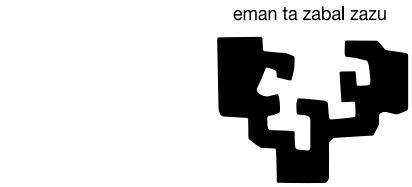


Universidad del País Vasco

Euskal Herriko Unibertsitatea

Gráficos por Computador

Daniel Ruskov

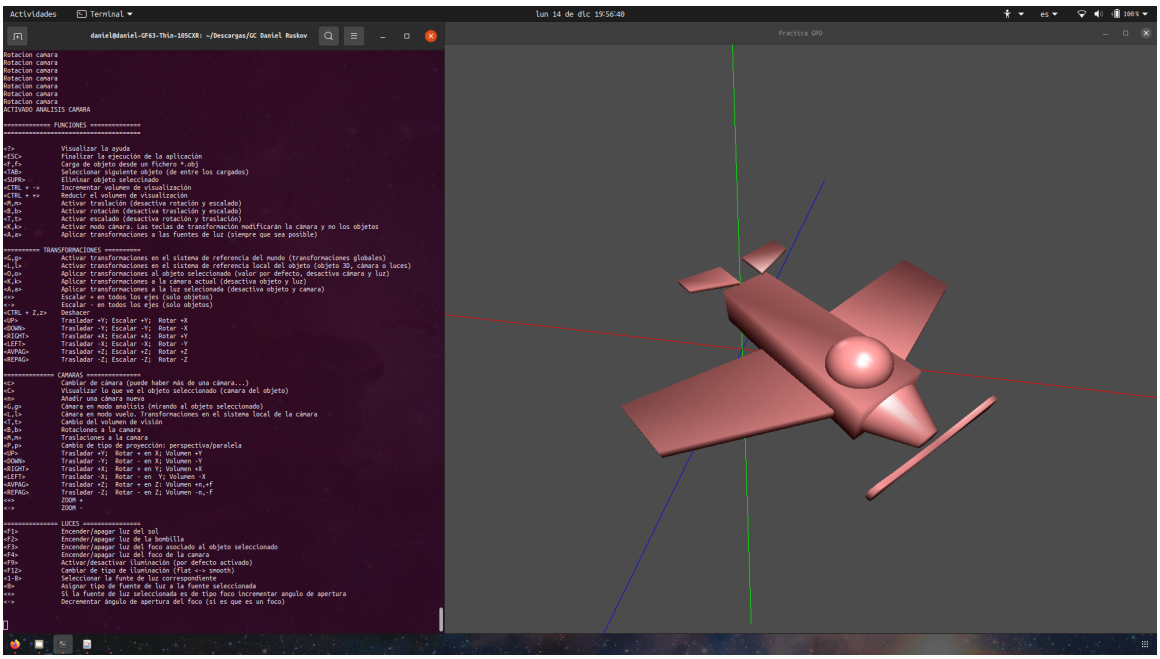
Diciembre 2020

Abstract

Gráficos por computador. Práctica 2 - motor gráfico básico implementado en lenguaje de programación C con la ayuda de OpenGL. Fases de transformaciones geométricas, cámaras e iluminación.

1 Introducción y objetivos

El objetivo principal es implementar una aplicación capaz de simular un espacio 3D, en el cual se pueden cargar objetos 3D, y visualizarlos mediante camaras en el espacio, controlando las posiciones y/o transformaciones de objetos, cámaras, luces, etc.



2 Estructuras

Las estructuras utilizadas se encuentran en el fichero definitions.h:

```
/** STRUCTURES **/

/*****************************
 * Structure to store the    *
 * coordinates of 3D points  *
 *****************************/
typedef struct {
    GLdouble x, y, z;
} point3;

/*****************************
 * Structure to store the    *
 * coordinates of 3D vectors *
 *****************************/
typedef struct {
    GLdouble x, y, z;
} vector3;

/*****************************
 * Structure to store the    *
 * colors in RGB mode        *
 *****************************/
typedef struct {
    GLfloat r, g, b;
} color3;

/*****************************
 * Structure to store        *
 * objects' vertices         *
 *****************************/
```

```
typedef struct {
    point3 coord;               /* coordinates,x, y, z */
    GLint num_faces;            /* number of faces that share this vertex */
    vector3 vnormal;            /* vector normal del vertice */
} vertex;

/***************************
 * Structure to store      *
 * objects' faces or       *
 * polygons                *
 ***************************/
typedef struct {
    GLint num_vertices;         /* number of vertices in the face */
    GLint *vertex_table;        /* table with the index of each vertex */
    vector3 vnormal;            /* vector normal de la cara */
} face;

/***************************
 * Structure to store a    *
 * pile of 3D objects      *
 ***************************/
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 */
    struct object3d *next;      /* next element in the pile of objects */
    struct matrices *primptr;   /* PRIMER ELEMENTO DE LA LISTA DE MATRICES  */
    struct camera *obj_cam;     /* camara propia del objeto */
    struct material *mater;     /* material del objeto */
};

typedef struct object3d object3d;

typedef struct camera {
    point3 coordenadas;         /* coordenadas de la camara */
    point3 pto_at;              /* coordenadas del punto de atencion */
    vector3 vup;                /* coordenadas del vector de verticalidad */
    GLdouble left;              /* parametro left para hallar el volumen de vision */
    GLdouble right;             /* parametro right para hallar el volumen de vision */
    GLdouble bottom;            /* parametro bottom para hallar el volumen de vision */
    GLdouble top;               /* parametro top para hallar el volumen de vision */
    GLdouble near;              /* parametro near para hallar el volumen de vision */
    GLdouble far;               /* parametro far para hallar el volumen de vision */
    int perspectiva;            /* modo si = 1 en perspectiva, si = 0 en ortogonal */
    int tiene_obj;              /* para determinar si la camara tiene objeto asociado */
    struct matrices *mcsr;      /* matriz de cambio de sistema de referencia */
    struct matrices *inversa;   /* matriz inversa a la de CSR */
    struct camera *next;        /* siguiente camara */
    struct camera *prev;        /* camara anterior */
} camera;

typedef struct matrices{
    GLfloat matriz[16];
    struct matrices *sigptr;
} matrices;

typedef struct light{
    GLfloat position[4];        /* posicion de la luz */
    GLfloat dir[3];             /* direccion a la que apunta */
    int encendida;              /* indica si la luz esta encendida o no */
    struct matrices *m_luz;     /* matriz de la luz */
    GLfloat ambient[4];         /* valores de ambient de la luz */
    GLfloat diffuse[4];         /* valores de diffuse de la luz */
    GLfloat specular[4];        /* valores de specular de la luz */
    GLfloat amplitud;           /* valor de la amplitud (el angulo) */
} light;

typedef struct material{
    GLfloat ambient[4];
```

```
    GLfloat diffuse[4];
    GLfloat specular[4];
    GLfloat shininess;
} material;
```

Hasta la estructura de "object3d" todas las estructuras previas son necesarias para el mismo, las de punto, vértice, etc. Por otra parte están las de la cámara, matrices, luces y materiales.

Primero, la estructura de matrices simplemente consiste en una matriz y un puntero hacia una estructura del mismo tipo, para poder hacer una lista de matrices y así implementar la opción de deshacer movimientos de los objetos.

Segundo, la de cámara, que consiste en los parámetros que necesitan las cámaras para su creación y uso, dentro podemos ver la posición, el punto de atención, el vector de verticalidad, los parámetros para el volumen de visión (near, far, etc.), si la cámara emplea vista con perspectiva paralela u ortogonal, si tiene un objeto asociado, una lista de matrices para la matriz de cambio de sistema de referencia, otra para la matriz inversa a la del cambio de sistema de referencia (sistema de referencia local de la cámara), y dos punteros que apuntan a la siguiente y a la cámara previa.

Tercero, la estructura de la luz, la cual tiene su posición, dirección en la que apunta, si está o no encendida, los valores de ambiente, difusa y especular y la amplitud del foco (interesa sobretodo si es para usar focos, y no tanto para el sol del mundo).

Por último, una estructura sencilla para los materiales, aunque en la aplicación solo se usa parámetros para un solo tipo de material (ruby). Guarda los valores de ambiente, difusa, especular y brillo.

3 Funciones a destacar

Hay varias funciones importantes en la aplicación, sobretodo las que permiten las transformaciones de objetos, cámaras y luces. A continuación se explica brevemente el funcionamiento de estas. El control de luces, de que objeto seleccionados, etc están explicadas en el anexo 1 - manual de usuario.

En io.c - cálculo de las matrices de cambio de sistema de referencia y de sistema de referencia local de las cámaras:

```
//CALCULA LA INVERSA DE MCSR
void matriz_inversa_camara(camera *c)
{
    matrices *minv;
    minv = malloc(sizeof(matrices));
    minv->matriz[0] = c->mcsr->matriz[0];
    minv->matriz[1] = c->mcsr->matriz[4];
    minv->matriz[2] = c->mcsr->matriz[8];
    minv->matriz[3] = 0;
    minv->matriz[4] = c->mcsr->matriz[1];
    minv->matriz[5] = c->mcsr->matriz[5];
    minv->matriz[6] = c->mcsr->matriz[9];
    minv->matriz[7] = 0;
    minv->matriz[8] = c->mcsr->matriz[2];
    minv->matriz[9] = c->mcsr->matriz[6];
    minv->matriz[10] = c->mcsr->matriz[10];
    minv->matriz[11] = 0;
    minv->matriz[12] = c->coordenadas.x;
    minv->matriz[13] = c->coordenadas.y;
    minv->matriz[14] = c->coordenadas.z;
    minv->matriz[15] = 1;
    minv->sigptr = c->inversa;
    c->inversa = minv;
```

```
}

//CALCULA LA INVERSA DE LA INVERSA DE LA MCSR, ES DECIR, LA MCSR
void matriz_camara(camera *c)
{
    matrices *mmcsr;
    mmcsr = malloc(sizeof(matrices));
    mmcsr->matriz[0] = c->inversa->matriz[0];
    mmcsr->matriz[1] = c->inversa->matriz[4];
    mmcsr->matriz[2] = c->inversa->matriz[8];
    mmcsr->matriz[3] = 0;
    mmcsr->matriz[4] = c->inversa->matriz[1];
    mmcsr->matriz[5] = c->inversa->matriz[5];
    mmcsr->matriz[6] = c->inversa->matriz[9];
    mmcsr->matriz[7] = 0;
    mmcsr->matriz[8] = c->inversa->matriz[2];
    mmcsr->matriz[9] = c->inversa->matriz[6];
    mmcsr->matriz[10] = c->inversa->matriz[10];
    mmcsr->matriz[11] = 0;
    mmcsr->matriz[12] = -1 * (c->inversa->matriz[12]
                        * c->inversa->matriz[0] + c->inversa->matriz[13]
                        * c->inversa->matriz[1] + c->inversa->matriz[14]
                        * c->inversa->matriz[2]);
    mmcsr->matriz[13] = -1 * (c->inversa->matriz[12]
                        * c->inversa->matriz[4] + c->inversa->matriz[13]
                        * c->inversa->matriz[5] + c->inversa->matriz[14]
                        * c->inversa->matriz[6]);
    mmcsr->matriz[14] = -1 * (c->inversa->matriz[12]
                        * c->inversa->matriz[8] + c->inversa->matriz[13]
                        * c->inversa->matriz[9] + c->inversa->matriz[14]
                        * c->inversa->matriz[10]);
    mmcsr->matriz[15] = 1;
    mmcsr->sigptr = c->mcsr;
    c->mcsr = mmcsr;
}
```

Las dos se calculan una a partir de la otra, manteniendo siempre actualizadas debidamente la de cambio de sistema de referencia y la de sistema de referencia local. Al ser una la inversa de la otra y viceversa, lo único que hacemos aquí es simplificar el cálculo de la inversa de ambas, pero a fin de cuentas calculamos la inversa de una para obtener la otra.

En io.c - cambiar la matriz de cambio de sistema de referencia y actualizar su inversa a la hora de hacer el modo análisis:

```
//AL ACTIVAR MODO ANALISIS NUESTRO PUNTO DE ATENCION PASA A SER EL CENTRO DEL OBJETO
void camb_sist_referencia()
{
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    gluLookAt(_selected_object->primptr->matriz[12]
                + _selected_camera->inversa->matriz[12],
              _selected_object->primptr->matriz[13]
                + _selected_camera->inversa->matriz[13],
              _selected_object->primptr->matriz[14]
                + _selected_camera->inversa->matriz[14],
              _selected_object->primptr->matriz[12],
              _selected_object->primptr->matriz[13],
              _selected_object->primptr->matriz[14],
              _selected_camera->vup.x,
              _selected_camera->vup.y,
              _selected_camera->vup.z);
    glGetFloatv(GL_MODELVIEW_MATRIX, _selected_camera->mcsr->matriz);
    _selected_camera->coordenadas.x = _selected_object->primptr->matriz[12]
                                    + _selected_camera->inversa->matriz[12];
    _selected_camera->coordenadas.y = _selected_object->primptr->matriz[13]
                                    + _selected_camera->inversa->matriz[13];
    _selected_camera->coordenadas.z = _selected_object->primptr->matriz[14]
                                    + _selected_camera->inversa->matriz[14];
    matriz_inversa_camara(_selected_camera);
```

```
}
```

En io.c - transformaciones en luces:

```
//TRANSFORMACIONES DE LAS LUCES
void transformacionLuz(float x, float y, float z, int alpha)
{
    //sol
    if (_selected_light == 0)
    {
        if (rotac_activa == 1)
        {
            matrices *m = malloc(sizeof(matrices));
            glMatrixMode(GL_MODELVIEW);
            glLoadIdentity();
            glRotatef(alpha, -y, x, z);
            glMultMatrixf(luces[0].m_luz->matriz);
            glGetFloatv(GL_MODELVIEW_MATRIX, m->matriz);
            m->sigptr = luces[0].m_luz;
            luces[0].m_luz = m;
            luces[0].position[0] = luces[0].m_luz->matriz[12];
            luces[0].position[1] = luces[0].m_luz->matriz[13];
            luces[0].position[2] = luces[0].m_luz->matriz[14];
        }
    }
    //bombilla
    if (_selected_light == 1)
    {
        if (trasl_activa == 1)
        {
            if (local == 1)
            {
                matrices *m = malloc(sizeof(matrices));
                glMatrixMode(GL_MODELVIEW);
                glLoadMatrixf(luces[1].m_luz->matriz);
                glTranslatef(x, y, z);
                glGetFloatv(GL_MODELVIEW_MATRIX, m->matriz);
                m->sigptr = luces[1].m_luz;
                luces[1].m_luz = m;
            }
            else
            {
                matrices *m = malloc(sizeof(matrices));
                glMatrixMode(GL_MODELVIEW);
                glLoadIdentity();
                glTranslatef(x, y, z);
                glMultMatrixf(luces[1].m_luz->matriz);
                glGetFloatv(GL_MODELVIEW_MATRIX, m->matriz);
                m->sigptr = luces[1].m_luz;
                luces[1].m_luz = m;
            }
        }
        if (rotac_activa == 1)
        {
            matrices *m = malloc(sizeof(matrices));
            glMatrixMode(GL_MODELVIEW);
            glLoadIdentity();
            glRotatef(alpha, -y, x, z);
            glMultMatrixf(luces[1].m_luz->matriz);
            glGetFloatv(GL_MODELVIEW_MATRIX, m->matriz);
            m->sigptr = luces[1].m_luz;
            luces[1].m_luz = m;
        }
        luces[1].position[0] = luces[1].m_luz->matriz[12];
        luces[1].position[1] = luces[1].m_luz->matriz[13];
        luces[1].position[2] = luces[1].m_luz->matriz[14];
    }
    //foco obj o foco cam
    if (_selected_light == 2 || _selected_light == 3)
    {
        if (rotac_activa == 1)
        {
```

```
            matrices *m = malloc(sizeof(matrices));
            glMatrixMode(GL_MODELVIEW);
            glLoadMatrixf(luces[_selected_light].m_luz->matriz);
            glRotatef(alpha, -y, x, z);
            glGetFloatv(GL_MODELVIEW_MATRIX, m->matriz);
            m->sigptr = luces[_selected_light].m_luz;
            luces[_selected_light].m_luz = m;
        }
    }
}
```

Cuando las transformaciones son en el sistema de referencia global, primero se carga la matriz identidad en el modelview, luego se hace la transformación y por último se multiplica la matriz del objeto (en este caso es una luz) para aplicar las transformaciones globales. Si, en cambio, son en local, primero se carga la matriz del objeto (en este caso la luz) y después se hace la transformación.

En io.c - transformaciones de cámara en modo vuelo:

```
//TRANSLACIONES EN MODO VUELO
void translacionVuelo(float x, float y, float z)
{
    matrices *minv;
    minv = malloc(sizeof(matrices));
    glMatrixMode(GL_MODELVIEW);
    glLoadMatrixf(_selected_camera->inversa->matriz);
    glTranslatef(x, y, z);
    glGetFloatv(GL_MODELVIEW_MATRIX, minv->matriz);
    minv->sigptr = _selected_camera->inversa;
    _selected_camera->inversa = minv;
    matriz_camara(_selected_camera);
    _selected_camera->coordenadas.x += x;
    _selected_camera->coordenadas.y += y;
    _selected_camera->coordenadas.z += z;
}

//ROTACIONES EN MODO VUELO
void rotacionVuelo(float x, float y, float z, int alpha)
{
    matrices *minv;
    minv = malloc(sizeof(matrices));
    glMatrixMode(GL_MODELVIEW);
    glLoadMatrixf(_selected_camera->inversa->matriz);
    glRotatef(alpha, x, y, z);
    glGetFloatv(GL_MODELVIEW_MATRIX, minv->matriz);
    minv->sigptr = _selected_camera->inversa;
    _selected_camera->inversa = minv;
    matriz_camara(_selected_camera);
}

//SI LAS TRANSLACIONES ESTAN ACTIVAS LLAMA A LAS TRANSLACIONES
void transformacionCamaraMundoVuelo(float x, float y, float z, int alpha)
{
    if (trasl_activa == 1)
    {
        translacionVuelo(x, y, z);
        printf("Traslacion camara\n");
    }
    if (rotac_activa == 1)
    {
        rotacionVuelo(-y, x, z, alpha);
        printf("Rotacion camara\n");
    }
}
```

Estas se hacen mediante tres funciones. La principal llama a una de las dos auxiliares en base a que estamos haciendo, si translaciones o rotaciones. Las transformaciones de cámara se hacen sobre la que está seleccionada (la que está visualizando el mundo).

En io.c - transformaciones de cámara en modo análisis:

```
//TRANSLACIONES EN MODO ANALISIS
void translacionAnalisis(float x, float y, float z)
{
    GLfloat m_at[16] = {
        1, 0, 0, 0,
        0, 1, 0, 0,
        0, 0, 1, 0,
        _selected_object->primptr->matriz[12],
        _selected_object->primptr->matriz[13],
        _selected_object->primptr->matriz[14], 1
    };
    GLfloat m_menos_at[16] = {
        1, 0, 0, 0,
        0, 1, 0, 0,
        0, 0, 1, 0,
        (-1) * _selected_object->primptr->matriz[12],
        (-1) * _selected_object->primptr->matriz[13],
        (-1) * _selected_object->primptr->matriz[14], 1
    };
    glMatrixMode(GL_MODELVIEW);
    glLoadMatrixf(m_at);
    glTranslatef(x, y, z);
    glMultMatrixf(m_menos_at);
    matriz_inversa_camara(_selected_camera); //calcula la m inversa mcsr
    glMultMatrixf(_selected_camera->inversa->matriz);
    glGetFloatv(GL_MODELVIEW_MATRIX, _selected_camera->inversa->matriz);
    matriz_camara(_selected_camera);
    _selected_camera->coordenadas.x = _selected_camera->inversa->matriz[12];
    _selected_camera->coordenadas.y = _selected_camera->inversa->matriz[13];
    _selected_camera->coordenadas.z = _selected_camera->inversa->matriz[14];
}

//ROTACIONES EN MODO ANALISIS
void rotacionAnalisis(float x, float y, float z, int alpha)
{
    GLfloat m_at[16] = {
        1, 0, 0, 0,
        0, 1, 0, 0,
        0, 0, 1, 0,
        _selected_object->primptr->matriz[12],
        _selected_object->primptr->matriz[13],
        _selected_object->primptr->matriz[14], 1
    };
    GLfloat m_menos_at[16] = {
        1, 0, 0, 0,
        0, 1, 0, 0,
        0, 0, 1, 0,
        (-1) * _selected_object->primptr->matriz[12],
        (-1) * _selected_object->primptr->matriz[13],
        (-1) * _selected_object->primptr->matriz[14], 1
    };
    glMatrixMode(GL_MODELVIEW);
    glLoadMatrixf(m_at);
    glRotatef(alpha, x, y, z);
    glMultMatrixf(m_menos_at);
    matriz_inversa_camara(_selected_camera); //calcula la m inversa mcsr
    glMultMatrixf(_selected_camera->inversa->matriz);
    glGetFloatv(GL_MODELVIEW_MATRIX, _selected_camera->inversa->matriz);
    matriz_camara(_selected_camera);
    _selected_camera->coordenadas.x = _selected_camera->inversa->matriz[12];
    _selected_camera->coordenadas.y = _selected_camera->inversa->matriz[13];
    _selected_camera->coordenadas.z = _selected_camera->inversa->matriz[14];
}
```

En io.c - transformaciones en objetos:

```
//ESTA FUNCION HACE QUE AL ROTAR UN OBJETO LA CAMARA ROTE CON EL
void rotacionCamObj(float x, float y, float z, int alpha)
{
    GLfloat m_at[16] = {
        1, 0, 0, 0,
        0, 1, 0, 0,
```

```
        0, 0, 1, 0,
        _selected_object->primptr->matriz[12],
        _selected_object->primptr->matriz[13],
        _selected_object->primptr->matriz[14], 1
    };
    GLfloat m_menos_at[16] = {
        1, 0, 0, 0,
        0, 1, 0, 0,
        0, 0, 1, 0,
        (-1) * _selected_object->primptr->matriz[12],
        (-1) * _selected_object->primptr->matriz[13],
        (-1) * _selected_object->primptr->matriz[14], 1
    };
    glMatrixMode(GL_MODELVIEW);
    glLoadMatrixf(m_at);
    glRotatef(alpha, x, y, z);
    glMultMatrixf(m_menos_at);
    matriz_inversa_camara(_selected_object->obj_cam); //calcula la m inversa mcsr
    glMultMatrixf(_selected_object->obj_cam->inversa->matriz);
    glGetFloatv(GL_MODELVIEW_MATRIX, _selected_object->obj_cam->inversa->matriz);
    matriz_camara(_selected_object->obj_cam);
    _selected_object->obj_cam->coordenadas.x =
        _selected_object->obj_cam->inversa->matriz[12];
    _selected_object->obj_cam->coordenadas.y =
        _selected_object->obj_cam->inversa->matriz[13];
    _selected_object->obj_cam->coordenadas.z =
        _selected_object->obj_cam->inversa->matriz[14];
}

/* DADOS UNAS X, Y, Z Y UN ANGULO ALFA, SI TOCA HACER TRANSLACIONES, UNICAMENTE NOS
MOVEMOS X, Y, Z. SI TOCA HACER ESCALADOS, ESCALAMOS 1 + X/10, 1 + Y/10, 1 + Z/10 (AL
ESCALAR ESTARIAMOS ESCALANDO, POR EJEMPLO: 1.0, 1.1, 1.0, SI ESCALASEMOS LA Y, YA QUE
LAS X, Y, Z SON O BIEN 0.0 O 1.0 O -1.0) Y POR ULTIMO LAS ROTACIONES USANDO EL ANGULO
ALFA DADO Y SIMPLEMENTE LAS X, Y, Z QUE SE LE DAN, NADA MAS */
void transforLocal(float x, float y, float z, int alpha)
{
    //TRANSFORMACION EN EL SISTEMA DE REF DEL OBJETO
    matrices *m;
    m = malloc(sizeof(matrices));
    glMatrixMode(GL_MODELVIEW);
    glLoadMatrixf(_selected_object->primptr->matriz);
    if (trasl_activa == 1)
    {
        glTranslatef(x, y, z);
        glGetFloatv(GL_MODELVIEW_MATRIX, m->matriz);
        m->sigptr = _selected_object->primptr;
        _selected_object->primptr = m;
        //transformaciones de su camara
        matriz_inversa_camara(_selected_object->obj_cam);
        _selected_object->obj_cam->inversa->matriz[12] += x;
        _selected_object->obj_cam->coordenadas.x += x;
        _selected_object->obj_cam->inversa->matriz[13] += y;
        _selected_object->obj_cam->coordenadas.y += y;
        _selected_object->obj_cam->inversa->matriz[14] += z;
        _selected_object->obj_cam->coordenadas.z += z;
        matriz_camara(_selected_object->obj_cam);
    }
    else if (escal_activo == 1)
    {
        glScalef(1.0 + x / 10, 1.0 + y / 10, 1.0 + z / 10);
        glGetFloatv(GL_MODELVIEW_MATRIX, m->matriz);
        m->sigptr = _selected_object->primptr;
        _selected_object->primptr = m;
        matriz_inversa_camara(_selected_object->obj_cam);
        matriz_camara(_selected_object->obj_cam);
    }
    else
    {
        glRotatef(alpha, -y, x, z);
        glGetFloatv(GL_MODELVIEW_MATRIX, m->matriz);
        m->sigptr = _selected_object->primptr;
        _selected_object->primptr = m;
```

```
        //transformaciones de su camara
        rotacionCamObj(-y, x, z, alpha);
    }
}

/* DADOS UNAS X, Y, Z Y UN ANGULO ALFA, SI TOCA HACER TRANSLACIONES, UNICAMENTE NOS
MOVEMOS X, Y, Z. SI TOCA HACER ESCALADOS, ESCALAMOS 1 + X/10, 1 + Y/10, 1 + Z/10 (AL
ESCALAR ESTARIAMOS ESCALANDO, POR EJEMPLO: 1.0, 1.1, 1.0, SI ESCALASEMOS LA Y, YA QUE
AS X, Y, Z SON O BIEN 0.0 O 1.0 O -1.0) Y POR ULTIMO LAS ROTACIONES USANDO EL ANGULO
ALFA DADO Y SIMPLEMENTE LAS X, Y, Z QUE SE LE DAN, NADA MAS */
void transforGlobal(float x, float y, float z, int alpha)
{
    //TRANSFORMACION EN EL SISTEMA DE REF DE LA CAMARA
    matrices *m;
    matrices *minv;
    m = malloc(sizeof(matrices));
    minv = malloc(sizeof(matrices));
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    if (trasl_activa == 1)
        glTranslatef(x, y, z);
    else if (escal_activo == 1)
        glScalef(1.0 + x / 10, 1.0 + y / 10, 1.0 + z / 10);
    else
        glRotatef(alpha, -y, x, z);
    GLfloat aux[16];
    glGetFloatv(GL_MODELVIEW_MATRIX, aux);
    glMultMatrixf(_selected_object->primptr->matriz);
    glGetFloatv(GL_MODELVIEW_MATRIX, m->matriz);
    glLoadMatrixf(aux);
    glMultMatrixf(_selected_object->obj_cam->inversa->matriz);
    glGetFloatv(GL_MODELVIEW_MATRIX, minv->matriz);
    minv->sigptr = _selected_object->obj_cam->inversa;
    _selected_object->obj_cam->inversa = minv;
    matriz_camara(_selected_object->obj_cam);
    m->sigptr = _selected_object->primptr;
    _selected_object->primptr = m;
}
```

La función "rotacionCamObj" como en los comentarios se indica facilita que la cámara siga al objeto al rotar este. Las otras dos son funciones, en función del modo local o global, hacen las transformaciones pertinentes. Cuando las transformaciones son en el sistema de referencia global, primero se carga la matriz identidad en el modelview, luego se hace la transformación y por último se multiplica la matriz del objeto para aplicar las transformaciones globales. Si, en cambio, son en local, primero se carga la matriz del objeto y después se hace la transformación.

En display.c - resumen de funciones auxiliares y la función display - hay algunas funciones auxiliares que facilitan el funcionamiento de la aplicación y no merece mucho la pena mencionarlas (como cop-mat, que copia los valores de una matriz en otra). La función interesante es display.c es la siguiente:

```
/**
 * @brief Callback display function
 */
void display(void)
{
  GLint v_index, v, f, l_index;
  object3d *aux_obj = _first_object;
  GLfloat EJE_X[] = {1, 0, 0, 1};
  GLfloat EJE_Y[] = {0, 1, 0, 1};
  GLfloat EJE_Z[] = {0, 0, 1, 1};
  /* Clear the screen */
  glClear(GL_DEPTH_BUFFER_BIT | GL_COLOR_BUFFER_BIT);
  if (iluminacion == 1)
    glEnable(GL_LIGHTING);
  else
    glDisable(GL_LIGHTING);
```

```
/* Define the projection */
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
//VISTA CON PERSPECTIVA
if (_selected_camera->perspectiva == 1)
  glFrustum(_selected_camera->left, _selected_camera->right,
        _selected_camera->bottom, _selected_camera->top,
        _selected_camera->near, _selected_camera->far);
else //VISTA CON ORTOGONAL
  glOrtho(_selected_camera->left, _selected_camera->right,
      _selected_camera->bottom, _selected_camera->top,
      _selected_camera->near, _selected_camera->far);
/* Now we start drawing the object */
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
if (_selected_object != 0)
{
  glLoadMatrixf(_selected_camera->mcsr->matriz);
  glMultMatrixf(_selected_object->primptr->matriz);
  glMultMatrixf(luces[2].m_luz->matriz);
  glLightfv(GL_LIGHT2, GL_POSITION, luces[2].position);
  glLightfv(GL_LIGHT2, GL_SPOT_DIRECTION, luces[2].dir);
  glLightfv(GL_LIGHT2, GL_AMBIENT, luces[2].ambient);
  glLightfv(GL_LIGHT2, GL_DIFFUSE, luces[2].diffuse);
  glLightfv(GL_LIGHT2, GL_SPECULAR, luces[2].specular);
  glLightf(GL_LIGHT2, GL_SPOT_EXPONENT, 5.0);
  glLightf(GL_LIGHT2, GL_SPOT_CUTOFF, luces[2].amplitud);
}
glLoadIdentity();
if (_selected_camera != 0)
{
  glMultMatrixf(luces[3].m_luz->matriz);
  glLightfv(GL_LIGHT3, GL_POSITION, luces[3].position);
  glLightfv(GL_LIGHT3, GL_SPOT_DIRECTION, luces[3].dir);
  glLightfv(GL_LIGHT3, GL_AMBIENT, luces[3].ambient);
  glLightfv(GL_LIGHT3, GL_DIFFUSE, luces[3].diffuse);
  glLightfv(GL_LIGHT3, GL_SPECULAR, luces[3].specular);
  glLightf(GL_LIGHT3, GL_SPOT_EXPONENT, 5.0);
  glLightf(GL_LIGHT3, GL_SPOT_CUTOFF, luces[3].amplitud);
}
//Sol
glLoadMatrixf(_selected_camera->mcsr->matriz);
glLightfv(GL_LIGHT0, GL_POSITION, luces[0].position);
glLightfv(GL_LIGHT0, GL_AMBIENT, luces[0].ambient);
glLightfv(GL_LIGHT0, GL_DIFFUSE, luces[0].diffuse);
glLightfv(GL_LIGHT0, GL_SPECULAR, luces[0].specular);
//Bombilla
glLoadMatrixf(_selected_camera->mcsr->matriz);
glLightfv(GL_LIGHT1, GL_POSITION, luces[1].position);
glLightfv(GL_LIGHT1, GL_SPOT_DIRECTION, luces[1].dir);
glLightfv(GL_LIGHT1, GL_AMBIENT, luces[1].ambient);
glLightfv(GL_LIGHT1, GL_DIFFUSE, luces[1].diffuse);
glLightfv(GL_LIGHT1, GL_SPECULAR, luces[1].specular);
glLightf(GL_LIGHT1, GL_CONSTANT_ATTENUATION, 1.5);
glLightf(GL_LIGHT1, GL_LINEAR_ATTENUATION, 0.1);
glLightf(GL_LIGHT1, GL_QUADRATIC_ATTENUATION, 0.05);
if (luces[0].encendida == 1)
  glEnable(GL_LIGHT0);
else
  glDisable(GL_LIGHT0);
if (luces[1].encendida == 1)
  glEnable(GL_LIGHT1);
else
  glDisable(GL_LIGHT1);
if (luces[2].encendida == 1)
  glEnable(GL_LIGHT2);
else
  glDisable(GL_LIGHT2);
if (luces[3].encendida == 1)
  glEnable(GL_LIGHT3);
else
  glDisable(GL_LIGHT3);
```

```
  /*First, we draw the axes*/
  glDisable(GL_LIGHTING);
  draw_axes();
  if (iluminacion == 1)
    glEnable(GL_LIGHTING);
  /*Now each of the objects in the list*/
  while (aux_obj != 0)
  {
    glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, aux_obj->mater->ambient);
    glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE, aux_obj->mater->diffuse);
    glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, aux_obj->mater->specular);
    glMaterialf(GL_FRONT_AND_BACK, GL_SHININESS, aux_obj->mater->shininess * 128.0);
    /* Select the color, depending on whether the current object is the selected one or not */
    if (aux_obj == _selected_object)
      glColor3f(KG_COL_SELECTED_R, KG_COL_SELECTED_G, KG_COL_SELECTED_B);
    else
      glColor3f(KG_COL_NONSELECTED_R, KG_COL_NONSELECTED_G, KG_COL_NONSELECTED_B);
    /* Draw the object; for each face create a new polygon with the corresponding vertices */
    glLoadMatrixf(_selected_camera->mcsr->matriz);
    glMultMatrixf(aux_obj->primptr->matriz);
    for (f = 0; f < aux_obj->num_faces; f++)
    {
      glBegin(GL_POLYGON);
      if (flat_smooth == 1)
        glNormal3f(aux_obj->face_table[f].vnormal.x,
                   aux_obj->face_table[f].vnormal.y,
                   aux_obj->face_table[f].vnormal.z);
      for (v = 0; v < aux_obj->face_table[f].num_vertices; v++)
      {
        v_index = aux_obj->face_table[f].vertex_table[v];
        if (flat_smooth == 0)
          glNormal3f(aux_obj->vertex_table[v_index].vnormal.x,
                     aux_obj->vertex_table[v_index].vnormal.y,
                     aux_obj->vertex_table[v_index].vnormal.z);
        glVertex3d(aux_obj->vertex_table[v_index].coord.x,
                aux_obj->vertex_table[v_index].coord.y,
                aux_obj->vertex_table[v_index].coord.z);
      }
      glEnd();
    }
    glBegin(GL_LINES);
    glEnd();
    aux_obj = aux_obj->next;
  }
  glutSwapBuffers();
  /*Do the actual drawing*/
  glFlush();
}
```

Si la iluminación está activa se aplica en el display. Primero se decide la vista de la cámara seleccionada. Después se carga la matriz identidad y se aplican la cámara, el objeto seleccionado y su foco, después la cámara y su foco. Más tarde, van el sol y la bombilla. Si están los parámetros de encendido se activan las luces pertinentes y por último empiezan a dibujarse los objetos.

En load-obj-joseba.c:

```
// material del que se componen los objetos (RUBY)
void setMaterial(object3d *obj)
{
    obj->mater = malloc(sizeof(material));
    obj->mater->ambient[0] = 0.1745;
    obj->mater->ambient[1] = 0.01175;
    obj->mater->ambient[2] = 0.01175;
    obj->mater->ambient[3] = 1;
    obj->mater->diffuse[0] = 0.61424;
    obj->mater->diffuse[1] = 0.4136;
    obj->mater->diffuse[2] = 0.4136;
    obj->mater->diffuse[3] = 1;
    obj->mater->specular[0] = 0.727811;
    obj->mater->specular[1] = 0.626959;
```

```
    obj->mater->specular[2] = 0.626959;
    obj->mater->specular[3] = 1;
    obj->mater->shininess = 0.6;
}

// Inicializacion de la camara asociada al objeto
void initCamObj(object3d *obj)
{
    obj->obj_cam = malloc(sizeof(camera));
    matrices *mmcsr;
    mmcsr = malloc(sizeof(matrices));
    matrices *minv;
    minv = malloc(sizeof(matrices));
    obj->obj_cam->coordenadas.x = 0.0;
    obj->obj_cam->coordenadas.y = 0.0;
    obj->obj_cam->coordenadas.z = obj->max.z;
    obj->obj_cam->pto_at.x = 0.0;
    obj->obj_cam->pto_at.y = 0.0;
    obj->obj_cam->pto_at.z = obj->max.z + 1.0;
    obj->obj_cam->vup.x = 0.0;
    obj->obj_cam->vup.y = 1.0;
    obj->obj_cam->vup.z = 0.0;
    obj->obj_cam->next = NULL;
    obj->obj_cam->tiene_obj = 1;
    obj->obj_cam->left = LEFT;
    obj->obj_cam->right = RIGHT;
    obj->obj_cam->bottom = BOTTOM;
    obj->obj_cam->top = TOP;
    obj->obj_cam->near = NEAR;
    obj->obj_cam->far = FAR;
    obj->obj_cam->perspectiva = 1;
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    gluLookAt(obj->obj_cam->coordenadas.x,
              obj->obj_cam->coordenadas.y,
              obj->obj_cam->coordenadas.z,
              obj->obj_cam->pto_at.x,
              obj->obj_cam->pto_at.y,
              obj->obj_cam->pto_at.z,
              obj->obj_cam->vup.x,
              obj->obj_cam->vup.y,
              obj->obj_cam->vup.z);
    glGetFloatv(GL_MODELVIEW_MATRIX, mmcsr->matriz);
    mmcsr->sigptr = NULL;
    obj->obj_cam->mcsr = mmcsr;
    minv->matriz[0] = 1;
    minv->matriz[1] = 0;
    minv->matriz[2] = 0;
    minv->matriz[3] = 0;
    minv->matriz[4] = 0;
    minv->matriz[5] = 1;
    minv->matriz[6] = 0;
    minv->matriz[7] = 0;
    minv->matriz[8] = 0;
    minv->matriz[9] = 0;
    minv->matriz[10] = 1;
    minv->matriz[11] = 0;
    minv->matriz[12] = 0;
    minv->matriz[13] = 0;
    minv->matriz[14] = obj->max.z;
    minv->matriz[15] = 1;
    minv->sigptr = NULL;
    obj->obj_cam->inversa = minv;
    _last_camera->next = obj->obj_cam;
    obj->obj_cam->prev = _last_camera;
    _last_camera = _last_camera->next;
}
```

Se inicializan los valores del material del objeto. También, se inicializan los valores de las cámaras de cada objeto. En el resto del documento se cargan los objetos.

En main.c - inicialización de luces:

```
// INICIALIZA LAS LUCES DEL MUNDO
void init_luces()
{
  //SOL
  luces[0].position[0] = 1;
  luces[0].position[1] = 1;
  luces[0].position[2] = 0;
  luces[0].position[3] = 0;
  luces[0].dir[0] = 0.0;
  luces[0].dir[1] = 0.0;
  luces[0].dir[2] = 0.0;
  luces[0].ambient[0] = 0;
  luces[0].ambient[1] = 0;
  luces[0].ambient[2] = 0;
  luces[0].ambient[3] = 1;
  luces[0].diffuse[0] = 1;
  luces[0].diffuse[1] = 1;
  luces[0].diffuse[2] = 1;
  luces[0].diffuse[3] = 1;
  luces[0].specular[0] = 1;
  luces[0].specular[1] = 1;
  luces[0].specular[2] = 1;
  luces[0].specular[3] = 1;
  luces[0].encendida = 1; // 0 es que esta apagada
  luces[0].m_luz = malloc(sizeof(matrices));
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
  glTranslatef(luces[0].position[0], luces[0].position[1], luces[0].position[2]);
  glGetFloatv(GL_MODELVIEW_MATRIX, luces[0].m_luz->matriz);
  luces[0].m_luz->sigptr = NULL;
  _selected_light = 0;
  //BOMBILLA
  luces[1].position[0] = 0.0;
  luces[1].position[1] = 4.0;
  luces[1].position[2] = 0.0;
  luces[1].position[3] = 1;
  luces[1].dir[0] = 0.0;
  luces[1].dir[1] = 0.0;
  luces[1].dir[2] = 0.0;
  luces[1].ambient[0] = 0;
  luces[1].ambient[1] = 0;
  luces[1].ambient[2] = 0;
  luces[1].ambient[3] = 1;
  luces[1].diffuse[0] = 1;
  luces[1].diffuse[1] = 1;
  luces[1].diffuse[2] = 1;
  luces[1].diffuse[3] = 1;
  luces[1].specular[0] = 1;
  luces[1].specular[1] = 1;
  luces[1].specular[2] = 1;
  luces[1].specular[3] = 1;
  luces[1].encendida = 0; // 0 es que esta apagada
  luces[1].m_luz = malloc(sizeof(matrices));
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
  glTranslatef(luces[1].position[0], luces[1].position[1], luces[1].position[2]);
  glGetFloatv(GL_MODELVIEW_MATRIX, luces[1].m_luz->matriz);
  luces[1].m_luz->sigptr = NULL;
  // Foco_obj
  luces[2].position[0] = 0.0;
  luces[2].position[1] = 0.0;
  luces[2].position[2] = 0.0;
  luces[2].position[3] = 1;
  luces[2].dir[0] = 0.0;
  luces[2].dir[1] = 0.0;
  luces[2].dir[2] = 1.0;
  luces[2].ambient[0] = 0.0;
  luces[2].ambient[1] = 0.0;
  luces[2].ambient[2] = 0.0;
  luces[2].ambient[3] = 1;
  luces[2].diffuse[0] = 0.5;
  luces[2].diffuse[1] = 0.5;
```

```
  luces[2].diffuse[2] = 0.5;
  luces[2].diffuse[3] = 1;
  luces[2].specular[0] = 0.5;
  luces[2].specular[1] = 0.5;
  luces[2].specular[2] = 0.5;
  luces[2].specular[3] = 1;
  luces[2].encendida = 0; // 0 es que esta apagada
  luces[2].amplitud = 45.0;
  luces[2].m_luz = malloc(sizeof(matrices));
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
  glTranslatef(luces[2].position[0], luces[2].position[1], luces[2].position[2]);
  glGetFloatv(GL_MODELVIEW_MATRIX, luces[2].m_luz->matriz);
  luces[2].m_luz->sigptr = NULL;
  // Foco_cam
  luces[3].position[0] = 0.0;
  luces[3].position[1] = 0.0;
  luces[3].position[2] = 0.0;
  luces[3].position[3] = 1;
  luces[3].dir[0] = 0.0;
  luces[3].dir[1] = 0.0;
  luces[3].dir[2] = -1.0;
  luces[3].ambient[0] = 0.5;
  luces[3].ambient[1] = 0.5;
  luces[3].ambient[2] = 0.5;
  luces[3].ambient[3] = 1;
  luces[3].diffuse[0] = 0.75;
  luces[3].diffuse[1] = 0.75;
  luces[3].diffuse[2] = 0.75;
  luces[3].diffuse[3] = 1;
  luces[3].specular[0] = 0.5;
  luces[3].specular[1] = 0.5;
  luces[3].specular[2] = 0.5;
  luces[3].specular[3] = 1;
  luces[3].encendida = 0; // 0 es que esta apagada
  luces[3].amplitud = 45.0;
  luces[3].m_luz = malloc(sizeof(matrices));
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
  glTranslatef(luces[3].position[0], luces[3].position[1], luces[3].position[2]);
  glGetFloatv(GL_MODELVIEW_MATRIX, luces[3].m_luz->matriz);
  luces[3].m_luz->sigptr = NULL;
}
```

Se usa el número de luces mínimo, el sol, la bombilla, un foco en el objeto seleccionado y uno en la cámara. El sol está apuntando desde arriba a la derecha, en la dirección (1,1,0). La bombilla esta encima del centro del mundo (en la posición 0,4,0), y como bombilla que es ilumina en todas direcciones, tiene atenuación (aunque eso se declara en el display). El foco del objeto aunque se inicialice con ciertos valores, muchos como la posición y la dirección dependen del objeto seleccionado. Lo mismo con el foco de la cámara, pero que depende de la cámara seleccionada.

En main.c - flat smooth e inicialización de la primera cámara:

```
/** GENERAL INITIALIZATION **/
void initialization()
{
  /*Initialization of all the variables with the default values*/
  //_window_ratio = (GLdouble) KG_WINDOW_WIDTH / (GLdouble) KG_WINDOW_HEIGHT;
  /*Definition of the background color*/
  glClearColor(KG_COL_BACK_R, KG_COL_BACK_G, KG_COL_BACK_B, KG_COL_BACK_A);
  /*Definition of the method to draw the objects*/
  glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
  glEnable(GL_DEPTH_TEST); // activar el test de profundidad (Z-buffer)
  glEnable(GL_LIGHTING);
  glEnable(GL_NORMALIZE);
  init_luces();
  if (flat_smooth == 1)
    glShadeModel(GL_SMOOTH); // hacen falta los vectores normales de cada vertice
  else
```

```
    glShadeModel(GL_FLAT); // basta con vector normal del poligono
  //PRIMERA CAMARA DEL MUNDO POR DEFECTO
  if (_first_camera == 0)
  {
    camera *c;
    c = malloc(sizeof(camera));
    matrices *mmcsr;
    mmcsr = malloc(sizeof(matrices));
    matrices *minv;
    minv = malloc(sizeof(matrices));
    c->coordenadas.x = 0.0;
    c->coordenadas.y = 0.0;
    c->coordenadas.z = 10.0;
    c->pto_at.x = 0.0;
    c->pto_at.y = 0.0;
    c->pto_at.z = 0.0;
    c->vup.x = 0.0;
    c->vup.y = 1.0;
    c->vup.z = 0.0;
    c->left = LEFT;
    c->right = RIGHT;
    c->bottom = BOTTOM;
    c->top = TOP;
    c->near = NEAR;
    c->far = FAR;
    c->perspectiva = 1;
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    gluLookAt(c->coordenadas.x,
          c->coordenadas.y,
          c->coordenadas.z,
          c->pto_at.x,
          c->pto_at.y,
          c->pto_at.z,
          c->vup.x,
          c->vup.y,
          c->vup.z);
    glGetFloatv(GL_MODELVIEW_MATRIX, mmcsr->matriz);
    mmcsr->sigptr = NULL;
    c->mcsr = mmcsr;
    minv->matriz[0] = mmcsr->matriz[0];
    minv->matriz[1] = mmcsr->matriz[4];
    minv->matriz[2] = mmcsr->matriz[8];
    minv->matriz[3] = 0;
    minv->matriz[4] = mmcsr->matriz[1];
    minv->matriz[5] = mmcsr->matriz[5];
    minv->matriz[6] = mmcsr->matriz[9];
    minv->matriz[7] = 0;
    minv->matriz[8] = mmcsr->matriz[2];
    minv->matriz[9] = mmcsr->matriz[6];
    minv->matriz[10] = mmcsr->matriz[10];
    minv->matriz[11] = 0;
    minv->matriz[12] = 0;
    minv->matriz[13] = 0;
    minv->matriz[14] = 10;
    minv->matriz[15] = 1;
    minv->sigptr = NULL;
    c->inversa = minv;
    c->tiene_obj = 0; //se supone que es la camara del mundo
    c->next = 0;
    c->prev = 0;
    _first_camera = c;
    _selected_camera = _first_camera;
    _last_camera = _first_camera;
  }
}
```

4 Posibles mejoras

Se podrían mejorar aspectos en lo referente a la interactuación con el usuario como poder modificar algunos valores fijos (ej. ángulo de rotación). También se podría crear una lista de materiales para poder cambiarlos o incluso dar la posibilidad de que el usuario cree un nuevo material introduciendo los datos necesarios.

Internamente se puede mejorar la claridad del código, ya que es muy extenso. Se puede hacer más modular, separación en varios ficheros añadidos, uso de mas macros para facilidad del manejo y eficiencia en el tratamiento de estructuras y datos en lo referente a tiempo y espacio.

En conclusión, se pueden mejorar muchos aspectos en la aplicación y añadir funcionalidades, pero se necesita mas tiempo y calma para ello.

5 Anexo I - manual de usuario

La función print_help() muestra la información resumida que se describe a continuación. La salida se aprecia en la imagen del apartado 1 (introducción y objetivos).

1. Instrucciones
   - Pulsar la 'o' o 'O' permite trabajar sobre objetos.
   - Pulsar la 'k' o 'K' permite trabajar sobre cámaras.
   - Pulsar la 'a' o 'A' permite trabajar sobre la iluminación.
2. Objetos - transformaciones geométricas. Se aplican sobre el objeto seleccionado.
   - Pulsar la 'f' o 'F' carga nuevos objetos. Después de pulsar la tecla, deberemos escribir el "path" del objeto que queremos cargar.
   - Las siguientes instrucciones funcionan junto a las teclas: izq (en -x), dcha (en +x), abajo (en -y), arriba (en +y), avpage (en +z), repage (en -z).
     - Pulsar la 'm' o 'M' permite trasladar los objetos, tanto en el sistema de referencia local de estos o el global (no es lo mismo ir hacia abajo en el "abajo" del objeto que el del mundo).
     - Pulsar la 'b' o 'B' permite rotar los objetos, al igual que en las translaciones, tanto en el sistema local o global de referencia.
     - Pulsar la 't' o 'T' permite escalar los objetos.
   - Pulsar la 'g' o 'G' cambia de modo local a modo global.
   - Pulsar la 'l' o 'L' cambia de modo global a modo local.
   - Pulsar la tecla + (sin el control) escala los objetos en todas las direcciones. Para ello recordamos que es necesario tener la opción de escalado activa.
   - Pulsar la tecla - (sin el control) escala los objetos en todas las direcciones. Para ello recordamos como en el anterior apartado que es necesario tener la opción de escalado activa.
3. Cámaras - visualización del mundo y objetos. Las cámaras permiten visualizar el mundo en 3D. La primera y predeterminada es la que visualiza el mundo un poco alejado del centro del mismo, 10 unidades en concreto.

- Al crear un nuevo objeto, se crea una nueva cámara, es la cámara asociada al objeto.
- Pulsar la 'f' o 'F' crea una nueva cámara, en un punto distinto al de la original, en concreto el (x = 5, y = 5, z = 10) apuntando al centro del mundo, lo cual quiere decir que estaremos viendo el mundo desde un ángulo superior-derecho respecto al origen. Se pueden crear las cámaras que el usuario desee pero estas se crearan siempre en ese mismo punto que acabábamos de describir.
- Pulsar la 'c' cambia a la siguiente cámara y visualiza lo que ve la misma.
- Pulsar la 'C' visualiza lo que ve el objeto seleccionado (con su cámara asociada).
- Las siguientes instrucciones funcionan junto a las teclas: izq (en -x), dcha (en +x), abajo (en -y), arriba (en +y), avpage (en +z), repage (en -z).
  - Pulsar la 'm' o 'M' permite trasladar las cámaras, ya sea en vuelo o en modo análisis, en modo análisis solo podemos acercarnos y alejarnos de los objetos (+z o -z).
  - Pulsar la 'b' o 'B' permite rotar las cámaras, tanto en modo vuelo como en modo análisis.
- Pulsar la 'p' o 'P' cambia de vista con perspectiva a vista ortogonal y viceversa.
- Pulsar la 'g' o 'G' cambia de modo vuelo a modo análisis.
- Pulsar la 'l' o 'L' cambia de modo análisis a modo vuelo.

4. Iluminación. La iluminación permite iluminar la escena mediante distintas luces. Habrá 4 tipos de luces: luz del sol, luz de bombilla, foco del objeto y foco de la cámara. Al iniciar el mundo permanecerán todas apagadas menos la del sol, que apunta en un ángulo de 45º.

   - Al pulsar la 'a' o 'A' se activa el modo de iluminación como hemos dicho antes, pero esto también es lo que permite activar las transformaciones a las fuentes de luz (siempre que estas sean posibles).
   - Mediante las teclas f1, f2, f3 y f4 podremos encender y apagar las 4 luces correspondientes. Con f1 el sol, f2 la bombilla, f3 el foco del objeto seleccionado y f4 el foco de la cámara seleccionada.
   - Con los número del 1 al 4 podremos seleccionar la luz sobre la que queremos aplicar alguna transformación. El orden es el mismo que el anteriormente mencionado, el 1 para el sol, 2 para la bombilla, 3 para el foco del objeto y 4 para el foco de la cámara.
   - Pulsar el '+', si la fuente de luz que tenemos seleccionada es de tipo foco, es decir, el foco del objeto o el de la cámara seleccionada, aumentará el ángulo de apertura del foco hasta un máximo de 90 grados.
   - Pulsar el '-', si la fuente de luz que tenemos seleccionada es de tipo foco, es decir, el foco del objeto o el de la cámara seleccionada, aumentará el ángulo de apertura del foco hasta un mínimo de 0 grados.
   - Las siguientes instrucciones funcionan junto a las teclas: izq (en -x), dcha (en +x), abajo (en -y), arriba (en +y), avpage (en +z), repage (en -z).

  - Pulsar la 'm' o 'M' permite trasladar las luces, como los dos focos están asociados o bien a un objeto o bien a una cámara, estos se moverán cuando se muevan los mismos. Un sol no tiene mucho sentido que se traslade como quiera, por lo tanto las traslaciones no son efectivas en él, por tanto las traslaciones se podrán aplicar solamente a la bombilla.
  - Pulsar la 't' o 'T' permite escalar las luces, esto solo es aplicable a los focos. Cuando tienen el escalado activado, se podrá pulsar el '+' y el '-', y harán sus respectivas funciones.
  - Pulsar la 'b' o 'B' permite rotar las luces, tanto en modo local como en modo global. Los focos solo se moverán en su sistema local. La bombilla y el sol solo rotarán en modo global.
- Pulsar la 'g' o 'G' cambia de modo local a modo global.
- Pulsar la 'l' o 'L' cambia de modo global a modo local.

5. Otras instrucciones

   - Pulsar el tabulador cambia el objeto seleccionado al siguiente introducido a él en el mundo.
   - Pulsar la tecla suprimir borra el objeto seleccionado.
   - Pulsar la tecla "esc" sirve para cerrar la aplicación.
   - Antes ya hemos mencionado las palabras local y global, pues bien, pulsar 'l' cambia al modo local, para hacer las transformaciones en el sistema de referencia del objeto y pulsar la 'g' cambia al modo global, para hacerlas en el sistema de referencia del mundo.
   - Pulsar control más 'z' ("Ctrl" + "z") permite deshacer los cambios que hemos hecho sobre el objeto seleccionado.
   - Independientemente de en que modo estemos, pulsar la tecla control más + o - aumenta o disminuye el volumen de visión de la cámara que estamos usando ("Ctrl" + "+" o "Ctrl" + "-").
   - Pulsar la tecla "F9" activará y desactivará la iluminación de las luces.
   - Pulsar la tecla "F12" sirve para cambiar la iluminación entre "flat" y "smooth".

6 Anexo II - notas

1. Las opciones de translación, escalado y rotación son excluyentes entre ellas. No podemos escalar y rotar al mismo tiempo, por ejemplo.

2. Los modos locales y globales o de vuelo y análisis también son excluyentes, o estamos en un modo o en el otro.

3. Ocurre igual con las opciones objeto, cámara y luz, solo podemos estar en una al mismo tiempo.

4. Cuando se cambia de entre las opciones de objeto y cámara se ponen unas opciones por defecto.

- Objeto: modo local y traslaciones del objeto activadas.
- Cámara: modo local y traslaciones de la cámara activadas.
- Iluminación: modo local y traslaciones de las luces activadas.

5. El modo vuelo es el modo en el que la cámara se mueve con libertad por el mundo, sin restricciones, mientras que el análisis tiene una esfera alrededor del objeto sobre la que únicamente puede rotar y acercarse o alejarse del mismo (aumentando o disminuyendo esa esfera.

6. Por comodidad, hay opciones que nos permitan hacer cosas como aumentar el volumen de visión de la cámara sin tener que salir del modo objeto.

7. También hay otras como suprimir que simplemente funcionan en cualquier modo. Estas opciones están en la sección de "Otras instrucciones".