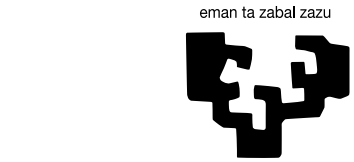

Gráficos por computador

I. Hidalgo

December 31, 2021

Abstract

Desarrollo de una aplicación de renderizado de objetos mediante la librería de C OpenGL. El usuario será capaz de cargar y realizar transformaciones (rotado, translación y escalado) sobre los objetos en el escenario, las cámaras desde las que se visualizan y las fuentes de iluminación.

1 Funcionamiento de nuestra aplicación

Nuestra aplicación consta de tres modos que permitirán al usuario modificar diferentes aspectos del escenario.

1.1 Modo Objeto

Al arrancar la aplicación se dibujará un escenario vacío. Para poder adentrarnos más en las funcionalidades lo primero que deberemos hacer es cargar un objeto pulsando la tecla F y escribiendo la ruta hasta su descriptor.

La aplicación cargará objetos en extensión .obj, para ello leerá la información del fichero y la guardará en una tabla como la de la Figura 1. Una vez cargado un objeto y guardado en una estructura nuestro programa lo dibuja y se mostrará en el escenario.

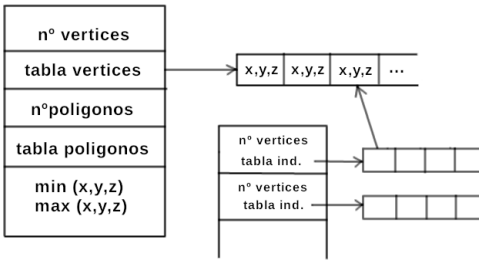

Figure 1: Tabla de un objeto

Con el objeto en nuestro mundo ya es posible transformarlo. Usaremos las teclas R, T y E para seleccionar el tipo de transformación y usaremos las flechas de direccionamiento y las teclas AvPag y RePag para aplicar la transformación. Las transformaciones se pueden realizar bien en el sistema de referencia local correspondiente al objeto o en el sistema global del mundo, para cambiar entre estas dos opciones se usan las teclas G y L.

Si además queremos cargar más objetos podemos volver a pulsar la tecla F y cargar uno nuevo junto con los que ya teníamos cargados previamente. Usaremos TAB para navegar entre las entidades cargados, podremos diferencias cual de ellas es la que está seleccionada ya que está tendrá un cubo que la delimite.

Además si nos cansamos de un objeto o nuestra máquina no puede renderizar tantos como los que hemos cargado se puede eliminar el objeto que tengamos seleccionado pulsando la tecla SUPR.

1.2 Modo Cámara

Nuestro escenario se visualiza en un principio desde el punto de vista de una cámara fija que se carga al iniciar la aplicación, pero esto puede variar si cambiamos al modo Cámara pulsando la tecla C. De esta manera estaremos aplicando cambios a la cámara y no a los objetos del escenario.

La cámara tiene dos modos en los que se puede mover: modo vuelo y modo análisis. Podemos cambiar entre estos dos métodos de desplazamiento pulsando las teclas L y G respectivamente.

- Modo Vuelo. En este modo la cámara se comporta como un objeto independiente. Podremos rotarla usando las flechas de direccionamiento y hacer que avance y retroceda con las teclas AvPag y RePag. También podremos variar su campo de visión pulsando la tecla E. En este modo usaremos AvPag y RePag para aumentar o reducir el zoom general y las flechas para afectar al zoom en un solo eje.

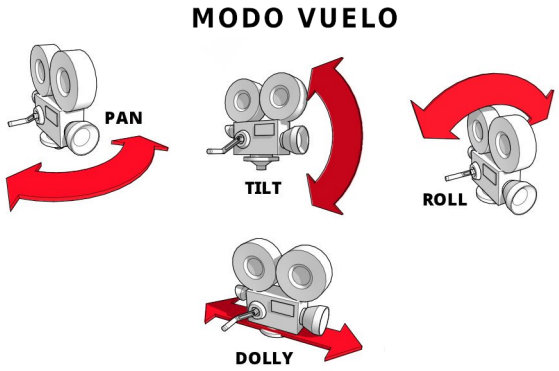

Figure 2: Cámara en modo vuelo

- Modo Análisis. En este modo la cámara apuntará al objeto seleccionado y todos sus movimientos dependerán de él. Usando las flechas moveremos la cámara en los ejes X y Y tomando como origen el objeto seleccionado, de esta manera la cámara estará orbitando sobre el objeto seleccionado. Con las teclas AvPag y RePag sin embargo podremos alejar o acercar la cámara al objeto seleccionado.

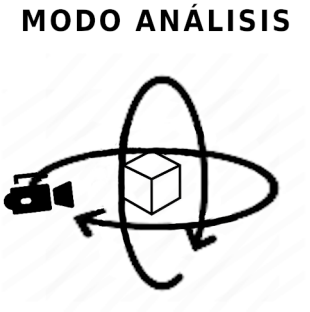

Figure 3: Cámara en modo análisis

Si cambiamos el objeto seleccionado estando en este modo la cámara apuntará al nuevo objeto seleccionado y sus movimientos dependerán ahora de él.

1.3 Modo Iluminación

Por último para cambiar las luces del escenario deberemos entrar en el modo iluminación pulsando la tecla A. En este modo podremos cambiar las luces que iluminan los objetos del escenario, mover su posición, amplia/reducir la apertura de los focos, etc.

Tendremos cuatro fuentes de iluminación que podremos activar/desactivar mediante las teclas F1-F4.

1. Sol
2. Bombilla
3. Foco del objeto seleccionado
4. Foco de la cámara

Además también podremos seleccionarlas con las teclas numéricas 1-4 y así realizar transformaciones sobre ellas mediante las flechas y las teclas + y -.

2 Datos recibidos

Los objetos a dibujar son estructuras 3D formadas por diversas lineas que crean polígonos, dichos polígonos son las caras de nuestro objeto modelado. En el fichero .obj se definen los puntos y polígonos que conforman el objeto mediante la siguiente sintaxis.

- Vértices → las lineas del fichero que empiezan por el carácter 'v' tienen información sobre los vértices del objeto. Seguidos del carácter 'v' separados por un espacio están las coordenadas x, y, z del vértice en formato float.

  Ej: v -1.000000 1.000000 1.000000

- Nº vértices → tras las lineas correspondientes a los vértices habrá una linea que comience por el carácter '#', que contendrá el número total de vértices del objeto.

  Ej: # 8 vértices

- Polígonos → después de la linea que contiene el nº de vértices, separada por una linea vacía, estarán las lineas de los polígonos, que comienzan por el carácter 'p' y estarán seguidos de varios números, que representan el índice de los vértices que los forman.

  Ej: f 8 7 6 5
  (Cuadrado formado por los vértices con índice 8, 7, 6 y 5)

- Nº polígonos → finalmente, la última linea del fichero contendrá el número de polígonos por los que está formado el objeto, comenzará por el carácter '#'.

  Ej: # 6 elements

Para guardar los objetos el programa realizará dos lecturas, en la primera de ellas buscará el número de vértices y polígonos por los que está formado el objeto, de esta manera podrá saber cuanta memoria deberá reservar para la estructura de objeto vista en la Figura 1. En la segunda pasada guardará la información de los vértices y los polígonos en la estructura creada previamente.

3 Funciones de teclado

La interacción del usuario con la aplicación consistirá en presionar varias teclas que tendrán diversas funciones. En primer lugar tenemos varias teclas de propósito general:

- ? → Imprime la ayuda de la aplicación, que tiene la función de cada tecla.
- ESC → Salir de la aplicación.
- F,f → Cargar un objeto. Una vez pulsada la tecla, el programa nos solicitará la ruta desde el directorio actual hasta el fichero .obj del objeto que queramos cargar.
- TAB → Navegar entre los objetos cargados.
- SUPR → Borrar el objeto seleccionado.

El resto de las teclas tendrán diferentes funciones dependiendo del modo estemos.

- O,o → Modo Objeto
- C,c → Modo Cámara
- A,a → Modo Iluminación

3.1 Teclado: Objeto

En este modo podremos transformar el objeto seleccionado. Los cambios se pueden aplicar en función del sistema de referencia del objeto o de manera global.

- G,g → Activar transformaciones en el sistema de referencia del mundo (transformaciones globales).
- L,l → Activar transformaciones en el sistema de referencia local del objeto.

Además debemos determinar que tipo de transformaciones se aplicarán sobre el objeto.

- T,t → Activar traslación (desactiva rotación y escalado).
- R,r → Activar rotación (desactiva traslación y escalado).
- E,e → Activar escalado (desactiva rotación y traslación).

Una vez escogido el sistema de referencia se aplicarán las transformaciones seleccionadas (traslación, rotación o escalado) usando las siguiente teclas.

- UP → Trasladar +Y; Escalar + Y; Rotar +X
- DOWN → Trasladar -Y; Escalar - Y; Rotar -X
- RIGHT → Trasladar +X; Escalar +X; Rotar +Y
- LEFT → Trasladar -X; Escalar -X; Rotar -Y
- AVPAG → Trasladar +Z; Escalar +Z; Rotar +Z
- REPAG → Trasladar -Z; Escalar - Z; Rotar -Z
- + → Escalar + en todos los ejes (solo objetos)
- - → Escalar - en todos los ejes (solo objetos)

Por último también es posible deshacer los cambios realizados sobre un objeto.

- CTRL + Z,z → Deshacer.

3.2 Teclado: Cámara

En el modo cámara realizaremos cambios sobre la cámara seleccionada, desde la cual estaremos viendo el escenario. En primer lugar deberemos establecer como queremos que se mueva la cámara.

- G,g → Cámara en modo análisis (mirando al objeto seleccionado).
- L,l → Cámara en modo vuelo. Transformaciones en el sistema local de la cámara.

También es importante definir el tipo de proyección con la que se verá el mundo

- P,p → cambio de tipo de proyección: perspectiva / paralela.

Al igual que en el caso de los objetos existen diferentes transformaciones aplicables sobre la cámara.

- T,t → Traslaciones sobre la cámara (desactiva rotación y visión).
- R,r → Rotaciones sobre la cámara (desactiva traslación y visión).
- E,e → Cambiar volumen de visión (desactiva rotación y traslación).

Una vez seleccionado el modo de la cámara realizaremos las transformaciones con las siguientes teclas.

- UP → Trasladar +pitch; Rotar +pitch; Volumen +Y
- DOWN → Trasladar -pitch; Rotar - pitch; Volumen -Y
- RIGHT → Trasladar +yaw; Rotar +yaw Volumen +X
- LEFT → Trasladar -X; Rotar - en Y; Volumen -X
- AVPAG → Trasladar -Z; Rotar + en Z: Volumen +n,+f
- REPAG → Trasladar +Z; Rotar - en Z; Volumen -n,-f
- + → ZOOM +
- - → ZOOM -

Además de poder controlar la cámara seleccionada, se podrán generar nuevas cámaras y cambiar de una a otra.

- n → Añadir una cámara nueva.
- k → Cambiar de cámara.

Por último también se podrá ver el escenario desde el punto de vista del objeto seleccionado.

- K → Visualizar lo que ve el objeto seleccionado (cámara del objeto).

Cuando tenemos la cámara de objeto activada las transformaciones afectarán al objeto desde el que estamos viendo el mundo, y al transformar el objeto se visualizará el escenario desde la nueva posición y orientación del objeto.

3.3 Teclado: Iluminación

La iluminación es una prestación que por defecto estará activada, pero es posible desactivarla. Si está activada, podremos visualizar los objetos de manera suavizada usando los vectores normales de cada vértice o algo más plano usando los vectores normales de cada polígono.

- F9 → Activar/desactivar iluminación.
- F12 → Cambiar tipo de iluminación (flat/smooth).

En el modo iluminación podremos encender/apagar las luces diferentes luces que afectan al escenario, salvo la luz de ambiente que siempre estará activa.

- F1-F4 → Encender/apagar la fuente de luz correspondiente

Si queremos aplicar transformaciones sobre una luz, primero deberemos seleccionarla.

- 1 → Seleccionar luz del sol (por defecto).
- 2 → Seleccionar luz de la bombilla.
- 3 → Seleccionar foco del objeto.
- 4 → Seleccionar foco de la cámara.

Una vez seleccionado podremos aplicar las transformaciones sobre las luces. En función del tipo de transformación y del tipo de fuente de luz no se podrán realizar, algunas transformaciones no tendrán sentido.

- UP → Trasladar +Y; Escalar + Y; Rotar +X
- DOWN → Trasladar -Y; Escalar - Y; Rotar -X
- RIGHT → Trasladar +X; Escalar +X; Rotar +Y
- LEFT → Trasladar -X; Escalar -X; Rotar -Y
- AVPAG → Trasladar +Z; Escalar +Z; Rotar +Z
- REPAG → Trasladar -Z; Escalar - Z; Rotar -Z
- + → Incrementar ángulo de apertura (focos).
- - → Decrementar ángulo de apertura (focos).

El sol solamente puede rotar alrededor del objeto seleccionado, las bombillas podrán ser transformadas de cualquier manera y los focos dependerán de la posición y orientación del objeto.

4 Transformaciones a los objetos

Sabemos que los objetos se cargan desde un fichero y se guardan en una estructura en la memoria del programa. Dicha estructura contiene la información de los vértices y los polígonos que lo conforman, pero esa información solo es correcta cuando el objeto es cargado, en el momento que comenzamos a realizar transformaciones las coordenadas de los puntos ya no coinciden con las iniciales.

Para poder realizar transformaciones manteniendo los datos iniciales de los objetos se ha hecho uso de matrices de transformación. Cada objeto tendrá asociada una lista de matrices que representarán las transformaciones que ha sufrido. Al cargar el objeto se inicializará su matriz como la matriz identidad.

$$\begin{pmatrix} 1 & 0 & 0 & m \\ 0 & 1 & 0 & n \\ 0 & 0 & 1 & o \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Figure 4:
Matriz de traslación.

$$\begin{pmatrix} p & 0 & 0 & 0 \\ 0 & q & 0 & 0 \\ 0 & 0 & r & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Figure 5:
Matriz de escalado.

Las matrices de rotación dependen del eje en el que se rote el objeto.

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{pmatrix}$$

Figure 6:
Rotación respecto al eje X.

$$\begin{pmatrix} \cos\alpha & 0 & \sin\alpha \\ 0 & 1 & 0 \\ -\sin\alpha & 0 & \cos\alpha \end{pmatrix}$$

Figure 7:
Rotación respecto al eje Y.

$$\begin{pmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Figure 8: Rotación respecto al eje Z.

Para facilitar las operaciones entre matrices se utilizan los métodos integrados en OpenGL *glRotate(angle,x,y,z)*, *glTranslate(x,y,z)* y *glScale(x,y,z)*. OpenGL trabaja con dos matrices principales la matriz GL_PROJECTION y la matriz GL_MODELVIEW. Estos métodos aplicarán la operación a la matriz con la que se esté trabajando en ese momento, por lo que habrá que seguir los siguientes pasos para conseguir la matriz resultado:

1. Se indica que se va a trabajar con una de las matrices de OpenGL.
2. Se carga la matriz previa del objeto en la matriz MODELVIEW.
3. Se llama a una de las funciones de transformación, que se ejecutará sobre la matriz que ya está en MODELVIEW.
4. Se obtiene la matriz resultado desde el MODELVIEW.
5. Se guarda la matriz resultado en una estructura de matriz, que tendrá un puntero que apunte a la matriz anterior del objeto.

- Si la transformación se hace en el sistema de referencia se debe premultiplicar la matriz de transformación por lo que habrá que cambiar el orden de los pasos anteriores.
- Las matrices de objeto previas se guardan en una lista ligada, de esta manera se pueden deshacer los cambios de las transformaciones realizadas a los objetos.

Este es un ejemplo de implementación de una transformación en C.

```
GLfloat matriz_transformada[16];
matrix *m = _selected_object->matrixptr;
glMatrixMode(GL_MODELVIEW);

if(referencia == LOCAL){
  glLoadMatrixf(m->values);
}
else {
  glLoadIdentity();
}

if (rotar){
  glRotatef(10,0,-1,0);
}
else if (trasladar){
  glTranslatef(-1,0,0);
}
else if (escalar){
  glScalef(0.9,1,1);
}

if (referencia == GLOBAL){
  glMultMatrixf(m->values);
}
glGetFloatv(GL_MODELVIEW_MATRIX,matriz_rotada);

matrix *sig_matriz = malloc(sizeof (matrix));
for(int i = 0; i < 16; i++){
  sig_matriz->values[i] = matriz_rotada[i];
}
sig_matriz->sigptr = m;
_selected_object->matrixptr = sig_matriz;
```

Una vez obtenida la matriz del objeto transformado, en la función de dibujado la deberemos cargar en la matriz MODELVIEW antes de dibujarlo, y de esta manera se aplicarán al objeto las transformaciones correspondientes a la matriz.

Este es el bucle que dibuja los objetos transformados.

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

while (aux_obj != 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);
        }

        /* We load the matrix object to the MODELVIEW matrix*/
        glLoadMatrixf(aux_obj->matrixptr->values);

        /* Draw the object; for each face create a new polygon with the
        corresponding vertices */
        for (f = 0; f < aux_obj->num_faces; f++) {
            glBegin(GL_POLYGON);
            for (v = 0; v < aux_obj->face_table[f].num_vertices; v++) {
                v_index = aux_obj->face_table[f].vertex_table[v];
                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();
        }
        aux_obj = aux_obj->next;
}
```

5 Cambios de sistema de referencia

Una vez que tenemos los objetos transformados debemos tener en cuenta que en función de la posición de la cámara el sistema de referencia variará y por tanto habrá que tenerlo en cuenta.

Para ello crearemos una estructura 'camara' que tendrá los mismos elementos que la estructura objeto, pero además de la matriz de objeto tendrá una matriz de cambio sistema de referencia.

$$\begin{pmatrix} x_c & y_c & z_c & e \\ | & | & | & | \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Figure 9:
Matriz de objeto

$$\begin{pmatrix} x_c & \text{——} & -e \cdot x_c \\ y_c & \text{——} & -e \cdot y_c \\ z_c & \text{——} & -e \cdot z_c \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Figure 10:
Matriz de CSR

Para obtener una matriz de cambio de sistema de referencia inicial, utilizamos de nuevo la matriz MODELVIEW de OpenGL. Cargaremos la matriz identidad y llamaremos a la función *gluLookAt(v1,v2,v3)*, que recibe 9 parámetros, o más bien tres vectores con tres elementos cada uno.

- v1: los primeros tres parámetros indican la posición de la cámara en el espacio (x,y,z).
- v2: los tres siguientes marcan a que punto estará mirando la cámara (x,y,z).
- v3: los últimos tres marcan el grado de inclinación de la cámara, por defecto (0,1,0).

Una vez obtenida la matriz CSR se puede obtener también la matriz de objeto a partir de ella. Para ello podemos usar la igualdad $M_{CSR} = {M_{OBJ}}^{-1}$ o también se puede hacer asignando valores en diferentes posiciones, ya que como se puede ver en las figuras 9 y 10 los valores de una matriz a otra son prácticamente idénticos, solamente variando su posición.

Este es un script de C que calcula la matriz de objeto en función de una matriz CSR y el vector e.

```
void get_matriz_objeto(GLfloat* m1, vector3* e,GLfloat* m2){
    for(int i = 0; i < 3; i++){
      for (int j = 0; j < 3;j++){
          m2[i+4*j] = m1[4*i+j];
      }
    }
    m2[3] = 0;
    m2[7] = 0;
    m2[11] = 0;
    m2[12] = e->x;
    m2[13] = e->y;
    m2[14] = e->z;
    m2[15] = 1;
}
```

Una vez teniendo las dos matrices iniciales podremos dibujar los objetos desde el punto de vista de la cámara, para ello deberemos cargar la matriz de CSR de la cámara en la matriz MODELVIEW antes del bucle que dibuja los objetos y después multiplicarla por la matriz de cada objeto.

Para implementar la opción de visualizar el mundo desde el punto de vista deberemos añadir una opción que si está activa cargará en el MODELVIEW la matriz CSR del objeto, que se calculara en una función a partir de la matriz de objeto.

```
glMatrixMode(GL_MODELVIEW);
if(modo == CAMARAOBJETO){
    matrix *aux_matrix = malloc( sizeof(matrix) );
    get_matriz_csr(_selected_object->matrixptr->values,aux_matrix->values);
    glLoadMatrixf(aux_matrix->values);
}
else{
    glLoadMatrixf(_selected_camera->matrixcsrptr->values);
}

while (aux_obj != 0) {
    (...)
}
```

La matriz objeto de la cámara nos servirá para dibujar las cámaras, que serán visibles si creamos más de una. Para ello haremos otro bucle que recorra la lista de las cámaras y las dibuje.

A la hora de realizar transformaciones deberemos transformar las dos matrices, para que los cambios se reflejen tanto en la vista desde la cámara como en el objeto que verán las otras cámaras. Para saber que transformación debemos hacer a la matriz de CSR para que sea equivalente partimos de la igualdad: $M_{CSR} = {M_{OBJ}}^{-1}$.

$$M'_{OBJ} = M_{OBJ} \cdot M_{Transf.}$$

$$M'_{CSR} = (M'_{OBJ})^{-1} = (M_{OBJ} \cdot M_{Transf.})^{-1} = {M_{Transf.}}^{-1} \cdot {M_{OBJ}}^{-1}$$

$$M'_{CSR} = {M_{Transf.}}^{-1} \cdot M_{CSR}$$

Por último para alternar la perspectiva de la aplicación usaremos la matriz de OpenGL GL_PROJECTION.

Antes de los bucles de dibujado estableceremos el modo de perspectiva. Para ello llamaremos al método *glMatrixMode(GL_PROJECTION)* que indicará que cargaremos matrices en la matriz de proyección. Una vez hecho esto en función de la vista que queramos tener llamaremos al método *glFrustum()* para vista en perspectiva y al método *glOrtho()*.

Aquí está el fragmento del programa que establece la vista.

```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();

if(proyection){
  glFrustum(_selected_camera->proyection[0],_selected_camera->proyection[1],
  _selected_camera->proyection[2],_selected_camera->proyection[3],
  _selected_camera->proyection[4],_selected_camera->proyection[5]);
}
else{
  glOrtho(_selected_camera->proyection[0],_selected_camera->proyection[1],
  _selected_camera->proyection[2],_selected_camera->proyection[3],
  _selected_camera->proyection[4],_selected_camera->proyection[5]);
}
```

Los parámetros que se le pasan a los métodos son los límites de visión que tendrá la cámara en cada eje.

- X: Left/Right.
- Y: Bottom/Top.
- Z: Near/Far.

Los parámetros se cargan desde la cámara ya que son independientes, debido a que están relacionados con el zoom que tiene cada una de ellas.

6 Iluminación del escenario

Para activar la luz en la aplicación se utiliza la función *glEnable()* sobre la constante de OpenGL GL_LIGHTING. Además para encender cada una de las luces se deberá ejecutar la misma función sobre las constantes que se refieren a cada luz, en nuestro caso desde GL_LIGHT0 hasta GL_LIGHT4. De esta manera las luz estará activada y las luces encendidas al iniciar la aplicación.

Los parámetros para cada luz se guardan en una lista de estructuras de tipo light. Cada estructura tiene un vector para guardar los parámetros de su luz de ambiente, especular y de difusión, además también se tiene su posición inicial y una matriz para manejar las transformaciones, igual que con los objetos y las cámaras. En el caso de los focos también tendremos su nivel de apertura.

Los parámetros inicializados se cargan en la función de dibujado tras haber cargado en la matriz MODELVIEW la matriz correspondiente a la luz.

1. En el caso del sol y la bombilla tendrán una matriz propia que se cargará antes del bucle de dibujado de objetos y cámaras.
2. El foco de la cámara se cargará tras haber cargado la matriz CSR de la cámara en el bucle de dibujado.
3. Por último los parámetros del foco del objeto se cargarán después de haber multiplicado la matriz de objeto del objeto seleccionado a la matriz CSR de la cámara en MODELVIEW.

Para reflejar como afecta cada luz a los objetos del escenario se usan los vectores normales de los objetos. Los vectores normales se calculan para cada

polígono y cada vértice al cargar el objeto.

Esta es la implementación en C para calcular los vectores normales.

```
/*Initialize vertex normal vectors*/
for(int i = 0; i < object_ptr->num_vertices; i++){
  object_ptr->vertex_table[i].normal.x = 0;
  object_ptr->vertex_table[i].normal.y = 0;
  object_ptr->vertex_table[i].normal.z = 0;
}
/*Calculate normal vector of every polygon*/
for (int i = 0; i < object_ptr->num_faces; i++){
  face *act = &object_ptr->face_table[i];

  vertex p1 = object_ptr->vertex_table[act->vertex_table[0]];
  vertex p2 = object_ptr->vertex_table[act->vertex_table[1]];
  vertex p3 = object_ptr->vertex_table[act->vertex_table[2]];

  vector3 v1, v2;

  v1.x = p2.coord.x - p1.coord.x;
  v1.y = p2.coord.y - p1.coord.y;
  v1.z = p2.coord.z - p1.coord.z;

  v2.x = p3.coord.x - p1.coord.x;
  v2.y = p3.coord.y - p1.coord.y;
  v2.z = p3.coord.z - p1.coord.z;

  GLfloat x,y,z;
  x = v1.y * v2.z - v2.y * v1.z;
  y = -(v1.x * v2.z - v2.x * v1.z);
  z = v1.x * v2.y - v2.x * v1.y;

  for (int j = 0; j < act->num_vertices; j++){
    vertex *v = &object_ptr->vertex_table[act->vertex_table[j]];
    v->normal.x = v->normal.x + x;
    v->normal.y = v->normal.y + y;
    v->normal.z = v->normal.z + z;
  }

  GLfloat length = sqrt(pow(x,2) + pow(y,2) + pow(z,2));
  if(length != 0){
    x = x / length;
    y = y / length;
    z = z / length;
  }

  act->normal.x = x;
  act->normal.y = y;
  act->normal.z = z;
}

/*Make the vertex normal vectors unitary*/
for(int i = 0; i < object_ptr->num_vertices; i++){
  vertex *act = &object_ptr->vertex_table[i];
  GLfloat length = sqrt(pow(act->normal.x,2) + pow(act->normal.y,2)
  + pow(act->normal.z,2));
  act->normal.x = act->normal.x / length;
  act->normal.y = act->normal.y / length;
  act->normal.z = act->normal.z / length;
}
```

Una vez calculados los vectores normales deberemos llamar a la función *glNormal3f()* en el bucle de dibujado, que recibirá las coordenadas de los vectores normales y generará sombras y brillos en función de la orientación del objeto y la posición de la luz. La función se deberá llamar una vez por cada polígono con su normal si estamos visualizando el objeto en modo flat y una vez por cada vértice si estamos en modo smooth.

Además de para calcular la incidencia de la luz en el objeto, los vectores normales se utilizan para ahorrar coste computacional a la GPU a la hora de dibujar los polígonos, debido a que gracias a ellos podemos determinar que polígonos son traseros y cuales delanteros, en función del ángulo que tengan con el vector que va hacia la cámara.

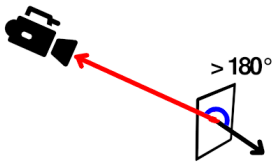

Figure 11:
Poligono trasero

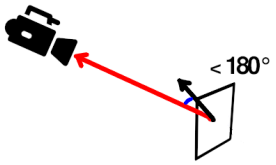

Figure 12:
Polígono delantero

Para saber si el ángulo entre el vector normal del polígono y el vector que va hacia la cámara tiene un ángulo mayor de 180º se utiliza el producto escalar.

Esta es la implementación en C para determinar si el polígono es delantero o no.

```
int poligono_delantero(object3d *o,face f){
  /*First we get the camera location*/
  vector3 e;
  if (modo == CAMARAOBJETO){
    e.x = _selected_object->matrixptr->values[12];
    e.y = _selected_object->matrixptr->values[13];
    e.z = _selected_object->matrixptr->values[14];
  }
  else{
    e.x = _selected_camera->matrixobjptr->values[12];
    e.y = _selected_camera->matrixobjptr->values[13];
    e.z = _selected_camera->matrixobjptr->values[14];
  }

  /*Then, we pass the camera location to the object reference system*/
  GLfloat csr_objeto[16];
  get_matriz_csr(o->matrixptr->values,csr_objeto);
  e.x = e.x*csr_objeto[0] + e.y*csr_objeto[4] + e.z*csr_objeto[8]
  + csr_objeto[12];
  e.y = e.x*csr_objeto[1] + e.y*csr_objeto[5] + e.z*csr_objeto[9]
  + csr_objeto[13];
  e.z = e.x*csr_objeto[2] + e.y*csr_objeto[6] + e.z*csr_objeto[10]
  + csr_objeto[14];

  /*Now we calculate the vector between the camera and the face*/
  vector3 v;
  vertex p_obj = o->vertex_table[f.vertex_table[0]];
  v.x = p_obj.coord.x - e.x;
  v.y = p_obj.coord.y - e.y;
  v.z = p_obj.coord.z - e.z;

  /*Finally we calculate the escalar product of the normal and the v vector
  to see the angle between them*/
  GLfloat escalar = v.x * f.normal.x + v.y * f.normal.y + v.z * f.normal.z;

  if(escalar > 0){
    return 0;
  }
  else{
    return 1;
  }
}
```

Por último, cada material tiene un comportamiento u otro ante la luz, para comprobar las características de los diferentes materiales se puede cambiar el objeto de los materiales pulando la tecla M desde el modo objeto. Existen 4 tipos de materiales disponibles en la aplicación;

1. Esmeralda
2. Rubí
3. Plástico azul
4. Goma amarilla

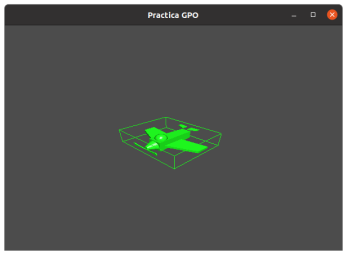

Figure 13:
Esmeralda

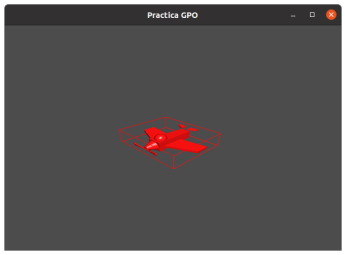

Figure 14:
Rubi

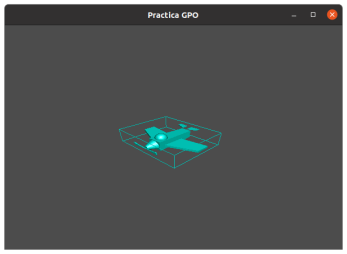

Figure 15:
Plástico azul

Figure 16:
Goma amarilla

7 Capturas de la aplicación

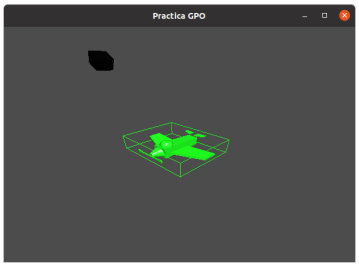

Figure 17:
Escenario con un objeto y una cámara

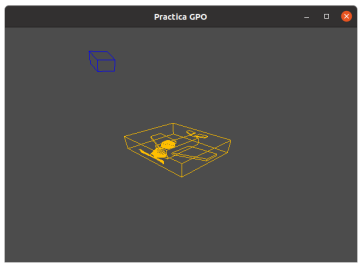

Figure 18:
Aplicación sin iluminación

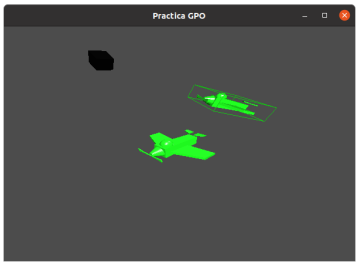

Figure 19:
Varios objetos cargados en el escenario

Figure 20:
Punto de vista del objeto seleccionado

8 Conclusiones

En este proyecto se han visto todas las posibilidades que ofrece OpenGL para renderizar objetos sobre el escenario, desde distintos puntos de vista y con varias fuentes de iluminación.

Además, también se entienden los conceptos matemáticos utilizados por los programas que se utilizan para la creación y modelado de escenarios que se usan en el desarrollo de videojuego, como por ejemplo Unity.