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. **Iluminacion.**

   (a) **Explicación.**

      i. **Luz.**

      ii. **Material.**

      iii. **Fuentes de luz.**

   (b) **Vectores normales.**

   (c) **Creación de luces.**

   (d) **Modificación de luces.**

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**.
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 realizara 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, dandole a la "f" antes, nosotros recomendamos el uso del siguiente comando para ahorrarnos el tener que escribir el path cada vez que cargamos un objeto: **cp objects/abioia.obj ./a** donde objects/abioia.obj es el path al objeto deseado. De esta forma para cargar un objeto bastará con escribir únicamente "a".
Una vez cargado el objeto tenemos acceso completo a 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 iluminacion 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.

ii. **Rotación.**
iii. **Traslación.**
iv. **Modo Análisis/Vuelo.**
v. **Cambio de tipo de proyección.**

3. **Iluminación**

(a) **Explicación.** A continuación se explicará el funcionamiento de la luz y los materiales en OpenGL. Para empezar es importante saber que OpenGL usa la composición de colores RGB (RedGreenBlue). Esto significa que los colores primarios son el rojo, el verde y el azul y el resto de colores los generará con una mezcla de estos. El blanco sería la unión perfecta de los tres colores en la misma proporción y el negro la ausencia de estos.

i. **Luz.** Existen 4 tipos de luz en OpenGL, estos son, la luz ambiental, la difusa, la especular y la emisiva.
   A. Luz ambiental: Esta es la luz que ha sido tan esparcida despues de rebotar tanto que es imposible determinar su origen. Viene de todas direcciones, es simplemente la luz que hay en el ambiente que no proviene de un origen concreto.
   Cuando la luz ambiental golpea una superficie, esta se esparce de forma equitativa en todas direcciones.
   B. Luz difusa: Esta luz, al contrario que la ambiental, proviene de una dirección concreta, pero de forma difusa. La luz del sol sería un buen ejemplo de una luz difusa.
   Esta luz es más brillante si golpea directamente una superficie que si lo hiciera de forma parcial. Al igual que la ambiental al golpear un objeto se esparce equitativamente en el espacio.

C. Luz especular: Esta es la luz que proviene de una dirección particular y tiende a rebotar hacia otra concreta dirección. Piensa en ella como el brillo de la superficie, dependiendo del brillo rebotará con más o menos intensidad y hacia una dirección concreta. Un espejo o un metal, por ejemplo, tendran mucha luz especular ya que reflejan muy bién la luz.

Simplificando, podríamos decir que la luz está compuesta por la intensidad de luz, roja, verde o azul que emite. Cabe destacar que estos 3 números toman valores entre 0 y 1.

ii. **Material.** Un material se verá de X color dependiendo de la cantidad y tipo de luz que emita de los rayos de luz que lo golpean. Si simplificamos, la luz está compuesta por 3 valores, que son la cantidad de luz roja, verde o azul que emiten. El material (simplificando) también estará compuesto por estos tres valores, pero en este caso simbolizan la cantidad de luz que emitirá en función de la cantidad de rayos de luz que lo golpeen. Esto quiere decir que si iluminamos con una luz blanca a un material que queremos que sea rojo, el valor de luz roja del material deberá de ser mayor al de los otros dos valores. Si la luz es blanca (Rojo=1,Verde=1,Azul=1) y el material tiene estos valores, Rojo=1,Verde=0,Azul=0, el objeto se verá rojo.

Cabe destacar que esto es una explicación simplificada. Al igual que la luz, los materiales también tienen distinción entre el tipo de luz que reflejan (emisiva, especular, difusa y ambiental).

iii. **Fuentes de luz.** Nosotros trabajamos con 3 tipos diferentes de fuentes de luz, soles, bombillas y focos.

A. Soles. El sol es una fuente de luz que proviene desde una dirección a una distancia infinita. Sus rayos de luz vienen en paralelo.

Este tipo de fuente no puede ser trasladado ya que no tiene una posición concreta pero si que puede ser rotado para modificar la dirección en la que viene.

B. Bombillas. Este tipo de fuente de luz tiene una posición fija en el espacio y ilumina en todas direcciones por igual (aunque esto podría ser modificado a placer).

Puede ser trasladada porque tiene una posición finita, pero no tiene sentido rotarla porque ilumina a todas di-

recciones por igual.

C. Focos. Al igual que la bombilla, el foco tiene una posición fija en el mundo pero no ilumina en todas direcciones por igual, ilumina en una dirección concreta y de forma limitada. Un foco, al igual que una linterna real, ilumina en una dirección y con un haz de luz de forma cónica, este cono va definido por el ángulo de apertura que le especifiquemos, cuanto mayor sea, mas extenso será la base del cono y iluminará mas cosas cercanas a él, cuanto mas pequeño, más le costará iluminar objetos cercanos y mas concentrada estará la luz en el punto al que mira.
El foco puede ser tanto rotado, como trasladado sin problema alguno.

(b) **Vectores normales.** Un vector normal es un vector perpendicular a la superficie del polígono al que pertenece, su distancia es de 1. Con este vector podemos calcular la cantidad de luz que llegará al objeto, dependiendo de como se usen tendremos 2 formas de iluminación distintas (FLAT o SMOOTH).

i. FLAT. Esta forma de iluminar usa únicamente los vectores normales de los polígonos del objeto, asi que cada superficie estará compuesta por un único tono de iluminación.

ii. SMOOTH. Esta forma de iluminar usa tanto los vectores normales de los polígonos como los de los vértices. Al usar ambos vectores, el objeto tendra sombras más gradientes, no terminarán de forma tan abrupta como con el modo FLAT.

Para almacenar los vectores normales de los vértices y de las caras hemos tenido que modificar la estructura de las caras y los vértices de los objetos, quedando de la siguiente manera.

```
typedef struct {
    point3 coord;
    GLint num\_faces;
    vector3 normalV;      /*Vector normal del vertice*/
} vertex;

typedef struct {
    GLint num_vertices;
    vector3 normalF;      /*Vector normal del poligono*/
    GLint *vertex\_table;
} face;
```

Para calcular el vector normal de una cara necesitamos tres de sus puntos (a,b y c), una vez obtenidos obtenemos 2 vec-

tores (vector1 = 3-1,vector2 = 2-1). Una vez obtenidos estos dos vectores realizaremos su producto vectorial para obtener el vector perpendicular a la cara.
Ahora solo queda normalizar el vector, es decir, hacer que su distancia sea de 1. Esto lo haremos calculando su distancia (raiz cuadrada de la suma de cuadrados) y dividiendo sus coordenadas por esta recién calculada distancia. De esta forma ya tenemos el vector final de ese polígono concreto.
Para calcular el vector normal de los vértices, primero les sumaremos los vectores normales de todas las superficies de las que formen parte y una vez hecho esto, normalizaremos dichos vectores, al igual que hemos explciado anteriormente.
Cabe destacar, que en el código estos cálculos tendrán que ser realizados una única vez al cargar los objetos.

```
/*Esta parte de codigo es muy extensa*/
/*por lo que si desea verlo debera de*/
/*dirigirese al codigo donde lo hemos*/
/*explicado en detalle, esta situado */
/*en load_obj_joseba.c al cerrar el  */
/*archivo tras la primera pasada y   */
/*antes de la segunda.               */
```

Ahora ya tenemos los vectores normales calculados, pero nos falta decirle a OpenGL cuando y cual va a ser el vector normal de los vértices. Esto lo haremos a la hora de dibujar el objeto, es decir, en el 'display.c', justo antes de que dibuje los vértices.

```
/* Draw the object; for each face create a new polygon
with the corresponding vertices */
for (f = 0; f < aux_obj->num_faces; f++) {
    glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
    glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient);
    glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
    glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);
    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];
        //Decimos cual es el vector normal del poligono
        glNormal3d(aux_obj->vertex_table[aux_obj->
        face_table[f].vertex_table[v]].normalV.x,
            aux_obj->vertex_table[aux_obj->
            face_table[f].vertex_table[v]].normalV.y,
            aux_obj->vertex_table[aux_obj->
            face_table[f].vertex_table[v]].normalV.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);

    }
```

Con esto ya habriamos terminado de implementar los vectores normales que son necesarios a la hora de especificar el tipo de

iluminación que queremos (flat o smooth).
Cabe destacar que en este mismo bucle le asignamos el material deseado a las caras del objeto (también podriamos asignárselo únicamente al objeto). En nuestro caso, el material de todos nuestros objetos oro (esto puede ser cambiado a voluntad modificando los valores utilizados).

(c) **Creación de luces.** Nuestra aplicación dispone de 4 fuentes de luz. 1:Un sol, 2: Una bombilla, 3:Un foco en el objeto y 4:Un foco externo. Las luces han sido creadas en el 'display.c'. La iluminación se apaga y enciende pulsando la tecla 'f9', pero si deseamos apagar/encender una fuente de luz en concreto tendremos que pulsar las teclas del 'f1' al 'f4', cada una apaga/enciende la luz correspondiente a su número.
Cabe destacar la diferencia entre una fuente de luz posicional y no. El sol, por ejemplo, es una fuente no posicional, esto significa que no tiene una posición fija en el espacio, esto se define con la siguiente llamada: 'glLightfv(GL_LIGHT0, GL_POSITION, sun_position)' donde el vector "sun_position" está compuesto por 4 elementos, si el cuarto elemento es un 0, significa que la fuente será un sol y sus 3 valores restantes definiran de donde proviene la luz. En caso de que el cuarto valor valga 1, los tres primero definirán la posición en el mundo de la fuente.
Sabiendo esto, solo queda aclarar que nuestra bombilla está en la posición 1,1,1 y ilumina en todas direcciones. Nuestro foco relacionado al objeto se encuentra en la posición del objeto seleccionado, apunta hacia el mismo objeto y el ángulo de su foco es de 25º. Y el foco externo está colocado en la posicion -1,-1,-1, apunta a 0,0,-1 y su ángulo de foco es de 15º.
Nuestra aplicación dispone de una variable 'iluminacion' que indica si la iluminación general está activada (iluminacion=1) o no (iluminacion=0). También existe un array de integers de longitud igual al número de luces (4), este array representará si esa luz en concreto está apagada o no y también será la forma que tendremos de seleccionar una luz concreta a través de un puntero, esto se explicará en el siguiente apartado.
Cabe destacar que paa desactivar la iluminación basta con desactivarla en general, no hace falta desactivar todas las fuentes de luz una a una.

```
case GLUT_KEY_F9:
    if(iluminacion==0) {
        printf("Usted acaba de activar el modo de iluminacion.\n");
```

```
        iluminacion=1;
        luces[0]=1; luces[1]=1; luces[2]=1; luces[3]=1;
        glEnable(GL_LIGHTING);
        glEnable(GL_LIGHT0);
        glEnable(GL_LIGHT1);
        glEnable(GL_LIGHT2);
        glEnable(GL_LIGHT3);
        glEnable(GL_DEPTH_TEST);
    }
    else{
        iluminacion=0;
        glDisable(GL_LIGHTING);
        luces[0]=0; luces[1]=0; luces[2]=0; luces[3]=0;
    }
break;

case GLUT_KEY_F12:
    if(iluminacion==1) {
        iluminacion=2;
        printf("Usted acaba de activar el modo SMOOTH.\n");
        glShadeModel(GL_SMOOTH);
    }
    else{
        iluminacion=1;
        printf("Usted acaba de activar el modo FLAT.\n");
        glShadeModel(GL_FLAT);
    }
break;

case GLUT_KEY_F1:
    if(iluminacion!=0) {
        if(luces[0]==1) {//Luz 0 activada, la desactivamos
            luces[0]=0;
            printf("Se ha DESACTIVADO la fuente de luz 0,
            el sol.\n");
            glDisable(GL_LIGHT0);
        }
        else{//Luz 0 desactivada, la activamos
            luces[0]=1;
            printf("Se ha ACTIVADO la fuente de luz 0,
            el sol.\n");
            glEnable(GL_LIGHT0);
        }
    }
    else {
        printf("Activa la iluminacion para encender/apagar
        fuentes de luz.\n");
    }
break;
//f3 y f4 son identicos a f1 o f2
...
...
```

(d) **Modificación de luces.** Nuestra aplicación solo permite trasladar fuentes de luz y aumentar o reducir el ángulo de apertura de un foco. Dado las características de las fuentes, solo seremos capaces de trasladar la bombilla y el foco externo, pero podremos modificar el ángulo de apertura de tanto el foco del objeto como el externo.

Para seleccionar la fuente que deseamos modificar deberemos de pulsar las teclas del '1' al '4', cada una seleccionará la fuente correspondiente a su número. Aunque solo podremos trasladar la 2 y 4 y modificar el ángulo de apertura de los focos 3 y 4.
Este es un fragmento de la función 'traslate(a,b,c)' que hemos creado para trasladar los objetos, la variable 'moverluces' representa si debemos de modificar una luz o no. 'a','b' y 'c' son los valores en los que nos trasladamos, por lo que únicamente deberemos de volver a designar la posición de la luz modificándola con los valores anteriormente mencionados.
Para aumentar/reducir el ángulo de apertura de un foco usaremos un proceso simiar, simplemente volveremos a definir ese ángulo aumentandolo o reduciéndolo en una proporcion del 10 por ciento.

```
//Funcion traslate(a,b,c)
...
if (moverluces==1)
  {
    if(luzseleccionada==luces[1]){
      printf("Trasladamos la bombilla\n");
      pbombilla[0]+=a;
      pbombilla[1]+=b;
      pbombilla[2]+=c;
      glLightfv(GL_LIGHT1, GL_POSITION, pbombilla);
      pbombilla[1],pbombilla[2]);
    }
    else if(luzseleccionada==luces[3]){
      printf("Trasladamos el foco externo\n");
      pfoco2[0]+=a;
      pfoco2[1]+=b;
      pfoco2[2]+=c;
      glLightfv(GL_LIGHT3, GL_POSITION, pfoco2);
      pfoco2[1],pfoco2[2]);
    }
    else {
      printf("La fuente de luz seleccionada no
      puede ser traladada.\n");
    }
...
//-------------------------------------------------------
//Funcion de teclado
...
case '-':
      printf("Acaba de pulsar '-'\n");
      if(iluminacion=!0 && luces[2]==1
      && luzseleccionada==luces[2]){
        glLightf(GL_LIGHT2, GL_SPOT_CUTOFF, angulofoco*0.9);
        angulofoco=angulofoco*0.9;
      }
      else if(iluminacion=!0 && luces[3]==1
      && luzseleccionada==luces[3]){
        glLightf(GL_LIGHT3, GL_SPOT_CUTOFF, angulofoco2*0.9);
        angulofoco2=angulofoco2*0.9;
      }
      else {
        printf("No pasa nada\n");
```

```
      }
      ...

case '+':
      printf("Acaba de pulsar '+'\n");
      if(iluminacion=!0 && luces[2]==1
      && luzseleccionada==luces[2]){
        glLightf(GL_LIGHT2, GL_SPOT_CUTOFF, angulofoco*1.1);
        angulofoco=angulofoco*1.1;
      }
      else if(iluminacion=!0 && luces[3]==1
      && luzseleccionada==luces[3]){
        glLightf(GL_LIGHT3, GL_SPOT_CUTOFF, angulofoco2*1.1);
        angulofoco2=angulofoco2*1.1;
      }
      else {
        printf("No pasa nada\n");
      }
      ...
```

Cabe destacar que tanto en el caso de pulsar '-' o '+' existe más código, pero al no tener nada que ver con esta parte, hemos decidido no representarlo aqui.