRotatef(angle,a, b, c);
            glMatrixMode(GL_MODELVIEW);
            glMultMatrixd(camaraG->matrix->matrix);
            guardar_estado();
            representar_matriz();
```

Primero cargamos la matriz de identidad, y de esta forma "limpiamos" la matriz activa de openGL. Despues, aplicamos la matriz de rotación sobre esta matriz de identidad (la cámara rotará en un ángulo angle, en el vector definido por a, b y c). Posteriormente, multiplicamos a la matriz activa de openGL la matriz de la cámara (también llamada la matriz de cambio de sistema de referencia). Las dos siguientes líneas de código

se encargan de guardar en la pila de matrices de la cámara el estado actual de la cámara y de representar la matriz en el terminal.

Si nos encontramos en modo análisis, el procedimiento para rotar cambia bastante:

```
            matrix_l *matrizaux;
        matrizaux = (matrix_l *) malloc(sizeof (matrix_l));


        //Matriz inversa de la MCSR
        matrizaux->matrix[0] = camaraG->matrix->matrix[0];
    matrizaux->matrix[1] = camaraG->matrix->matrix[4];
    matrizaux->matrix[2] = camaraG->matrix->matrix[8];
    matrizaux->matrix[3] = 0.0;


    matrizaux->matrix[4] = camaraG->matrix->matrix[1];
    matrizaux->matrix[5] = camaraG->matrix->matrix[5];
    matrizaux->matrix[6] = camaraG->matrix->matrix[9];
    matrizaux->matrix[7] = 0.0;


    matrizaux->matrix[8] = camaraG->matrix->matrix[2];
    matrizaux->matrix[9] = camaraG->matrix->matrix[6];
    matrizaux->matrix[10] = camaraG->matrix->matrix[10];
    matrizaux->matrix[11] = 0.0;
    matrizaux->matrix[12] = camaraG->posicionC.x;
        matrizaux->matrix[13] = camaraG->posicionC.y;
        matrizaux->matrix[14] = camaraG->posicionC.z;
    matrizaux->matrix[15] = 1.0;


        glLoadIdentity();
        glMatrixMode(GL_MODELVIEW);
        glTranslatef(_selected_object->matrix->matrix[12],
        _selected_object->matrix->matrix[13],
        _selected_object->matrix->matrix[14]);
      glRotatef(angle,a,b,c);
        glTranslatef(-_selected_object->matrix->matrix[12],
        -_selected_object->matrix->matrix[13],
        -_selected_object->matrix->matrix[14]);
        glMultMatrixd(matrizaux->matrix);


        glMatrixMode(GL_MODELVIEW);
        glGetDoublev(GL_MODELVIEW_MATRIX,matrizaux->matrix);
        camaraG->posicionC.x = matrizaux->matrix[12];
        camaraG->posicionC.y = matrizaux->matrix[13];
        camaraG->posicionC.z = matrizaux->matrix[14];

    //Generamos la Mcsr a partir de la inversa
    camaraG->matrix->matrix[0] = matrizaux->matrix[0];
    camaraG->matrix->matrix[1] = matrizaux->matrix[4];
    camaraG->matrix->matrix[2] = matrizaux->matrix[8];
    camaraG->matrix->matrix[3] = 0.0;
```

```
camaraG->matrix->matrix[4] = matrizaux->matrix[1];
camaraG->matrix->matrix[5] = matrizaux->matrix[5];
camaraG->matrix->matrix[6] = matrizaux->matrix[9];
camaraG->matrix->matrix[7] = 0.0;


camaraG->matrix->matrix[8] = matrizaux->matrix[2];
camaraG->matrix->matrix[9] = matrizaux->matrix[6];
camaraG->matrix->matrix[10] = matrizaux->matrix[10];
camaraG->matrix->matrix[11] = 0.0;


camaraG->matrix->matrix[12] = -((matrizaux->matrix[0]*matrizaux->matrix[12
camaraG->matrix->matrix[13] = -((matrizaux->matrix[4]*matrizaux->matrix[12
camaraG->matrix->matrix[14] = -((matrizaux->matrix[8]*matrizaux->matrix[12
camaraG->matrix->matrix[15] = 1.0;

glLoadMatrixd(camaraG->matrix->matrix);
    guardar_estado();
    representar_matriz();
```

Primero definimos una matriz auxiliar llamada matrizaux de tipo matrix_l y le asignamos un espacio de memoria. Esta matriz será la encargada de modificar la posición a la matriz de la cámara. Antes de continuar con la explicación, es necesario saber qué datos contiene la matriz de la cámara:

$$\mathbf{Mcsr} = \begin{pmatrix} Xc & . & . & -E * Xc \\ Yc & . & . & -E * Yc \\ Zc & . & . & -E * Zc \\ 0 & . & . & 1 \end{pmatrix}$$

iv. **Traslación.**

(b) **Cambio de tipo de proyección.**

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_STEPn_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 y 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.

   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, 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 transformacion, ya que de lo contrario nos daría un error de segmentación.

```
void traslate(float a, float b, float c)
{
    if(_selected_object!=NULL) {
        if(modog==0) {//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();
                representar_matriz();
            }
            else {//Hacemos el cambio respecto al mundo
                printf("Traslacion respecto al mundo\n");
                glMatrixMode(GL_MODELVIEW);
                glLoadIdentity();
                glTranslatef(a, b, c);
                glMatrixMode(GL_MODELVIEW);
                glMultMatrixd(_selected_object->matrix->matrix);
                guardar_estado();
                representar_matriz();
                }
        }
        //Cambio respecto a la camara
        else {
          if (modoV==0)//Modo vuelo
            {
              glMatrixMode(GL_MODELVIEW);
              glLoadIdentity();
              glTranslatef(a,b,c);
              printf("Posicion de la camara X %f Y %f Z %f\n",camaraG->posicionC.x,camaraG-
              camaraG->posicionC.x += a;
              camaraG->posicionC.y += b;
              camaraG->posicionC.z += c;
              printf("Posicion NUEVA de la camara X %f Y %f Z %f\n",camaraG->posicionC.x,ca
              glMatrixMode(GL_MODELVIEW);
              glMultMatrixd(camaraG->matrix->matrix);
              guardar_estado();
              representar_matriz();
          }
          else {//Modo analisis, movernos hacia adelante o hacia atras
            printf("posicionC.z:%f Z objeto:%f\n",camaraG->posicionC.z,_selected_object->ma
            double distanciaZ=absoluto(-camaraG->posicionC.z - _selected_object->matrix->ma
            printf("DistanciaZ:%f\n",distanciaZ);
            if(distanciaZ>c) {
              glLoadIdentity();
                glTranslatef(a,b,c);
                camaraG->posicionC.z += c;
                glMatrixMode(GL_MODELVIEW);
                glMultMatrixd(camaraG->matrix->matrix);
                guardar_estado();
                printf("CamaraG->posicionC.z:%f\n",camaraG->posicionC.z);
                representar_matriz();
            }
            else{
              printf("No nos podemos mover mas adelante\n");
              printf("DistanciaZ:%f <= Movimiento:%f\n",distanciaZ,c);
            }
          }
        }
        if (iluminacion!=0)
        {
          printf("Trasladamos la bombilla\n");
          pbombilla[0]+=a;
          pbombilla[1]+=b;
          pbombilla[2]+=c;
          glLightfv(GL_LIGHT1, GL_POSITION, pbombilla);
          printf("Posicion bombilla: X:%f Y:%f Z:%f\n",pbombilla[0],pbombilla[1],pbombilla[
        }
    }
    else {
```

```
        printf("ERROR: No hay ningun objeto cargado!\n");
    }
}

void scale(float a, float b, float c)
{
    if(_selected_object!=NULL) {
        if(modog==0) {
            //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();
                representar_matriz();
            }
            //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();
                representar_matriz();
            }
        }
        else {//Cambio respecto a la camara
            glMatrixMode(GL_MODELVIEW);
            glLoadIdentity();
            glScalef(a, b, c);
            glMatrixMode(GL_MODELVIEW);
            glMultMatrixd(camaraG->matrix->matrix);
            guardar_estado();
            representar_matriz();
        }
    }
    else {
        printf("ERROR: No hay ningun objeto cargado!\n");
    }
}

/*
struct matrix_l generarMCR(struct matrix_l *m){

  matrix_l *matrizaux;
  matrizaux = (matrix_l *) malloc(sizeof (matrix_l));

  matrizaux->matrix[0] = m->matrix[0];
  matrizaux->matrix[1] = m->matrix[4];
  matrizaux->matrix[2] = m->matrix[8];
  matrizaux->matrix[3] = 0.0;


  matrizaux->matrix[4] = m->matrix[1];
  matrizaux->matrix[5] = m->matrix[5];
  matrizaux->matrix[6] = m->matrix[9];
  matrizaux->matrix[7] = 0.0;
```

```
  matrizaux->matrix[8] = m->matrix[2];
  matrizaux->matrix[9] = m->matrix[6];
  matrizaux->matrix[10] = m->matrix[10];
  matrizaux->matrix[11] = 0.0;


  matrizaux->matrix[12] = (m->matrix[0]*m->matrix[12])+(m->matrix[1]*m->matrix[13]) + (m->m
  matrizaux->matrix[13] = (m->matrix[4]*m->matrix[12])+(m->matrix[5]*m->matrix[13]) + (m->m
  matrizaux->matrix[14] = (m->matrix[8]*m->matrix[12])+(m->matrix[9]*m->matrix[13]) + (m->m
  matrizaux->matrix[15] = 1.0;

  return matrizaux;

}
*/


void rotate(float angle,float a, float b, float c)
{
    //Hacemos el cambio respecto al objeto
    if(_selected_object!=NULL) {
        if(modog==0) {//Sobre el objeto
            if(modo2==0){
                printf("Rotacion respecto al objeto\n");
                glMatrixMode(GL_MODELVIEW);
                glLoadMatrixd(_selected_object->matrix->matrix);
                glRotatef(angle,a, b, c);
                guardar_estado();
                representar_matriz();
            }
        //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();
                representar_matriz();
            }
        }
        else {//Cambio respecto a la camara
            if (modoV==0)//Modo vuelo
            {
              glMatrixMode(GL_MODELVIEW);
              glLoadIdentity();
              glRotatef(angle,a, b, c);
              glMatrixMode(GL_MODELVIEW);
              glMultMatrixd(camaraG->matrix->matrix);
              guardar_estado();
              representar_matriz();
            }
            else{//modo analisis
              printf("ROTACION RESPECTO AL OBJETO???\n");
              matrix_l *matrizaux;
            matrizaux = (matrix_l *) malloc(sizeof (matrix_l));

            printf("Posicion de la camara\n");
            printf("Posicion de la camara X %f Y %f Z %f\n",camaraG->posicionC.x,camaraG->p
            //Matriz inversa de la MCSR
            matrizaux->matrix[0] = camaraG->matrix->matrix[0];
```

```
        matrizaux->matrix[1] = camaraG->matrix->matrix[4];
        matrizaux->matrix[2] = camaraG->matrix->matrix[8];
        matrizaux->matrix[3] = 0.0;


        matrizaux->matrix[4] = camaraG->matrix->matrix[1];
        matrizaux->matrix[5] = camaraG->matrix->matrix[5];
        matrizaux->matrix[6] = camaraG->matrix->matrix[9];
        matrizaux->matrix[7] = 0.0;


        matrizaux->matrix[8] = camaraG->matrix->matrix[2];
        matrizaux->matrix[9] = camaraG->matrix->matrix[6];
        matrizaux->matrix[10] = camaraG->matrix->matrix[10];
        matrizaux->matrix[11] = 0.0;
        matrizaux->matrix[12] = camaraG->posicionC.x;
            matrizaux->matrix[13] = camaraG->posicionC.y;
            matrizaux->matrix[14] = camaraG->posicionC.z;
        matrizaux->matrix[15] = 1.0;


            glLoadIdentity();
            glMatrixMode(GL_MODELVIEW);
            glTranslatef(_selected_object->matrix->matrix[12],
            _selected_object->matrix->matrix[13],
            _selected_object->matrix->matrix[14]);
          glRotatef(angle,a,b,c);
            glTranslatef(-_selected_object->matrix->matrix[12],
            -_selected_object->matrix->matrix[13],
            -_selected_object->matrix->matrix[14]);
            glMultMatrixd(matrizaux->matrix);




            glMatrixMode(GL_MODELVIEW);
            glGetDoublev(GL_MODELVIEW_MATRIX,matrizaux->matrix);
            camaraG->posicionC.x = matrizaux->matrix[12];
            camaraG->posicionC.y = matrizaux->matrix[13];
            camaraG->posicionC.z = matrizaux->matrix[14];

        //Generamos la Mcsr a partir de la inversa
        camaraG->matrix->matrix[0] = matrizaux->matrix[0];
        camaraG->matrix->matrix[1] = matrizaux->matrix[4];
        camaraG->matrix->matrix[2] = matrizaux->matrix[8];
        camaraG->matrix->matrix[3] = 0.0;


        camaraG->matrix->matrix[4] = matrizaux->matrix[1];
        camaraG->matrix->matrix[5] = matrizaux->matrix[5];
        camaraG->matrix->matrix[6] = matrizaux->matrix[9];
        camaraG->matrix->matrix[7] = 0.0;


        camaraG->matrix->matrix[8] = matrizaux->matrix[2];
        camaraG->matrix->matrix[9] = matrizaux->matrix[6];
        camaraG->matrix->matrix[10] = matrizaux->matrix[10];
        camaraG->matrix->matrix[11] = 0.0;


        camaraG->matrix->matrix[12] = -((matrizaux->matrix[0]*matrizaux->matrix[12])+(matri
```

```
        camaraG->matrix->matrix[13] = -((matrizaux->matrix[4]*matrizaux->matrix[12])+(matri
        camaraG->matrix->matrix[14] = -((matrizaux->matrix[8]*matrizaux->matrix[12])+(matri
        camaraG->matrix->matrix[15] = 1.0;

        glLoadMatrixd(camaraG->matrix->matrix);
            guardar_estado();
            representar_matriz();
            }
        }
    }
    else {
        printf("ERROR: No hay ningun objeto cargado!\n");
    }
}
```

Para guardar los cambios realizados en el objeto usaremos la funcion **'guardar_estado()'**. Simplemente 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.

```
void guardar_estado()
{
    matrix_l *matrizaux;
    matrizaux = (matrix_l *) malloc(sizeof (matrix_l));
    matrizaux->next=_selected_object->matrix;
    glGetDoublev(GL_MODELVIEW_MATRIX,matrizaux->matrix);
    _selected_object->matrix=matrizaux;
}
```

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 asi 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");
    }
}
```