Práctica final

Pep Toni Bibiloni[1], Jhonier Meneses[2]

[1]josep.bibiloni1@estudiant.uib.cat
[2]jhonier.meneses1@estudiant.uib.cat

1. RESUMEN

En este documento se expone la implementación de las diferentes etapas de la práctica de informática gráfica.Así como los diferentes conceptos que se han puesto en práctica al ir solventando los diferentes problemas.

2. INTRODUCCIÓN

Para realizar esta práctica se ha dividido en diversas etapas.Para las cuales se han realizado con openGL.OpenGL (Open Graphics Library) es una especificación estándar que define una API multilenguaje y multiplataforma para escribir aplicaciones que produzcan gráficos 2D y 3D. La interfaz consiste en funciones diferentes que pueden usarse para dibujar escenas tridimensionales complejas a partir de primitivas geométricas simples, tales como puntos, líneas y triángulos

3. ENTORNO DE TRABAJO

Equipo

- Jhonier Duvan Meneses Magon (49608202P)
- Josep Antoni Bibiloni Fontirroig (41617674W)

Entorno de desarrollo

El editor de código utilizado es Visual Studio Code. La versión de g++ es la 9. También se hace uso de Git para el control de versiones. Para el desarrollo de los gráficos se utilizaron librerías de OpenGL.Sistema Operativo linux.

4. ETAPAS

4.1 Configuración entorno

4.1.1 Descripción

Generación del Workspace y primer programa en Opengl mediante GLUT.

4.1.2 Implementación

Se configuró el entorno de trabajo en el cual fue necesario instalar la librería freeglut. En la figura 1 podemos ver el resultado de la ejecución.

Figura 1: Ejecución etapa1

4.2 Doble buffer y escalado

4.2.1 Descripción

- **Doble Buffer**: En el programa anterior de la animación, las primitivas se realizan directamente sobre el buffer del dispositivo de pantalla. Lo que se produce parpadeo al borrar la ventana y a continuación redibujarla. Para evitar este efecto se utiliza un doble buffer, se visualiza sobre un segundo buffer y una vez se obtiene la imagen deseada se intercambia con el buffer de pantalla. Las funciones a utilizar son glutInitDisplayMode () y glutSwapBuffers().
- **Mantenimiento de aspect/ratio**: Al cambiar el tamaño de la ventana, el cuadrado se deforma porque el volumen de recorte definido permanece intacto, corregir el código de tal manera que cuando la ventana cambie de

tamaño se ajuste a las proporciones de un cuadrado (alto=ancho), para verificar mejor podemos insertar un cuadrado.

- **Escena 2D y pendulo doble**:Implementar una escena 2D con primitivas de GLUT y aplicar transformaciones. Luego implementar un péndulo doble.

4.2.2 *Implementación*

- **Doble Buffer**: Primeramente, para hacer uso del doble buffer se han utilizado las funciones:

```
    glutInitDisplayMode (GLUT_RGBA | GLUT_DOUBLE);
    glutSwapBuffers();
```

- **Mantenimiento del aspect/ratio**:

  Capturamos el evento de cambio de tamaño de ventana, al darse este evento modificamos el volumen de visualización para que tenga las mismas proporciones que la ventana. La declaración de la función GLUT donde registramos el evento de cambio de ventana es:

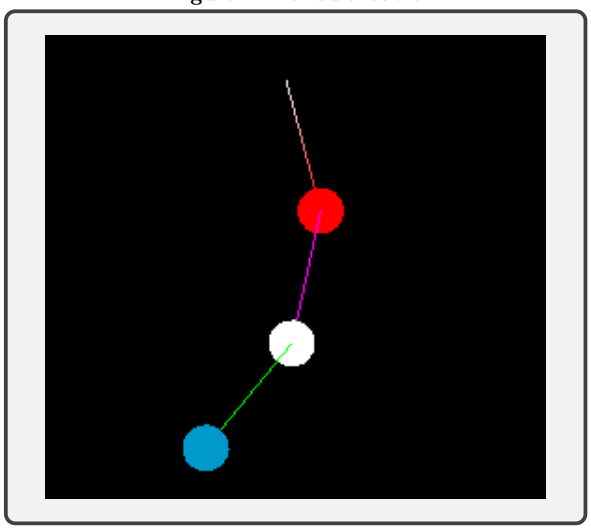
```
    //reshape es la funcion de callback
    void glutReshapeFunc(reshape);
```

  En donde la función de callback "reshape" es:

```
// Funcin que visualiza la escena OpenGL
void reshape(GLsizei w, GLsizei h){
    if(w==h){return;}
    if(w>h){
        aspecto = (GLdouble)h / (GLdouble)w;
        double  wi = aspecto * h;
        wi=wi/aspecto;
        double left=(w-wi)/2.0f;
        glViewport(left, 0, wi, h);
    }else{
        aspecto = (GLdouble)w / (GLdouble)h;
        double hi = aspecto * w;
        double bottom = (h - hi)/2.0f;
        hi=hi/aspecto;
        glViewport(0, bottom, w, hi);
    }
}
```

- **Escena 2D y péndulo doble**: En la figura 2 podemos ver la ejecución del péndulo doble. En donde se utilizaron primitivas y transformaciones 2D para su implementación.

Figura 2: Péndulo doble



4.3 Escena 3D

4.3.1 *Descripción*

En esta etapa pasamos al espacio 3D. El objetivo es similar a los pasos de la etapa 2, pero ahora definimos un sistema de coordenadas cartesiano con los ejes (X, Y, Z). Tendremos que definir las funciones de ejes, y de planos de referencias, que pueden ser visibles o no a petición del usuario (o con cierta transparencia, alpha channel). El objetivo de esta etapa es crear una escena simple con primitivas 3D y aplicar las funciones de transformación y visualización del tema 2 y 3.

4.3.2 *Implementación*

En esta etapa se ha implementado una escena 3D,en la cual se ha aprovechado y se ha implementado una primera versión de la etapa final.

- **Primitivas 3D**: En esta escena se han definido nuestras propias primitivas 3D como por ejemplo la siguiente:

```
void drawCylinder(GLfloat base, GLfloat height){
        GLUquadricObj *qobj;
        qobj = gluNewQuadric();
        gluQuadricDrawStyle(qobj, GLU_FILL);
        glRotatef(90.0f,-1.0f,0.0f,0.0f);
        gluDisk(qobj,0.0f,base,50,50);
        gluCylinder(qobj,base,base,height,50,50);
        glTranslatef(0.0f,0.0f,height);
        gluDisk(qobj,0.0f,base,50,50);
        glTranslatef(0.0f,0.0f,-height);
        glRotatef(90.0f,1.0f,0.0f,0.0f);
}
```

El código anterior permite dibujar un cilindro.

En el siguiente fragmento de código se puede ver la especificación de todas las funciones que nos permiten dibujar nuestros propios objetos.

```
void draw3dRectangle(GLfloat width,GLfloat height,
                GLfloat depth);

```

```
4  void drawTwo3dRectanle(GLfloat angle,GLfloat height,
5              GLfloat thickness);
6
7  void drawLamp(GLfloat base,GLfloat height);
8
9  void drawScrew(GLfloat base, GLfloat height);
10
11 void drawBase();
```

La función ***draw3dRectangle*** nos dibuja un rectangulo 3d, con la longitud, altura y anchura indicadas.

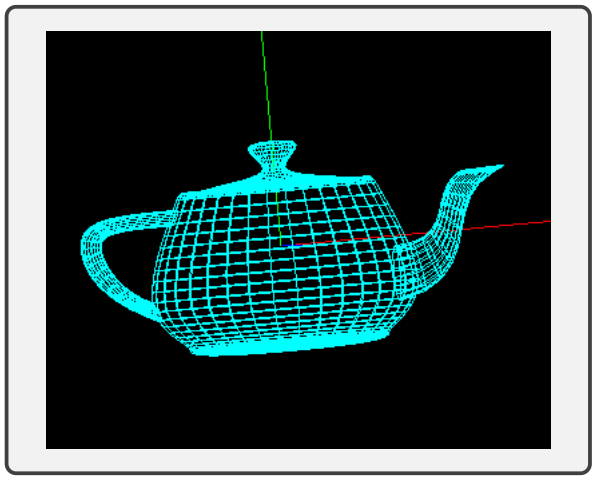
La función ***drawTwo3dRectangle*** nos dibuja dos rectangulos 3d paralelos. En la función se indica la distancia entre los dos rectangulos.

La función ***drawLamp*** dibuja el cono de la lámpara. Que estará formado por un cilindro y un cono.

La función ***drawBase*** dibuja la base de la lámpara. Para crear la base hemos utilizado la función de dibujar cilindro con la función glutSolidThorus para dar un borde circular.

También hemos probado a dibujar primitivas ya definidas en la librería glut, como por ejemplo con la función glutWireTeapot(), que dibuja una taza de té no sólida (ver figura 3).

Figura 3: Taza de té



- **Perspectiva**: Se ha definido una perspectiva con la función glFrustum():

```
1  glFrustum(1,1,1/aspectRatio,1/aspectRatio,
2          2.0, 2.0);
```

Frustum permite que hacer que los objetos lejanos parezcan más pequeños.Mietras que glOrtho y gluOrtho2D configuran modos de proyección 2D (es decir, proyección paralela). Estos no son necesariamente 2D, pero implican que los objetos más lejanos no son más pequeños que los más cercanos.

- **Buffer de profundidad**: El buffer de profundidad nos permite tener en cuenta de la proximidad de los objetos.

  Activamos el buffer de profundidad con:

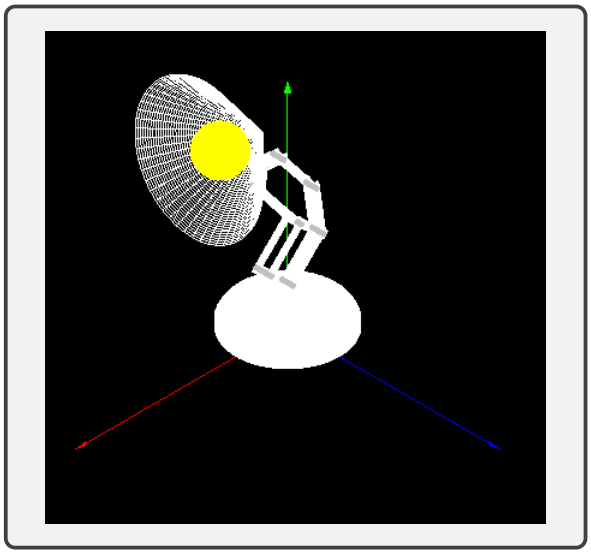
```
1  glEnable(GL_DEPTH_TEST);
```

  Con glClear podemos borra el buffer de profundidad pasándole el flag GL-DEPTH-BUFFER-BIT.

```
1  glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
```

- **Escena 3D**: Se ha implementado una escena 3D con primitivas propias y de glut.La escena consiste en una primera versión de la etapa 6.En donde la lámpara es articulada y, por lo tanto, se puede subir y bajar la cabeza con las teclas "w" y "s" respectivamente.En la figura 4 podemos ver la ejecución de la escena.

Figura 4: Escena 3D



4.4 Movimiento de cámara

4.4.1 Descripción

En esta etapa el objetivo es que el usuario pueda moverse libremente por la escena. Es decir, la cámara virtual que nos da la vista de nuestra escena podrá variar de acuerdo con una seria de funciones. El movimiento de la cámara vendrá dado por las funciones siguientes de orientación y translación:

- Movimiento de la cámara de Paneo (la cámara gira sobre su propio eje de izquierda a derecha). LEFT Girar Izquierda/ RIGHT Girar Derecha.
- Movimiento de la cámara de Tilt (la cámara gira sobre su propio eje de hacia arriba o hacia abajo)

- Movimientos laterales o acercamiento/alejamiento (travelling / Dolly (in / out)
- (Flecha UP Avanzar/DOWN Retroceder, IZQ/DER)
- Movimientos angularesde la cámara. Coordenadas Esféricas. Objetos en el centro de la escena. Rotación respecto al origen de la cámara. Ángulos: Nadir, Contrapicado, Normal, Picado y Cenital.

4.4.2 *Implementación*

Se ha realizado diferentes configuraciones para el movimiento de la cámara y diferentes perspectivas

- **Movimientos**: Para tener en cuenta los movimientos de la cámara, se tienen en cuenta diversos valores propios de esta. La cámara tiene 3 vectores; *eye, center* y *up*. También tiene los ángulos de rotación *pitch* y *yaw*.

  En movimientos hacia delante y hacia atrás, se modifica el vector *eye*. En movimientos angulares, se modifica el vector *center*. Para estos movimientos angulares, se calcula el nuevo vector teniendo en cuenta que son coordenadas polares.

- **Cámara libre**: Con la cámara libre se puede mover la cámara libremente por el escenario.Podemos ver en la siguiente tabla que tecla corresponde a cada movimiento:

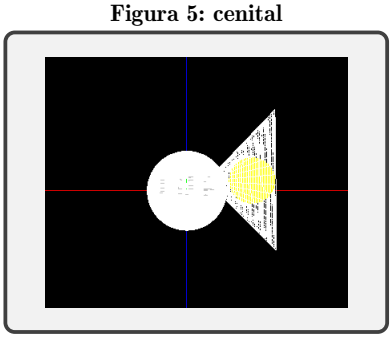
Tecla	Movimiento
↑	arriba
↓	abajo
←	izquierda
→	derecha
"+"	acercar
"-"	alejar

- **Perspectivas**: Primeramente, veamos las perspectivas que se han implementado.
  - **Cenital** Para poner la perspectiva cenital se tiene que pulsar la tecla "F1".El comando que cambia la perspectiva de la cámara a un plano cenital es :

```
1        gluLookAt(0,3,0.15,0,0,0,0,1,0);
```

  En la figura 5 podemos ver dicha perspectiva en la escena.

Figura 5: cenital

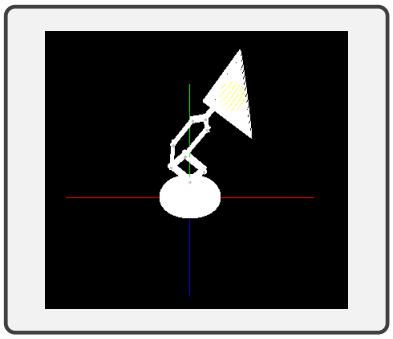


  - **Picado** Para poner la perspectiva cenital se tiene que pulsar la tecla "F2".El comando que cambia la perspectiva de la cámara a un plano picado es:

```
1        gluLookAt(0,2.5,3,0,0,0,0,1,0);
```

  En la figura 7 podemos ver dicha perspectiva en la escena.

Figura 6: picado

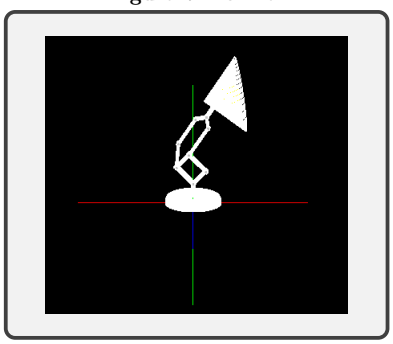


  - **Normal** Para poner la perspectiva cenital se tiene que pulsar la tecla "F3".El comando que cambia la perspectiva de la cámara a un plano picado es:

```
1        gluLookAt(0,-2,0.15,0,0,0,0,1,0);
2            break;
```

  En la figura 6 podemos ver dicha perspectiva en la escena.

Figura 7: normal

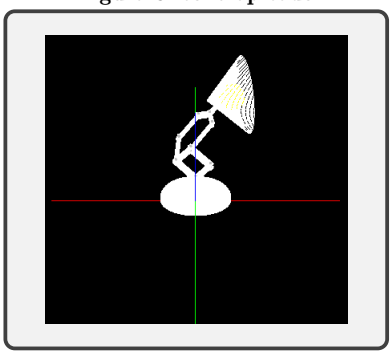


  - **Contrapicado** Para poner la perspectiva cenital se tiene que pulsar la tecla "F4".El comando que cambia la perspectiva de la cámara a un plano normal es:

```
1        gluLookAt(0,-1.5,3,0,0,0,0,1,0);
```

  En la figura 8 podemos ver dicha perspectiva en la escena.

Figura 8: contrapicado

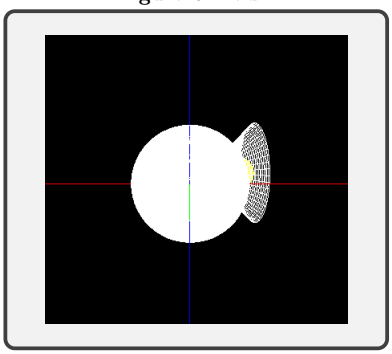


- **Nadir** Para poner la perspectiva cenital se tiene que pulsar la tecla "F5".El comando que cambia la perspectiva de la cámara a un plano nadir es:

```
1        gluLookAt(0,2.5,3,0,0,0,0,1,0);
```

En la figura 9 podemos ver dicha perspectiva en la escena.

Figura 9: nadir



4.5 Luces y Materiales, Sombreado

4.5.1 Descripción

En esta etapa, iniciamos el proceso de realismo. Para ello añadimos luces a la escena y materiales a los objetos.

El movimiento de la luz

- Sobre el eje x (Tecla "a" IZQ / Tecla "z" DER)
- Sobre el eje y (Tecla "d" ATRÁS / Tecla "c" ADELANTE)
- Sobre el eje z (Tecla "s" ARR / Tecla "x" ABJ)

Ajustes de la luz:

- Especular (Tecla "1" decrementa/ Tecla "2" incrementa)
- Ambiente (Tecla "3" decrementa/ Tecla "4" incrementa)
- Difusa (Tecla "5" decrementa/ Tecla "6" incrementa)

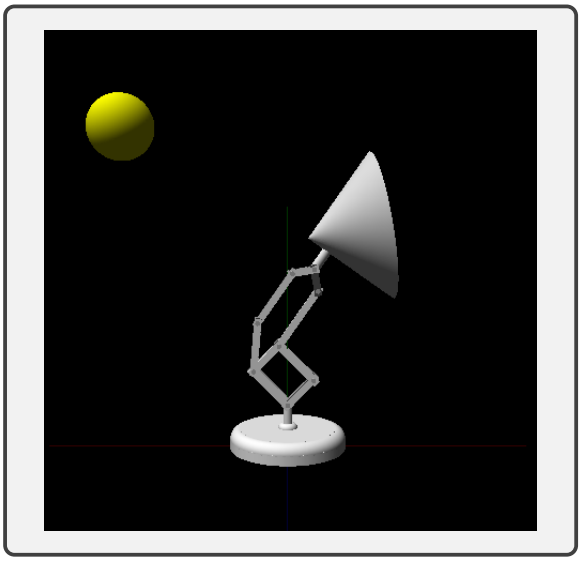
El tipo de sombreado se puede cambiar de GL_FLAT a GL_SMOOTH y viceversa pulsando ENTER.

4.5.2 Implementación

- **Luces**:

  Para la luz tendremos guardados sus valores: posición, luz especular, luz difusa y luz ambiente. Estos valores se podrán modificar mediante las teclas, según la tecla se incrementara o decrementara el valor seleccionado en 0.1f.

Figura 10: Luces



- **Materiales**:

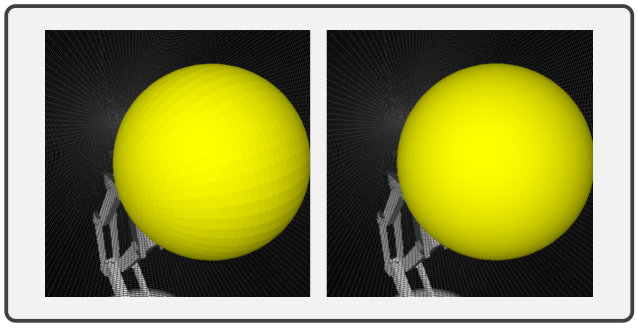
  La función glColorMaterial hace que un color de material realice un seguimiento del color actual

```
1  glEnable(GL_COLOR_MATERIAL);
2  glColorMaterial(GL_FRONT_AND_BACK,
3      GL_AMBIENT_AND_DIFFUSE);
```

- **Sombreado**:

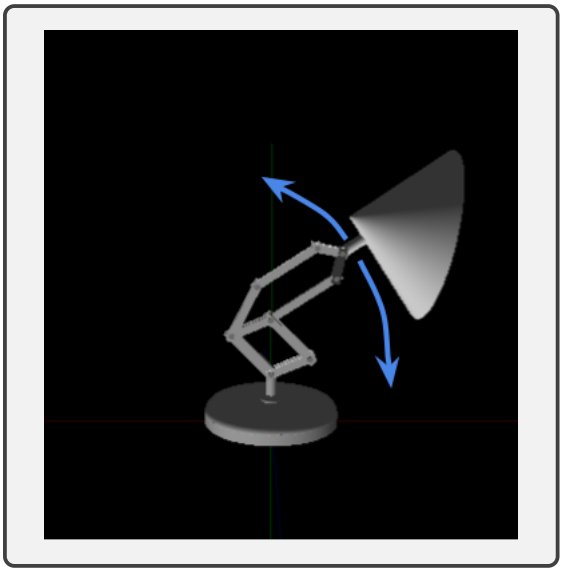
  Mediante la función *glShadeModel* podemos elegir el tipo de sombreado. En esta practica se usan el GL_FLAT y el GL_SMOOTH.

Figura 11: GL_FLAT (IZQ) y GL_SMOOTH (DER)



- **Movimiento** Para observar el renderizamiento de objetos en movimiento, hemos añadido el primer movimiento de la lampara. Este movimiento consiste en subir o bajar la lámpara. Para subir se tiene que pulsar la tecla "o" y para bajar la tecla "l".

Figura 12: Primer movimiento de la lampara



4.6 Realismo

4.6.1 Descripción

En esta etapa se mejorará la lámpara ya creada en etapas anteriores con el objetivo de mejorar el realismo de la escena.

En la figura 13 podemos observar el resultado de la escena realista. Es una escena compleja con una lámpara que es articulada, con dos movimientos diferentes.

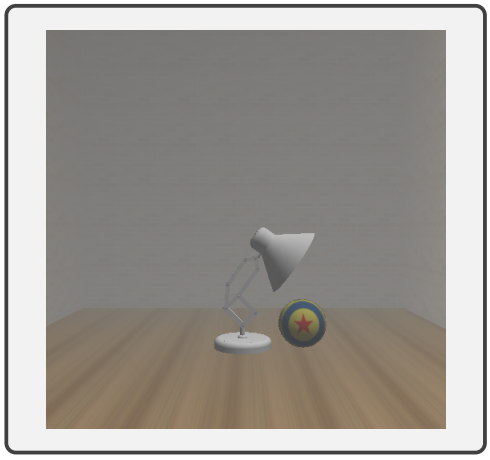
Figura 13: Escena realista

Como parte optativa se han añadido las siguientes funcionalidades:

- Niebla

  Se añade la posibilidad de añadir niebla a la escena. Mediante el menú de opciones, se podrá activar o desactivar.

Figura 14: Niebla



- Texturas

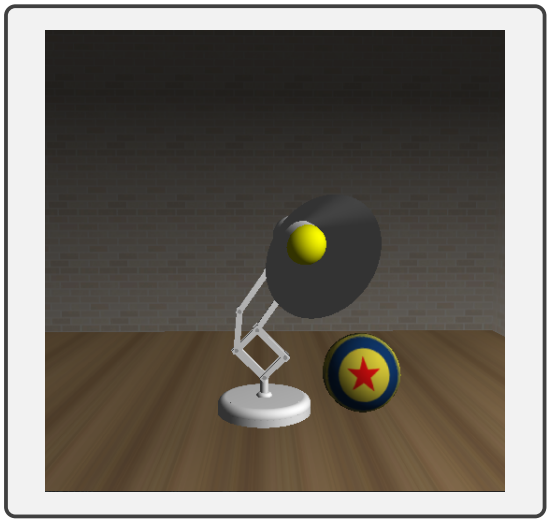
  Vamos a utilizar texturas para dar más realismo a la bola, las paredes y el suelo. Utilizaremos SOIL, la librería de imágenes de OpenGL.

Figura 15: Textura

- Movimiento

  Aparte del movimiento de abajo y arriba, anteriormente implementado en etapas posteriores, se ha añadido la posibilidad de girar la cabeza de la lámpara. Se gira con la tecla "j" a la izquierda y con la letra "l" a la derecha.

Figura 16: Girar



4.6.2 *Implementación*

- Niebla

  Para implementar la niebla se ha utilizado las funciones de Fog de gl.

```
        glFogi(GL_FOG_MODE, GL_LINEAR);
        glFogf(GL_FOG_START, 0.1);
        glFogf(GL_FOG_END,8.0);
        glFogf(GL_FOG_DENSITY,0.8);
        glHint(GL_FOG_HINT,GL_NICEST);
        float color[] = { 0.5,0.5,0.5,1.0};
        glFogfv(GL_FOG_COLOR, color);
```

- Texturas

  Para implementar las texturas se han utilizado las funciones de OpenGL. Podemos diferenciar entre el texturizado de superficies planas con el texturizado de esferas.

  Para hacer las texturas en superficies planas hemos utilizado esta función

```
void initTexture(int i){
        glEnable(GL_TEXTURE_2D);
        glBindTexture(GL_TEXTURE_2D,textures[i]);
        GLfloat plano_s[4] = {2, 0, 0, 0}; // s=x
        GLfloat plano_t[4] = {0, 2, 0, 0}; // t=y
        glTexParameteri(GL_TEXTURE_2D,
            GL_TEXTURE_WRAP_S,GL_REPEAT);
        glTexParameteri(GL_TEXTURE_2D,
            GL_TEXTURE_WRAP_T,GL_REPEAT);
        glTexEnvi(GL_TEXTURE_ENV,
            GL_TEXTURE_ENV_MODE,GL_MODULATE);
        glTexGeni (GL_S, GL_TEXTURE_GEN_MODE,
            GL_OBJECT_LINEAR);
        glTexGenfv (GL_S, GL_OBJECT_PLANE, plano_s);
        glEnable (GL_TEXTURE_GEN_S);
        glTexGeni (GL_T, GL_TEXTURE_GEN_MODE,
            GL_OBJECT_LINEAR);
        glTexGenfv (GL_T, GL_OBJECT_PLANE, plano_t);
        glEnable (GL_TEXTURE_GEN_T);
}
```

  Se texturizan el suelo y las paredes, por lo que, son superficies grandes. Al ser superficies grandes, la textura se alargaba y no daba el resultado esperado. Para solucionar esto, se han dividido las superficies en múltiples, para que así la textura se aplique correctamente. Además, se han especificado las normales de cada una de estas superficies mediante glNormal.

  La forma de aplicar la textura a la bola ha sido la siguiente:

```
void initTextureSphere(int i){
        glEnable(GL_TEXTURE_2D);
        glBindTexture(GL_TEXTURE_2D,textures[i]);
        glTexEnvi(GL_TEXTURE_ENV,
            GL_TEXTURE_ENV_MODE,GL_MODULATE);
        glTexGeni (GL_S, GL_TEXTURE_GEN_MODE,
            GL_SPHERE_MAP);
        glEnable (GL_TEXTURE_GEN_S);
        glTexGeni (GL_T, GL_TEXTURE_GEN_MODE,
            GL_SPHERE_MAP);
        glEnable (GL_TEXTURE_GEN_T);
}
```

5. GUÍA DE USO

- **Menú** Haciendo clic derecho hacemos aparecer el menú.En el cual podemos cambiar la niebla, la perspectiva y la luz

Figura 17: Menú

Figura 18: Selección Niebla

Figura 19: Selección Perspectiva

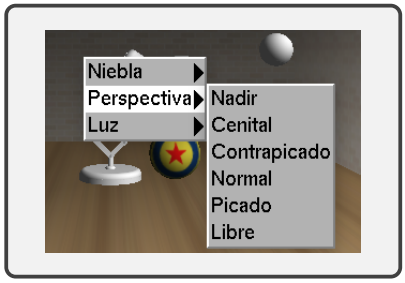
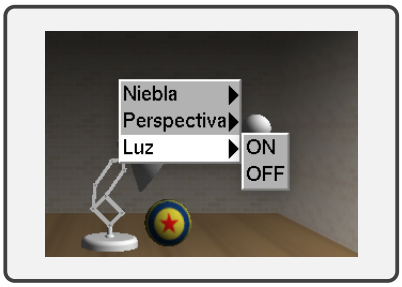
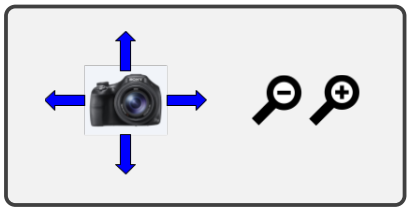


Figura 20: Selección Luz



- Cámara

Figura 21: Movimiento camara



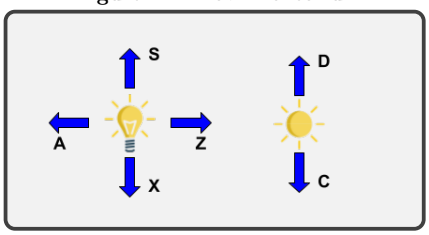
La cámara se puede mover mediante las teclas flechas indicadas en la figura 22, el zoom se puede hacer mediante las teclas "-" y "+".

Otra opción es mover la cámara mediante el ratón seleccionando y arrastrando. Para hacer zoom se puede hacer con la rueda del ratón.

- Luz De la luz se pueden cambiar los componentes:
  - Luz especular
    - '1': Decrementa 0.1
    - '2': Incrementa 0.1
  - Luz ambiente
    - '3': Decrementa 0.1
    - '4': Incrementa 0.1
  - Luz difusa
    - '5': Decrementa 0.1
    - '6': Incrementa 0.1

  Y su posición

Figura 22: Movimiento luz



Mediante el menú se pueden encender y apagar las luces

- Shade Model

  Mediante el espacio, se puede intercambiar el modelo de shade entre el modelo Flat y el modelo Smooth

- Movimiento Lámpara

  La lámpara tiene 4 movimientos:

  - "I" Arriba
  - "K" Abajo
  - "J" Izquierda
  - "L" Derecha

6. CONCLUSIÓN

Durante la realización de la práctica hemos comprendido cómo funcionan los gráficos por ordenador y los diferentes conceptos teóricos que se han visto durante el curso.Támbien ha sido un aprendizaje progresivo y ordenado debido a que a medida que se aumentaba de etapa se ponían en práctica conceptos más complejos o que dependían de etapas anteriores

En conclusión, esta es una práctica muy útil para comprender los conceptos de la asignatura.

7. BIBLIOGRAFÍA

[1] SOIL OpenGL [En línea] en: https://technoteshelp.com/c-how-do-i-install-soil-simple-opengl-image-loader/ [Accedido 22-may-2022]

[2] Wikipedia. OpenGL [En línea] Disponible en: https://es.wikipedia.org/wiki/OpenGL [Accedido 22-mar-2022]

8. AGRADECIMIENTOS

Gracias a todos los profesores que nos han enseñado aquellos conceptos necesarios para poder realizar esta práctica sin mayor complicación.