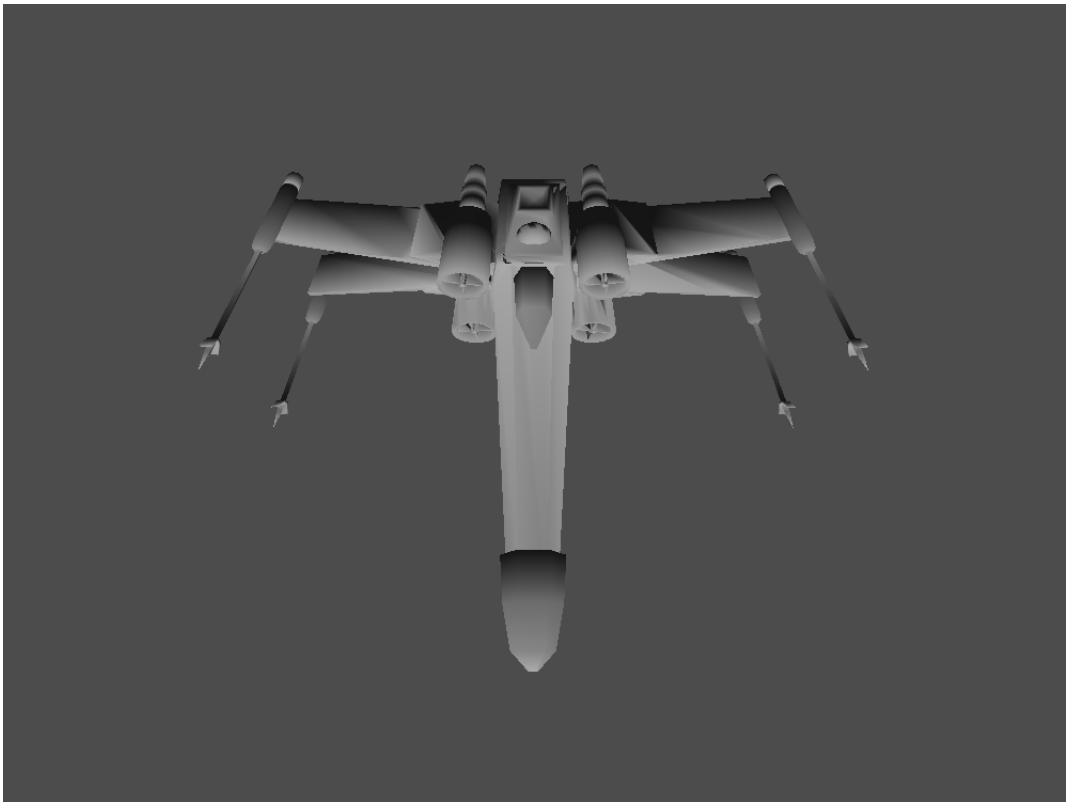
Entorno virtual

Jon Perez Etxebarria y Yara Diaz de Cerio Arzamendi

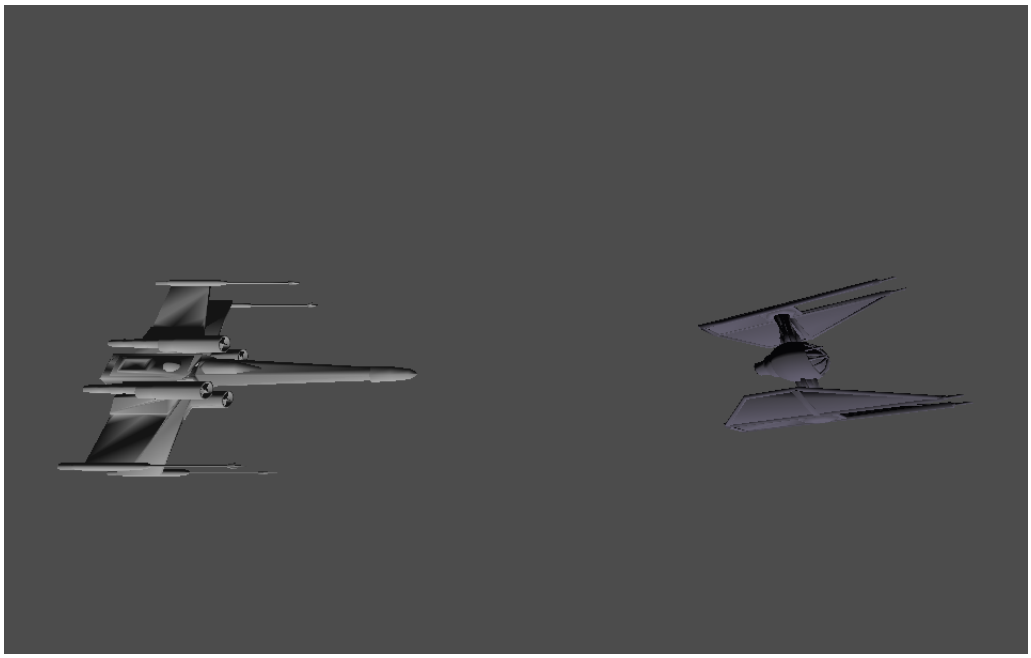
January 8, 2020

Aplicacion de visualizado de objetos en un entorno virtual.



1 Objetivos de nuestra aplicación

Con esta aplicación pretendemos visualizar objetos en un entorno virtual, ademas de transformarlos según nuestras necesidades. La observación detallada y desde distintos ángulos es un foco importante de lo que pretendemos conseguir por eso hemos implementado un modulo de cámaras para observar el objeto desde el angulo que al usuario mas le venga en gana o incluso desde mas de un angulo o mas de un objeto. Por ultimo pretendemos que los objetos simulen en la medida de lo posible la realidad por eso hemos añadido texturas que pueda observarse de manera mas realista como seria el modelo real.



2 Descripción

En esta aplicación se podrán cargar y visualizar modelos 3D del formato '.obj' . Ademas también podrán aplicarse a estos distintas transformaciones: desplazarlos, rotarlos y escalarlos. También cuenta con una cámara inicializada desde el comienzo y con la posibilidad de la posibilidad de crear cámaras adicionales. Estas cámaras podrán moverse libremente o fijarse en el objeto para un análisis mas preciso del objeto. El ultimo modulo implementado es el de luces y texturas. Este te permite elegir una textura para el objeto y activar y desactivar las luces para visualizarlo de esta forma, así evitando

ver un objeto translucido. Ademas contiene dos tipos de iluminación para mayor elección del usuario.

3 Manual de usuario

Para utilizar la aplicación será necesario tener las librerías de OpenGl y sus dependencias. Para iniciar la aplicación se deberá abrir la terminal de linux en la carpeta de la aplicación, escribir "./visualizacion3D" y pulsar la tecla enter. Una vez hecho, se abrirá una ventana, que será la que muestra la imagen de los objetos que cargaremos más adelante y se mostrará un menú en la terminal con las siguientes opciones:

```
FUNTZIO NAGUSIAK
<?>             Laguntza hau bistaratu
<ESC>           Programatik irten
<F>             Objektua bat kargatu
<TAB>           Kargaturiko objektuen artean bat hautatu
<DEL>           Hautatutako objektua ezabatu
<CTRL + ->      Bistaratze-eremua handitu
<CTRL + +>      Bistaratze-eremua txikitu
LUCES:
<F9>            Activar/desactivar la iluminacion
<F1-F8>         Encender/apagar la fuente correspondiente
<1-8            Seleccionar la fuente correspondiente
<0>             Asignar tipo de fuente a la luz seleccionada
<Insert>                Cambiar de tipo de iluminacion: FLAT/SMOOTH
<K, k>          Cambiar a modo camara
<O, o>          Cambiar a modo objeto
<A,a>           Aplicar transformaciones a las fuentes de luz


CONTROLES DE LA CAMARA

Las transformaciones se realizan con las flechas y F11, F12
<M, m>          Trasladar
<B, b>          Rotar
<T,t>           Cambiar volumen de vision

<c>             Cambiar de camara
<C>             Visualizar lo que ve el objeto seleccionado
<G, g>          Modo analisis
<L, l>          Modo vuelo
<P, p>          Cambiar tipo de proyeccion: perspectiva/paralela
<P, p>          Cambiar tipo de proyeccion: perspectiva/paralela
```

Para usar esta aplicación hacen falta tanto el terminal como la ventana emergente. La ventana sera la que nos muestre el entorno virtual y para recibir las ordenes mediante las teclas que se listaran a continuación mientras que la consola servirá para pedirle al usuario información o detalles en ciertas ocasiones como por ejemplo donde se desea general un foco o que fichero deberá cargar para mostrar su objeto.

Comandos:

- '?': Abrir el menú de la imagen anterior.

```
case '?':
    print_help();
    if(mCamara == 0 && mLuces == 0) {
        printHelpObjetos();
    } else if(mCamara == 1 && mLuces == 0) {
        printHelpCamara();
    } else {
        printHelpIlum();
    }
    break;
```

- 'ESC': Cerrar aplicación.

```
case 27: /* <ESC> */
    exit(0);
    break;
```

- 'f/F' cargar objeto.

```
case 'f':
case 'F':
    /*Ask for file*/
    printf("%s", KG_MSSG_SELECT_FILE);
    scanf("%s", fname);
    /*Allocate memory for the structure and read the file*/
    auxiliar_object = (object3d *) malloc(sizeof (object3d));
    read = read_wavefront(fname, auxiliar_object);
    switch (read) {
    /*Errors in the reading*/
    case 1:
        printf("%s: %s\n", fname, KG_MSSG_FILENOTFOUND);
        break;
    case 2:
        printf("%s: %s\n", fname, KG_MSSG_INVALIDFILE);
        break;
    case 3:
        printf("%s: %s\n", fname, KG_MSSG_EMPTYFILE);
        break;
    /*Read OK*/
    case 0:

        auxiliar_object->mtrxPTR= (lMatrix *)malloc(sizeof(lMatrix));
        auxiliar_object->mtrxPTR->next = 0;


        glMatrixMode(GL_MODELVIEW_MATRIX);
        glLoadIdentity();
        glGetDoublev(GL_MODELVIEW_MATRIX,auxiliar_object->mtrxPTR->mat);

       vNoarmlesObjeto(auxiliar_object);
       printf("Vectores normales calculados \n");

       //Por defecto sera del material obsidiana
       auxiliar_object->matPTR = lMateriales[3];
       printf("Material asignado \n");

       //Creamos el foco del objeto
       crearFoco(auxiliar_object);
       lLuces[2] = auxiliar_object->focoPTR;
       printf("Foco asociado al objeto creado \n");

       //Crear la camara asociada al objeto
       auxiliar_object->miCamara= (camara *)malloc(sizeof(camara));
       crearCamara(auxiliar_object->miCamara,
       auxiliar_object->mtrxPTR->mat[12],auxiliar_object->mtrxPTR->mat[13],auxiliar_object->mtrxPTR->mat[14],
       auxiliar_object->mtrxPTR->mat[12]+auxiliar_object->mtrxPTR->mat[8],auxiliar_object->mtrxPTR->mat[13]
        +auxiliar_object->mtrxPTR->mat[9],auxiliar_object->mtrxPTR->mat[14]+auxiliar_object->mtrxPTR->mat[10],
       auxiliar_object->mtrxPTR->mat[4],auxiliar_object->mtrxPTR->mat[5],auxiliar_object->mtrxPTR->mat[6]
       );
       printf("Camara asociada al objeto creada \n");

       auxiliar_object->next = _first_object;
       _first_object = auxiliar_object;
       _selected_object = _first_object;

       printf("%s\n",KG_MSSG_FILEREAD);
       borrar=1;
       break;
    }
    break;
```

- 'TAB': Alternar entre los diferentes objetos cargados, en caso de no haber ninguno cargado se mostrara un mensaje por terminal.

```
case 9: /* <TAB> */
    if (_selected_object)
    {
        _selected_object = _selected_object->next;
        //si el siguiente objeto es nulo volver al primero
        if (_selected_object == 0){
            _selected_object = _first_object;
        }
        if(vuelo == 0){
            empezarAnalisis();
        }
        if(selectedLuz == lLuces[2] ){
            //actualzizar el foco
            selectedLuz = _selected_object->focoPTR;
            lLuces[2] = _selected_object->focoPTR;
        }
    }else{
        printf("Debes cargar un objeto");
    }
    break;
```

- 'DEL': eliminar objeto seleccionado, si no hay ninguno cargado, se muestra un mensaje

```
case 127: /* <SUPR> */
    if(_selected_object!=0){
     if (_selected_object == _first_object)
        {
            //Si es el primero objeto pasar al siguiente y si no hay siguiente dejar el parametro borrar a 0
            _first_object = _first_object->next;
            deletePoligon(_selected_object);
            _selected_object = _first_object;
            if (_selected_object==0)
            {
                borrar=0;
            }

        } else {
            //Pasar al objeto anterior
            auxiliar_object = _first_object;
            while (auxiliar_object->next != _selected_object)
            auxiliar_object = auxiliar_object->next;
            auxiliar_object->next = _selected_object->next;
            deletePoligon(_selected_object);
            _selected_object = auxiliar_object;
    }
}
    break;
```

- 'CTRL ++': escalar el objeto en todas las direcciones o el zoom de la cámara o el volumen de apertura de un foco dependiendo de en que modo se este trabajando , si aun esta cargado, mensaje. (El control solo es necesario si se desea especificar la transformacional al objeto)

```
case '+':


    if ((glutGetModifiers() == GLUT_ACTIVE_CTRL) || (mCamara == 1 && mLuces == 0)){
        //Reducimos el volumen de vision haciendo zoom
            selectedCamara->params[1] = selectedCamara->params[1] * 0.5;
            selectedCamara->params[0] = selectedCamara->params[0] * 0.5;
            selectedCamara->params[2] = selectedCamara->params[2] / 0.5;
            selectedCamara->params[3] = selectedCamara->params[3] * 0.5;
            printf(" Aumentar el ZOOM\n");
    } else if (mCamara == 0 && mLuces == 0 ){
        if (_selected_object)
        {
            //Aumenta la escala
            printf("Aumentar la escala\n");
            glMatrixMode(GL_MODELVIEW);
            if (GLOBAL == 0){
                glLoadMatrixd(_selected_object->mtrxPTR->mat);
            }
            else{
                    glLoadIdentity();
            }
            glScaled(1.1,1.1,1.1);
            if ( GLOBAL == 1){
                glMultMatrixd(_selected_object->mtrxPTR->mat);
            }
            glGetDoublev(GL_MODELVIEW_MATRIX, auxMatrixptr->mat);
            auxMatrixptr->next = _selected_object->mtrxPTR;
            _selected_object->mtrxPTR =  auxMatrixptr;
            actualizarFoco();
        }else{
            printf("Carga un objeto primero\n");
        }
    } else {
        //Aumentar el corte de la camara
            if (selectedLuz->tipo == 2 )
            {
                    if (selectedLuz->corte < 180.0)
                    {
                        if (selectedLuz->corte==90.0){
                                selectedLuz->corte = 180.0;
                        }
                        else{
                                selectedLuz->corte+=1.0;
                        }

                    }else{
                        printf("No se puede reducir mas el angulo de apertura\n");
                    }
            }else{
                printf("La luz debe ser un foco\n");
            }
    }
    break;
```

- 'CTRL +-': disminuir la escala del objeto en todas las direcciones o el zoom de la cámara o el volumen de apertura de un foco dependiendo de en que modo se este trabajando, si aun esta cargado, mensaje. (El control solo es necesario si se desea especificar la transformacional al objeto)

```
case '-':
//  Aumentamos el volumen de vision aumentando zoom
    if ((glutGetModifiers() == GLUT_ACTIVE_CTRL) || (mCamara == 1 && mLuces == 0)){
            selectedCamara->params[1] = selectedCamara->params[1] / 0.5;
            selectedCamara->params[0] = selectedCamara->params[0] / 0.5;
            selectedCamara->params[2] = selectedCamara->params[2] * 0.5;
            selectedCamara->params[3] = selectedCamara->params[3] / 0.5;
            printf("Disminuir el ZOOM\n");
    } else if (mCamara == 0 && mLuces == 0 ){
        if (_selected_object)
        {
            //escalar hacia abajo todo el objeto
            printf("Disminuir la escala\n");
            glMatrixMode(GL_MODELVIEW);
            if (GLOBAL == 0){
                glLoadMatrixd(_selected_object->mtrxPTR->mat);
            }
            else{
                    glLoadIdentity();
            }
            glScaled(0.9,0.9,0.9);
            if ( GLOBAL == 1){
                 glMultMatrixd(_selected_object->mtrxPTR->mat);
            }
            glGetDoublev(GL_MODELVIEW_MATRIX, auxMatrixptr->mat);
            auxMatrixptr->next = _selected_object->mtrxPTR;
            _selected_object->mtrxPTR =  auxMatrixptr;
            actualizarFoco();
        }else{
            printf("Carga un objeto primero\n");
        }
    } else {
        //Redocir el corte de la luz
            if (selectedLuz->tipo == 2 )
            {
                    if (selectedLuz->corte > 1.0)
                    {
                        if (selectedLuz->corte==180.0){
                                selectedLuz->corte = 90.0;
                        }
                        else{
                                selectedLuz->corte-=1.0;
                        }

                    }else{
                        printf("No se puede reducir mas el angulo de apertura\n");
                    }
            }else{
                printf("La luz debe ser un foco\n");
            }
    }
    break;
```

4 Carga del objeto

Una vez pulsada la f/F, se seleccionará que objeto cargar, para esto habrá que escribir el nombre de la siguiente manera: "./objektuak/nombreObjeto.obj", siendo objektuak la carpeta en la que se guardan los objetos. La figura se mostrará inmediatamente en la ventana, si la figura no se ha encontrado saldrá un mensaje en la consola avisando de ello.

5 Transformaciones

Se aplican a un objeto, una cámara (según el modo seleccionado) o incluso a una fuente de luz para trasladar, rotar o cambiar su tamaño (esta ultima no se aplica a luces).

- Traslación(tecla 'm/M'): cambiar el elemento seleccionado de posición, esta se realizara mediante las flechas para moverlo hacia delante, atrás, izquierda o derecha y se utilizaran las teclas especiales F11 y F12 para acercar y alejar.

```
case 'M':
case 'm':
    //Activar traslaciones modo traslacion
    printf("%s\n"," Pasamos a modo traslacion" );
    modo = 0;
    break;
```

- Rotación(tecla 'b/B'): flechas izquierda y derecha para rotar sobre el eje y. Arriba y abajo para hacerlo sobre el x. F11 y F12 para hacerlo sobre el z.

```
case 'B':
case 'b':
    //Activar el modo rotacion
    printf("%s\n"," Pasamos a modo rotacion" );
    modo = 1;
    break;
```

- Escalado(tecla 't/T'): flecha izquierda/derecha para disminuir/aumentar sobre el eje x. abajo/arriba para hacerlo sobre el eje y. F11/F12 sobre el z.

```
case 'T':
case 't':
    //Activar el modo escalado
    printf("%s\n"," Pasamos a modo escalado" );
    modo = 2;
    break;
```

- Global(tecla 'g/G'): transformar en el sistema de referencia del mundo.

```
case 'G':
case 'g':

   //Activar el modo analisis si estas en modo camara o pasar al modo global si no
   if(mCamara == 1){
      if( selected_object != 0){
      vuelo = 0;
      printf("%s\n"," Modo Analisis" );

      empezarAnalisis();
      }else
      {
          printf("Carga un objeto primero\n");
      }


   }else{
      printf("%s\n"," GLOBAL en modo GLOBAL" );

      GLOBAL = 1;
   }
   break;
```

- Local(tecla ‘l/L’): transformar en el sistema de referencia local (aplicar cambios sobre el objeto o cámara).

```
case 'L':
case 'l':
    //Activar el modo vuelo si estas en modo camara o poner el modo local
    if(mCamara == 1){
        printf("%s\n"," Camara en modo Vuelo" );
        vuelo = 1;
    }else{
        printf("%s\n"," GLOBAL en modo LOCAL" );
    GLOBAL = 0;
    }

    break;
```

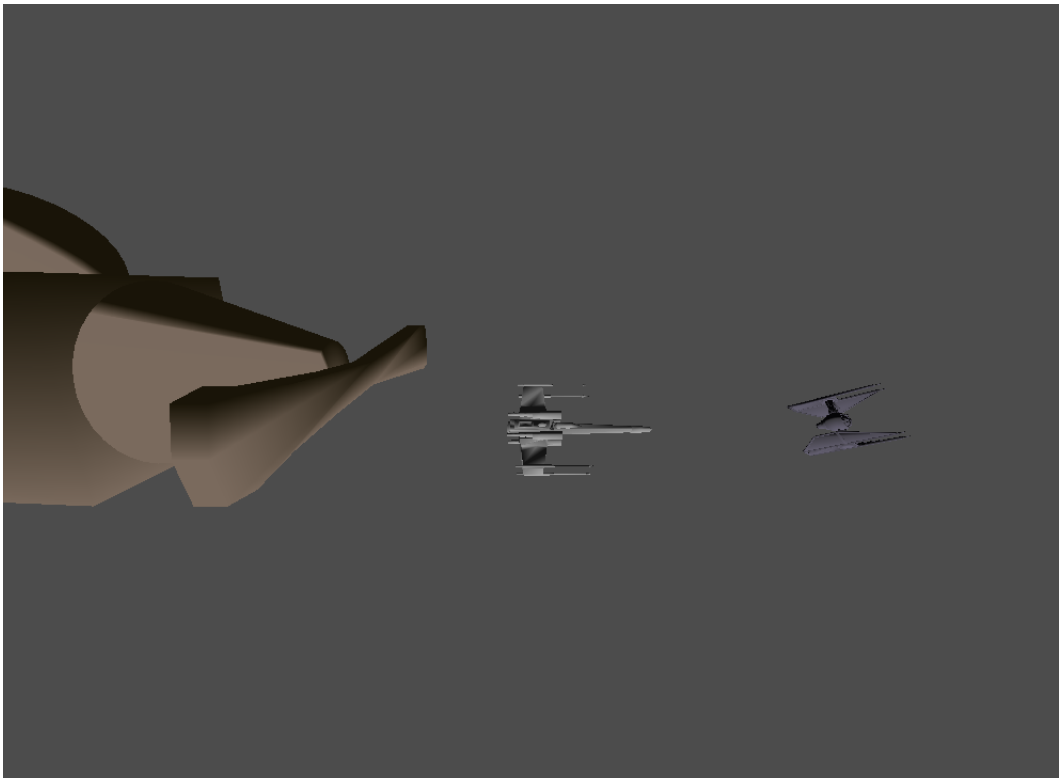
6 Modos cámara

La cámara se traslada y rota al igual que un objeto, sin embargo, el escalado sera en referente a su volumen de vision, no a su tamaño.

- Modo vuelo(tecla ‘l/L’): La cámara tiene libertad para poder moverse libremente por el espacio. Transformaciones en el sistema local de la cámara.
- Modo análisis(tecla ‘g/G’): La cámara apunta hacia el objeto seleccionado.
- Activar modo cámara (tecla ‘k’, ‘K’): Las transformaciones se realizaran a la cámara.
- Cambiar de cámara (tecla ‘c’): Pasaremos a la siguiente camara o a la primera en caso de que la seleccionada fuera la ultima.
- Cambiar a la cámara del objeto (tecla ‘C’): visualiza lo que ve el objeto seleccionado.
- Modo vuelo(tecla ‘n/N’): Crea una camara nueva estandarizada.

```
case 'N':
case 'n':
    //Creacion de una nueva camara
    nCamara = (camara  *) malloc(sizeof (camara));
    crearCamara(nCamara, 0.0, 0.0, 20.0,
                         0.0, 0.0, 0.0,
                         0.0, 1.0, 0.0);
    nCamara->next = iniCamara;
    iniCamara = nCamara;
    selectedCamara = iniCamara;

    break;
```

- Cambiar de tipo de proyección (teclas ‘p/P’): Alterna las perspectivas entre paralela y en perspectiva.



7 Modos cámara

Las luces vienen desactivadas al principio sin embargo tanto su creación activación del modulo o luces concretas o la creación de luces nuevas podrá ser hecha en cualquier momento. Las luces 1, 2 y 3 vienen activadas por defecto siendo estas el sol una bombilla y el foco asociado al objeto.

- Activar y desactivar la iluminación (tecla ‘F9’): Activa y desactiva el modulo de luces.

```
    //activar/desactivar luces
case GLUT_KEY_F9:
    if(glIsEnabled(GL_LIGHTING)){
        //desactivar luces
        glDisable(GL_LIGHTING);
        printf("Luces desactivadas \n");
    } else {
        //activar luces
        glEnable(GL_LIGHTING);
        printf("Luces activaddas\n");

    }
break;
```

- Encender/apagar luces (teclas F1-F8): encender y apagar la fuente fuente de luz seleccionada, si aun no esta creada se mostrara un mensaje por consola.

```
  case GLUT_KEY_F1:
  case GLUT_KEY_F2:
  case GLUT_KEY_F3:
  case GLUT_KEY_F4:
  case GLUT_KEY_F5:
  case GLUT_KEY_F6:
  case GLUT_KEY_F7:
  case GLUT_KEY_F8:
      //Activar o desactivar la luz vinculada a la tecla
      encenderApagarLuz(key-1);
  break;
```

- Seleccionar fuente de luz (teclas 1-8): seleccionada la fuente correspondiente a dicho numero.

```
case '1':
case '2':
case '3':
case '4':
case '5':
case '6':
case '7':
case '8':
//Elegir la luz

    selectedLuz = lLuces[atoi(&key)-1];
    printf("Has elegido la luz %d ",atoi(&key));
    if(selectedLuz->inicializada == 1){
        printf("Todo ha ido ido correctamente \n");
    }else{
        printf("Sin embargo todavia esta por inicializarse\n");
    }

    break;
```

- Flat/smooth (tecla ‘Insert’): cambia el tipo de iluminacion.

```
case GLUT_KEY_INSERT:
    //flat
    if(luzSmooth == 0){
        glShadeModel(GL_SMOOTH);
        luzSmooth = 1;
        printf("Iluminacion de tipo suave\n");
        glutPostRedisplay();


    } else { //smooth
        glShadeModel(GL_FLAT);
        luzSmooth = 0;
        printf("Iluminacion de tipo liso\n");


    }
break;
```

- Nueva fuente (tecla '0'): Crea una nueva luz si la seleccionada no ha sido creada o la sobrescribe. Si la luz seleccionada

```
case '0':
    //Siempre que no sea una las tres primeras luces reescribira o inicializara la luz seleccionada
    if(selectedLuz == lLuces[0] || selectedLuz == lLuces[1] || selectedLuz == lLuces[2] ){
        printf("Esta luz no puede ser modificada ni reescrita\n");
    }else{

        if(selectedLuz->inicializada == 1){
            printf("Se sobreescribira la luz seleccionada\n");
        }
        printf("Si desea crear un foco pulse 'f' si desea crear una bombilla pulse 'b\n");
        scanf(" %c", &c);
        getchar();
        if(c == 'f'){
            selectedLuz->tipo = 2;
            selectedLuz->inicializada = 1;
            glEnable(GL_LIGHT0+selectedLuz->num);
            selectedLuz->exponente = 0.0;
```

- Aplicar transformaciones a la luz (Tecla 'A', 'a'): aplicar transformaciones a la fuente de luz seleccionada, estas no se podran aplicar al sol.

```
case 'A':
case 'a':
printf("%s\n"," Pasamos a modo Luces" );
printHelpIlum();
//Activar el modo luces
    if(mLuces == 1) {
        mLuces = 0;
    } else {
        mLuces = 1;
    }
    break;
```

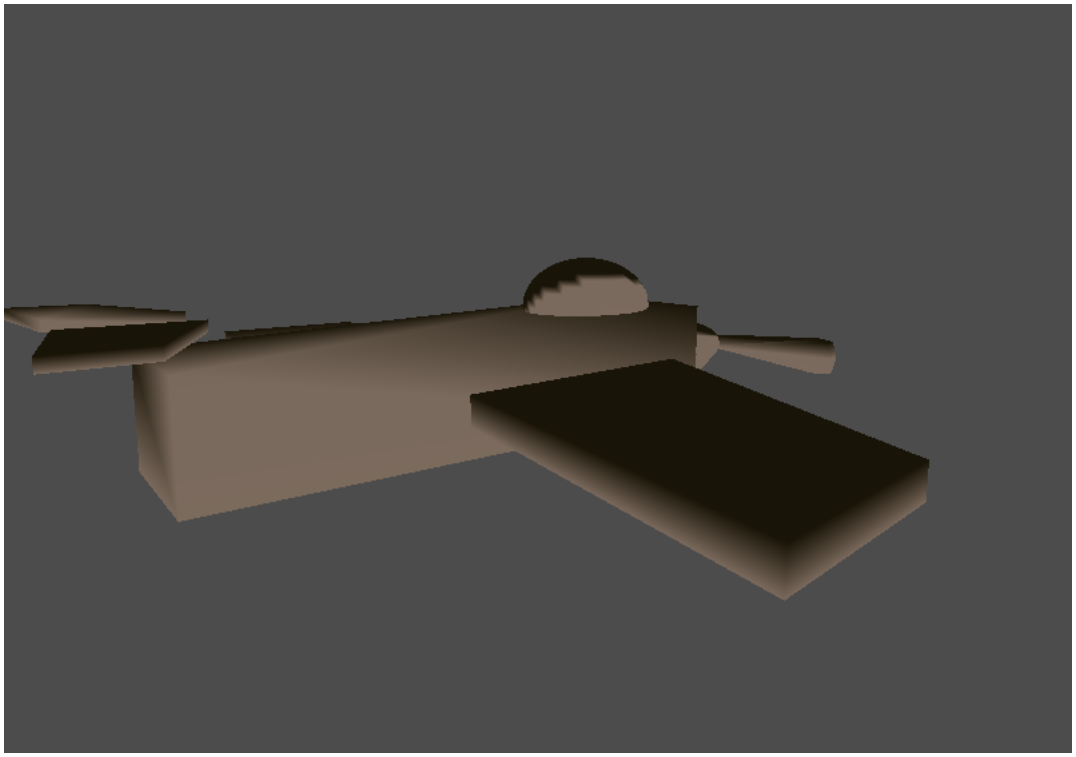
- Aplicar transformaciones a la luz (Tecla 'U', 'u'): Cambia el material del objeto.

```
case 'U':
case 'u':
//Cambiar el material asociado al objeto
    printf("Los mateeriales son los siguientes\n");
    printf("1.- Bronce\n");
    printf("2.- Plata\n");
    printf("3.- Oro\n");
    printf("4.- Obsidiana\n");
    printf("Introduce el numero correspondiente al material para cambiarlo\n");
    do {
        read = scanf("%d", &m);
        getchar();
        if(read!=1){
            printf("Vuelve a intentarlo\n");
        }
    }while(read!=1);

    if( m>=1 && m<5){
    _selected_object->matPTR = lMateriales[m-1];
    }else
    {
        printf("Ese material no esta disponible\n");
    }


    break;
```

- Incrementar apertura foco (tecla '+'): Incrementa el angulo de apertura del foco, solo es aplicable a los focos.
- Decrementa apertura foco (tecla '-'): Decrementa el angulo de apertura del foco, solo aplicable a focos.

8 Codigo C

Aqui encontraras otros fragmentos del codigo que son usados en la implementacion de la aplicacion:



```
//Cambia el estado de las luces
void encenderApagarLuz(int l){
    //luz creada
    if(lLuces[l]->inicializada) {

        int luz = GL_LIGHT0 + l;
        if(glIsEnabled(luz)) {
            //apagar luz
            glDisable(luz);
            printf("hemos apagado la luz %d\n", l);
        } else {
            //encender luz
            glEnable(luz);
            printf("hemos encendido la luz %d\n", l);
        }

    } else {
        printf("Esta luz no esta creada\n");

    }
}
```

```
//traslada la luz en el eje y lado dados
void lTrasladar(char eje, char lado) {
    int tipo = selectedLuz->tipo;

    printf("x %lf", selectedLuz->posicion[0]);
    printf("y %lf", selectedLuz->posicion[1]);
    printf("z %lf", selectedLuz->posicion[2]);
    printf("\n");

    if(tipo == 0) {
        printf("Este tipo de traslaciones no se pueden aplicar al sol\n");

    } else {
        if(eje == 'x'){
            if(lado == '+') {
                selectedLuz->posicion[0] += 0.1;
            } else {
                selectedLuz->posicion[0] -= 0.1;
            }
        } else if(eje = 'y') {
            if(lado == '+') {
                selectedLuz->posicion[1] += 0.1;
            } else {
                selectedLuz->posicion[1] -= 0.1;
            }
        } else {
            if(lado == '+') {
                selectedLuz->posicion[1] += 0.1;
            } else {
                selectedLuz->posicion[1] -= 0.1;
            }
        }

    }
}
```

```
//Rota la camara la cantidad de grados dada sobre el eje proporcionado
void rotacionLibre(double grados, double x, double y, double z){

    printf("%s\n", "Rotando la camara en modo Vuelo");

    point3 aux;
    aux.x = selectedCamara->imcsr[12];
    aux.y = selectedCamara->imcsr[13];
    aux.z = selectedCamara->imcsr[14];

    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glRotated(grados,x,y,z);
    glMultMatrixd(selectedCamara->imcsr);
    glGetDoublev(GL_MODELVIEW_MATRIX, selectedCamara->imcsr );

    selectedCamara->imcsr[12] = aux.x;
    selectedCamara->imcsr[13] = aux.y;
    selectedCamara->imcsr[14] = aux.z;

    printf("%s\n", "Fin de la rotacion");


}

//Rota la cama alrededor del objeto
void rotacionLigada(double grados, double x, double y, double z){

    printf("%s\n", "Rotacion de la Camara mirando al objeto");

    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glTranslated(_selected_object->mtrxPTR->mat[12], _selected_object->mtrxPTR->mat[13], _selected_object->mtrxPTR->mat[14]);
    glRotated(grados,x,y,z);
    glTranslated(-_selected_object->mtrxPTR->mat[12], -_selected_object->mtrxPTR->mat[13], -_selected_object->mtrxPTR->mat[14]);

    glMultMatrixd(selectedCamara->imcsr);
    glGetDoublev(GL_MODELVIEW_MATRIX, selectedCamara->imcsr );

    printf("%s\n", "Fin de la rotacion");

}
```