No. Cuenta: 316166496 Grupo: 4

Proyecto Final – Manual Tecnico

TEORÍA DE COMPUTACIÓN GRÁFICA

ING. CARLOS ALDAIR ROMAN BALBUENA

Fechas de entrega: 27 de mayo 2022

Introducción y objetivos:

En el siguiente documento se encontrará desarrollado el proyecto final de la clase de teoría de Computación Gráfica e Interacción Humano Computadora. En él se utilizaron todas las herramientas aprendidas durante el curso desde modelación de objetos hasta animación sencilla y compleja para presentar un trabajo capaz de asemejarse al espacio de referencia basado en la serie de Netflix llamada Arcane. En este manual se explicará a detalle cómo se utilizan las funciones para activar las animaciones con las que se mueven los objetos establecidos, además de como en aspectos generales se cumple con cada uno de los puntos.

El proyecto Final toma como base el proyecto final del laboratorio, añadiendo una nueva habitación conectada a través de unas escaleras. En esta se crearon nuevos objetos o muebles que son partes del ambiente planteado en las imágenes de referencia proporcionadas en el primer avance del proyecto.

Alcance del proyecto:

El proyecto presentado tiene como objetivo crear un espacio virtual con modelado 3D que cumpla con 5 animaciones: 3 sencillas y 2 complejas. Donde cada pieza de modelado y la temática se asemeje a lo planteado en el primer archivo entregado de imagen de referencia. Se colocarán figuras geométricas con las que se podrá interactuar presionando alguna de las teclas.

Desarrollo:

- Imagen de referencia:

Sección Proyecto del laboratorio:

Como punto de partida es importante recordar la primer entrega de este proyecto en la cual se establecieron los puntos u objetos a crear, la arquitectura a alcanzar y la temática dentro o fuera de la fachada para que coincidiera con lo estipulado a desarrollar para respetar el aspecto a evaluar de realismo.

Se recreará una taberna llamada “la última gota” la cual está construida en la historia a base de chatarra debido a que no es una ciudad pionera en tecnología. Se podría categorizar como genero steampunk.

Fachada:

Cuarto:

Elementos dentro del cuarto:

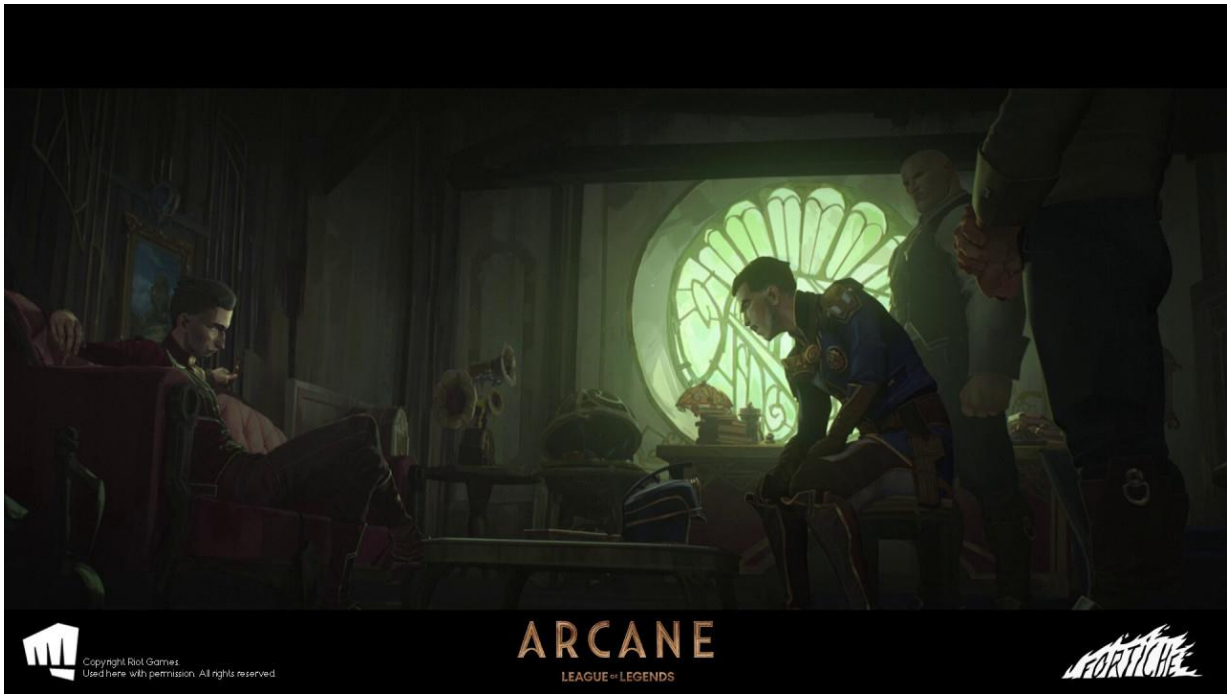
1. Rocola
2. Mesas, Sillones de terciopelo y sillas
3. Botellas de vidrio, sifones y Barriles de madera
4. Dianas de prueba de tiro
5. Guantes de boxeador (Guantes Hextech de Vi)
6. Juguete de changuito con platillos
7. Hookah

Estos elementos son los que se plantearon en un inicio para recrear en OpenGl, pero se agregaron otros objetos como una esfera de disco y la barra del bartender.

- Sección Proyecto de teoría:

Aprovechando el proyecto realizado en el laboratorio se agregará un nuevo cuarto al proyecto que consistirá en un estudio amueblado que se conectará mediante un paso de escaleras con la primera sección del proyecto desarrollado en teoría.


Copyright Riot Games
Used here with permission. All rights reserved.
ARCANE
LEAGUE OF LEGENDS
FORTICHE


Copyright Riot Games
Used here with permission. All rights reserved.
ARCANE
LEAGUE OF LEGENDS
FORTICHE

Elementos dentro del cuarto:

- Escritorio
- Sillas de estudio
- Sofa de 2 personas, mesa central
- Cosas sobre la mesa como los libros, la lampara y demás.
- Armario

- Desarrollo del proyecto:

El modelo desarrollado incluye los aspectos principales de la taberna.

- Fachada:

La fachada esta compuesta por ventanas y tuberías por las que pasa el vapor para proporcionar la energía. Además del cartel que anuncia el nombre de este en lo alto.

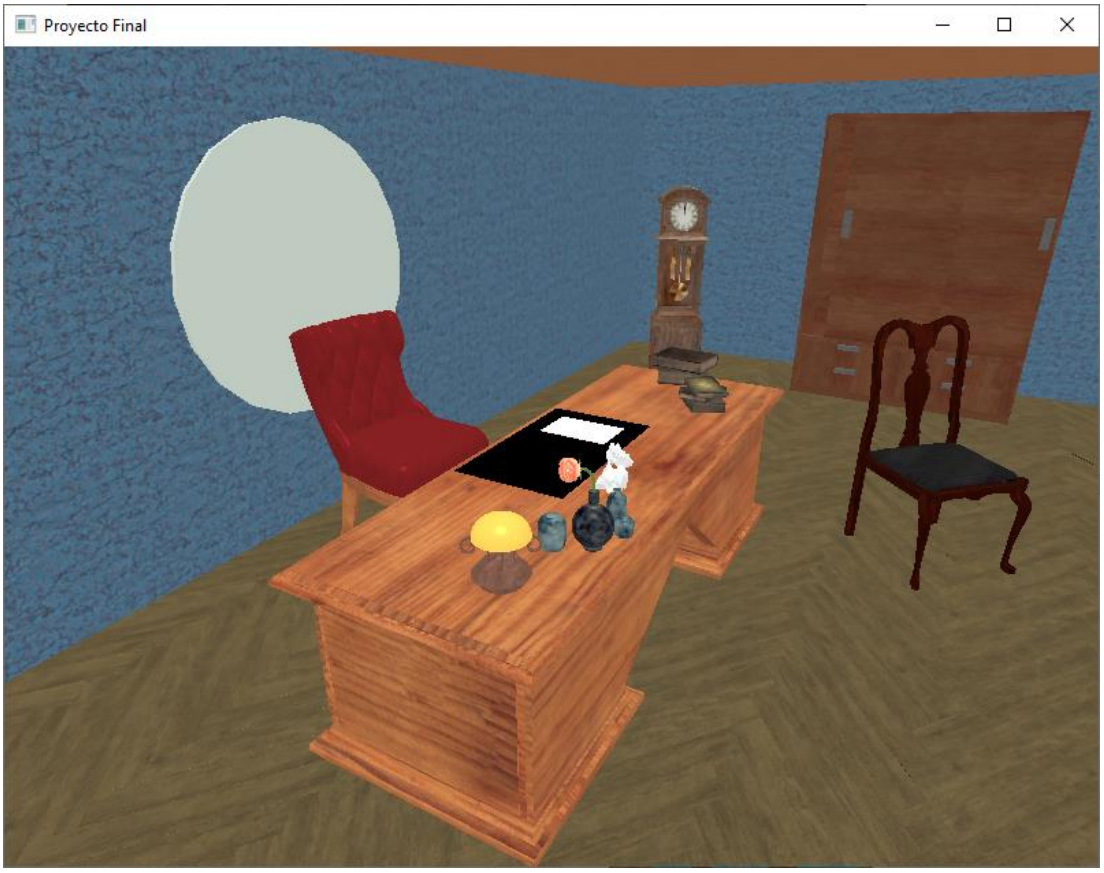
- Objetos inanimados:

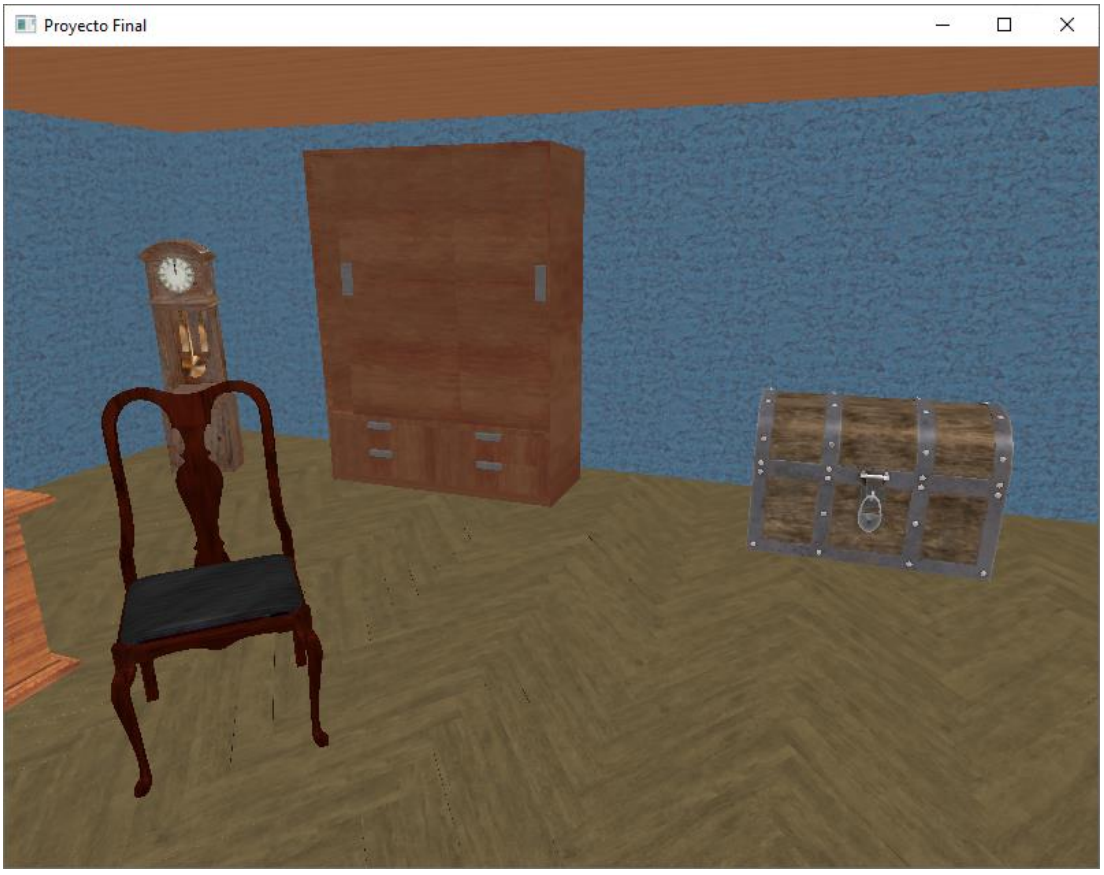
Dentro del proyecto existen objetos que no tienen ningún tipo de movimiento por lo que se les considera como inanimados. Estos son los siguientes y componen parte de la decoración del interior.

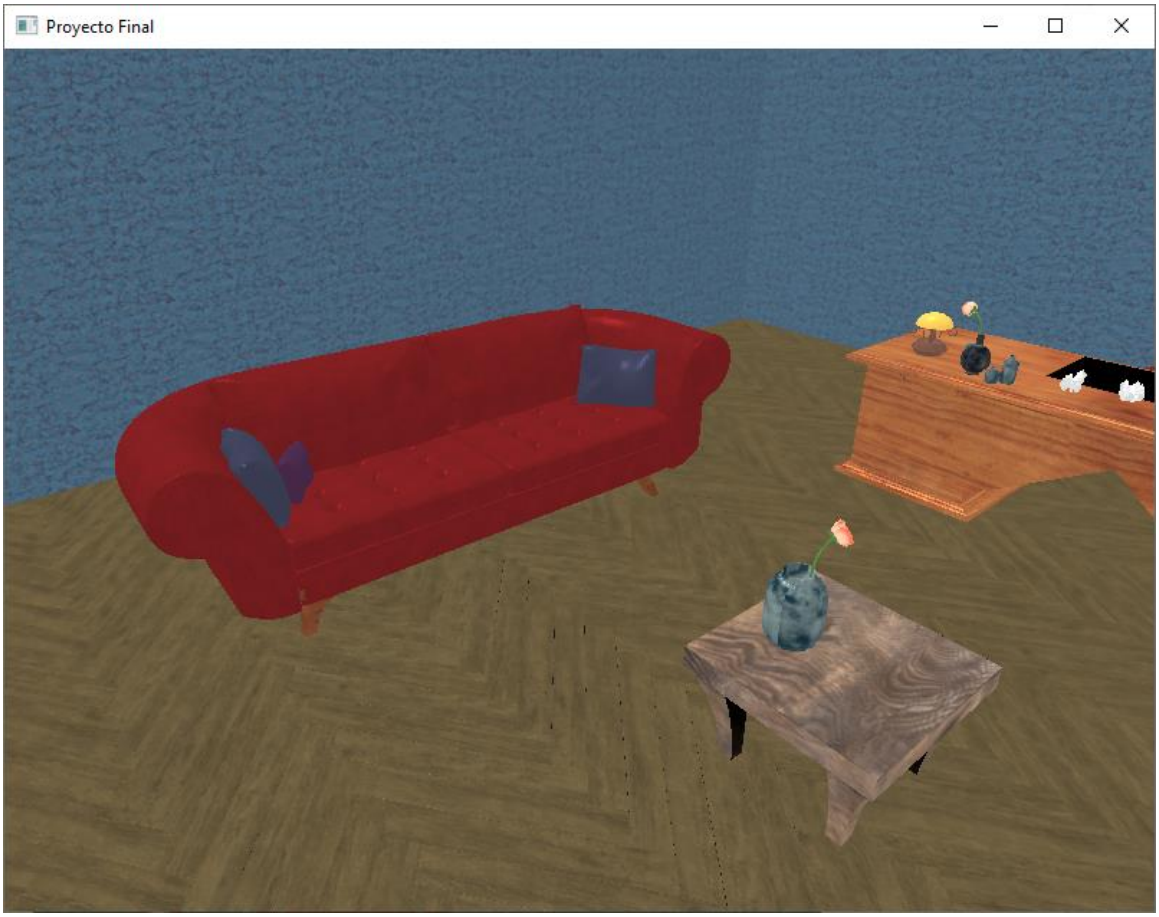
- Barriles, mesas, sillones, bancos, guantes, Hookah, Barra de tragos.

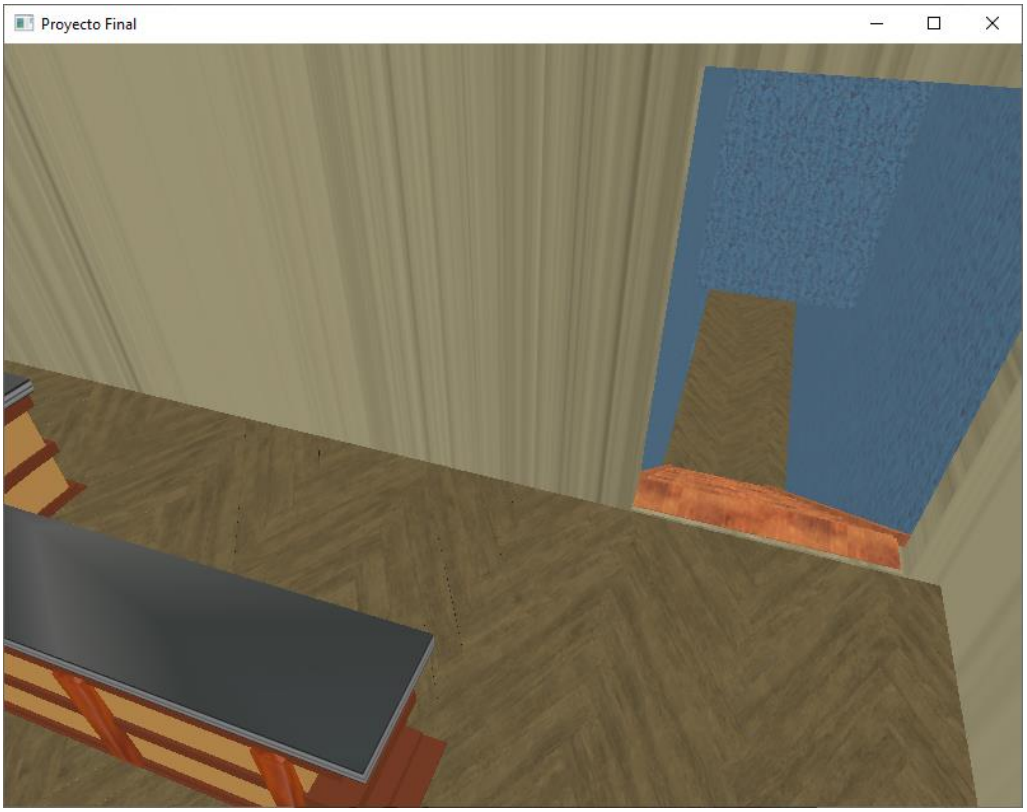
- Sección Proyecto de teoría:

Describiendo los objetos inanimados dentro del estudio que se desarrollaron y adjuntando evidencia:


Proyecto Final


Proyecto Final


Proyecto Final


Proyecto Final

- Animaciones:

Se reutilizaron las animaciones ya creadas en el proyecto de laboratorio y se enlistan a continuación. Las animaciones son aquellas que se activan luego de presionar una tecla. Se utilizaron 2 clasificaciones diferentes para ellas donde la primera son sencillas y las siguientes son complejas.

Animación sencilla:

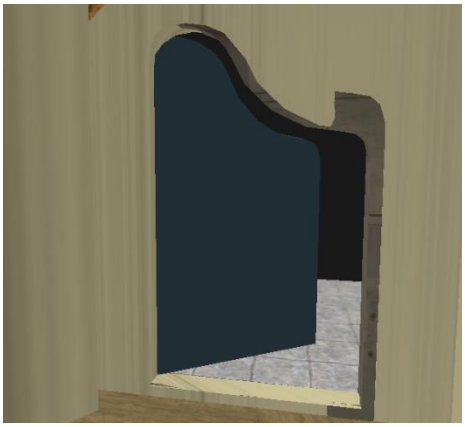
1. El disco de la rocola:

Al presionar la letra U se activa la función que nos dará el proceso que aumentará de manera simple el valor de la rotación del disco para que simule como si se estuviera reproduciendo música.

2. Esfera de disco en el techo:

Al presionar la letra P se mantendrá girando la bola reflectante en el techo, si se vuelve a presionar esta se parará de inmediato.

3. Puerta:



Si se presiona la letra K se moverá la puerta de entrada, abriéndose y cerrándose.

Animación compleja:

4. Diana:

La animación compleja de la Diana consiste en activar y desactivar estados al modo del coche de la práctica 11 para que al momento de llegar a un punto cambie de dirección.

La Diana se moverá como si recorriera un rectángulo para que se le pueda atinar en movimiento. Para activar el animación se debe oprimir la i y si se presiona la o se detendrá en cualquier punto.

5. Changuito con platillos:

La animación compleja del mono tocando los platillos se hace utilizando una técnica conocida como Keyframes. Para que este funcione se debe presionar la L cada vez que se desee iniciar el movimiento de aplaudir.

• Diagrama de Gantt:

	Abril				Mayo							
Actividades	19	22	26	29	3	5	10	12	17	19	24	26
Imagen de referencia												
Creación de la fachada												
Carga y diseño de modelados												
Implementación en opengl												
Documentación												

En este diagrama de Gantt no se incluye el tiempo que se utilizó para realizar las animaciones porque se elaboraron en el proyecto previo.

• Análisis de costos:

Considerando los gastos que se deben de realizar para entregar el proyecto eficientemente, el tiempo de realización, la mano de obra y las licencias se realizó esta estimación o presupuesto para el cliente.

Considerando que en la realización de este proyecto se cubrieron 3 roles principales por una única persona se buscará que la cotización del desarrollo sea equivalente al trabajo hecho en el mismo.

Se cubrieron las plazas del Project manager, del 3D Designer y Software Developer ya que se trabajó la parte gráfica del modelado de objetos, luego se configuró en el ambiente de desarrollo opengl que finalmente se completo con la coordinación y documentación del Project manager.

Siguiendo el diagrama de Gantt se observa como se repartió el trabajo en los días asignados. De esta forma cada uno tiene los siguientes presupuestos.

- Mano de Obra

➢ Project Manager: Realizó la planificación desde la imagen de referencia, la coordinación entre el diseño y código, además de la documentación. Percibe un sueldo por hora de $ 110, si este trabajó todos los días marcados del diagrama, 12 días, un total de 4 horas cada día, entonces su sueldo total deberá de ser igual a $ 5280.

➢ 3D Designer: Este rol se vió involucrado específicamente en la carga y diseño de modelados por lo que trabajó los días señalados en la tabla, un total de 8 días. Su sueldo por hora es de $ 65, pero se requiero su ayuda a tiempo completo (8 horas) estos días. Por lo que la cotización de sus servicios es igual a $ 4160.

➢ Software Developer: El desarrollador de software se necesitó para cargar y editar todos los modelos en opengl por lo que solo se le requirió 6 días solo 5 horas al día en los que cobró a $ 150 la hora. Así que el cobrará un total de $4500.

De esta forma el costo total por mano de obra es $ 13940 con lo que se conseguirá realizar el proyecto en 12 días en su totalidad.

- Holgura:

Si consideramos una holgura del 5% y utilidad al 50 % entonces obtenemos el presupuesto de $ 20,910.

- Costos fijos:

Aunado a esto se añaden los costos fijos de servicios:

En primer lugar, el internet que si al mes el precio del paquete cotizado es de 480 entonces por los 12 días de utilización del servicio será de $ 200.

De luz eléctrica por 12 días son aproximadamente $ 150 .

- Licencias y desgaste de equipo:

Debido a que las licencias son de por vida y que son parte de la empresa no se le pueden cobrar completamente al cliente, pero se cobra parte de este junto con los precios por desgaste de la materia prima como lo es la computadora de escritorio que requiere mantenimiento por el uso constante que se le da. Debido a este se cobra un 1% extra al costo de mano de obra a cada proyecto que se realiza. De esta forma se añaden $139.4 al costo total.

- Costo total:

Sumando cada uno de los apartados obtenemos la cotización por $21,399.4 MXN.

Mano de obra		Costos Fijos	Licencias y desgaste de equipo	Costo total
5280 + 4160 + 4500 = 13,940		200 + 150	13,940 x (.01)	
Añadiendo holgura	(1.5) x 13,940 = 20910	350	139.4	21,399.4

Documentación del código:

Podemos dividir el código en tres partes fácilmente reconocibles y podemos poner el código que explica perfectamente que hace cierta parte del código para que quede claro como funciona. Así mismo, se colocan las partes más relevantes del código, de las cuales se modificaron:

```
1) /*Desarrolló 316166496 - Díaz Alan */
2) //Proyecto Final de Lab. Computación Gráfica
3) //Proyecto Final: Modelado en 3D del bar "The Last Drop" de la serie animada Arcane
4)
5)
6) //Library import - Importación de las librerias de opengl
7) #include <iostream>
8) #include <cmath>
9) #include <GL/glew.h> // GLEW
10)#include <GLFW/glfw3.h>  // GLFW
11)#include "stb_image.h"// Other Libs
12)#include <glm/glm.hpp>
13)#include <glm/gtc/matrix_transform.hpp>
14)#include <glm/gtc/type_ptr.hpp> // GLM Mathematics
15)#include "SOIL2/SOIL2.h"//Load Models
16)
17)// Libraries for the control of camera, shaders, and correct texturization
18)// Librerias para el control de camara, texturizado y añadido de shaders
19)#include "Shader.h"
20)#include "Camera.h"
21)#include "Model.h"
22)#include "Texture.h"
23)#include "modelAnim.h"
24)
25)// Function declarations - Declaracion de funciones
26)void KeyCallback(GLFWwindow *window, int key, int scancode, int action, int mode); //Función
   que detecta el uso de una tecla - Key using function
27)void MouseCallback(GLFWwindow *window, double xPos, double yPos); //Detección del mouse -
   Mouse Detector
28)void DoMovement(); //Función activa algún proceso mientras se presione la tecla - While
   pressing a key a process is done
29)void animacion(); //Contiene los procesos para animación de objetos - Holds the instructions
   for the animation process.
30)
31)// Establece las dimensiones de la ventana - Window dimensions
32)const GLuint WIDTH = 800, HEIGHT = 600; //Ancho y alto
33)int SCREEN_WIDTH, SCREEN_HEIGHT;
34)
35)// Valores predeterminados para la camara - Camera settings at the beginning
36)Camera  camera(glm::vec3(-100.0f, 2.0f, -45.0f));
37)GLfloat lastX = WIDTH / 2.0;
38)GLfloat lastY = HEIGHT / 2.0;
39)bool keys[1024];
40)bool firstMouse = true;
41)float range = 0.0f;
42)float rot = 90.0f;
43)
44)//Variables para aumentar o disminuir la rotacion de algún objeto animado
45)// Values which save the rotation degrees for some objects in the animation process
46)float rotpuerta = 0; //rotar la puerta - Rotate door
47)float rotdisco = 0; //Rotar esfera reflectiva - Rotate disco Ball
48)float rotJuke = 0; //Rotar disco en rocola - Rotate Vinil disq
49)float movCamera = 0.0f;
50)
51)//Variables para designar el movimiento de la Diana en cada eje
52)// Bull'seye parameters to move it on real time
53)float movKitY = 0.0;
```

```
54)float movKitZ = 0.0;
55)float rotKit = 0.0;
56)
57)//Boleanos para activar o desactivar un estado en el recorrido de la diana.
58)//Booleans to switch between states for the Bull'seye movement.
59)bool circuito = false;
60)bool recorrido1 = true;
61)bool recorrido2 = false;
62)bool recorrido3 = false;
63)bool recorrido4 = false;
64)
65)// Atributos de la luz en el medio - Light attributes
66)glm::vec3 lightPos(0.0f, 0.0f, 0.0f);
67)glm::vec3 PosIni(-95.0f, 1.0f, -45.0f);
68)glm::vec3 lightDirection(0.0f, -1.0f, -1.0f);
69)
70)bool active;
71)
72)
73)// Deltatime
74)GLfloat deltaTime = 100.0f;     // Time between current frame and last frame
75)GLfloat lastFrame = 0.0f;       // Time of last frame
76)
77)// Variables de posición inicial para la implementacion de animacion Keyframes
78)// KeyFrames animation process has initial axis values
79)float posX =PosIni.x, posY = PosIni.y, posZ = PosIni.z, rotRodIzq = 0, rotManIzq;
80)float rotManDer;
81)
82)
83)//Definición de frames máximos posibles a guardar
84)// Max frames for the technique
85)#define MAX_FRAMES 9
86)int i_max_steps = 190;
87)int i_curr_steps = 0;
88)typedef struct _frame
89){
90)       //Variables para GUARDAR Key Frames
91)       float posX;        //Variable para PosicionX
92)       float posY;        //Variable para PosicionY
93)       float posZ;        //Variable para PosicionZ
94)       float incX;        //Variable para IncrementoX
95)       float incY;        //Variable para IncrementoY
96)       float incZ;        //Variable para IncrementoZ
97)       float rotInc;
98)       float rotInc3;
99)       float rotInc4;
100)            float rotManIzq; //Variable para Incremento en la rotación Mano Izq --
   Increments rotation value for the left hand of the monkey
101)            float rotManDer;//Variable para Incremento en la rotación Mano Der -- Increments
   rotation value for the right hand of the monkey
102)
103)     }FRAME;
104)
105)     FRAME KeyFrame[MAX_FRAMES];
106)     int FrameIndex = 0;            //Indice de la tupla para los frames - frameindex
   holder.
107)     bool play = false;
108)     bool aux = false;
109)     bool puerta = false;
110)     bool disco = false;
111)     bool Juke = false;
112)     int playIndex = 0;
113)
114)     // Posiciones de las luces de punto - Positions of the point lights
```

```
115)     glm::vec3 pointLightPositions[] = {
116)           glm::vec3(posX,posY,posZ),
117)           glm::vec3(0,0,0),
118)           glm::vec3(0,0,0),
119)           glm::vec3(0,0,0)
120)     };
121)
122)     glm::vec3 LightP1;
123)
124)
125)
126)     //FUnción para guardar el estado de la figura en el preciso momento en el que se mande
   a llamar.
127)     // SaveFrame function as its name says, saves the KeyFrame in that moment.
128)     void saveFrame(void)
129)     {
130)
131)           printf("posx %f\n", posX);
132)
133)           KeyFrame[FrameIndex].posX = posX; //Guardamos la posición con respecto al eje -
   Saves the axis position
134)           KeyFrame[FrameIndex].posY = posY;
135)           KeyFrame[FrameIndex].posZ = posZ;
136)
137)           KeyFrame[FrameIndex].rotManIzq = rotManIzq; //GUardamos la rotación de la figura
   - We save the new rotation of the hand
138)           KeyFrame[FrameIndex].rotManDer = rotManDer;
139)
140)           FrameIndex++;
141)     }
142)
143)
144)     // Regresa la posición actual de la pieza a la original- returns every movement in the
   figure to the beginning preset.
145)     void resetElements(void)
146)     {
147)           posX = KeyFrame[0].posX;
148)           posY = KeyFrame[0].posY;
149)           posZ = KeyFrame[0].posZ;
150)
151)           rotManIzq = KeyFrame[0].rotManIzq;
152)           rotManDer = KeyFrame[0].rotManDer;
153)
154)     }
155)
156)     // FUncion de interpolación para calcular las posiciones intermedias entre la primera
   y segunda ubicación - Interpolation of first and last position
157)     void interpolation(void)
158)     {
159)
160)           KeyFrame[playIndex].incX = (KeyFrame[playIndex + 1].posX -
   KeyFrame[playIndex].posX) / i_max_steps;
161)           KeyFrame[playIndex].incY = (KeyFrame[playIndex + 1].posY -
   KeyFrame[playIndex].posY) / i_max_steps;
162)           KeyFrame[playIndex].incZ = (KeyFrame[playIndex + 1].posZ -
   KeyFrame[playIndex].posZ) / i_max_steps;
163)
164)           KeyFrame[playIndex].rotInc3 = (KeyFrame[playIndex + 1].rotManIzq -
   KeyFrame[playIndex].rotManIzq) / i_max_steps;
165)           KeyFrame[playIndex].rotInc4 = (KeyFrame[playIndex + 1].rotManDer -
   KeyFrame[playIndex].rotManDer) / i_max_steps;
166)     }
167)
168)
```

```
169)
170)
171)      int main()
172)      {
173)            // Inicializa la librería Init GLFW
174)            glfwInit();
175)
176)
177)            // Creando la ventana de ejecución - Create a GLFWwindow object that we can use
   for GLFW's functions
178)            GLFWwindow* window = glfwCreateWindow(WIDTH, HEIGHT, "Proyecto Final", nullptr,
   nullptr);
179)
180)            // Si no se puede crear la ventana por alguna razón devolverá un fallo y acabará
   el programa
181)            // In case the window isn't created it crashes
182)            if (nullptr == window)
183)            {
184)                  std::cout << "Failed to create GLFW window" << std::endl;
185)                  glfwTerminate();
186)
187)                  return EXIT_FAILURE;
188)            }
189)
190)            glfwMakeContextCurrent(window);
191)
192)            glfwGetFramebufferSize(window, &SCREEN_WIDTH, &SCREEN_HEIGHT);
193)
194)            // Set the required callback functions
195)            glfwSetKeyCallback(window, KeyCallback);
196)            glfwSetCursorPosCallback(window, MouseCallback);
197)            printf("%f", glfwGetTime());
198)
199)            // GLFW Options
200)            glfwSetInputMode(window, GLFW_CURSOR, GLFW_CURSOR_DISABLED);
201)
202)
203)            // Set this to true so GLEW knows to use a modern approach to retrieving
   function pointers and extensions
204)            glewExperimental = GL_TRUE;
205)            // Initialize GLEW to setup the OpenGL Function pointers
206)            if (GLEW_OK != glewInit())
207)            {
208)                  std::cout << "Failed to initialize GLEW" << std::endl;
209)                  return EXIT_FAILURE;
210)            }
211)
212)            // Define the viewport dimensions
213)            glViewport(0, 0, SCREEN_WIDTH, SCREEN_HEIGHT);
214)
215)            // OpenGL options
216)            glEnable(GL_DEPTH_TEST);
217)            glEnable(GL_BLEND);
218)            //glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
219)
220)
221)            //Importamos las librerias para las luces - Import light shaders
222)            Shader lightingShader("Shaders/lighting.vs", "Shaders/lighting.frag");
223)            Shader lampShader("Shaders/lamp.vs", "Shaders/lamp.frag");
224)
225)            // Cargamos los modelos y los nombramos - We load each 3d model and assign it to
   a model name.
226)            Model Mesa_circular((char*)"Models/Mesa_circular/Mesa_circular.obj");
227)            Model Hookah((char*)"Models/Hookah/Hookah.obj");
```

```
228)            Model Sofa((char*)"Models/Sofa/Sofa.obj");
229)            Model Guante((char*)"Models/Guante/Guante.obj");
230)            Model Luces((char*)"Models/House/Luces_esfericas.obj");
231)            Model palco((char*)"Models/House2/palco.obj");
232)            Model tuberias((char*)"Models/House2/tuberias.obj");
233)            Model wall((char*)"Models/House3/wall_2.obj");
234)            Model wall1((char*)"Models/House3/wall1.obj");
235)            Model wall2((char*)"Models/House3/wall2.obj");
236)            Model wall3((char*)"Models/Houseall/puerta_cortada.obj");
237)            Model wallcita((char*)"Models/Houseall/paredcita.obj");
238)            Model vapor((char*)"Models/House3/vapor.obj");
239)            Model altar((char*)"Models/House4/altar.obj");
240)            Model beer((char*)"Models/HouseAll/beer.obj");
241)            Model ventana((char*)"Models/HouseAll/ventana.obj");
242)            Model piso((char*)"Models/HouseAll/piso.obj");
243)            Model Puerta2((char*)"Models/HouseAll/puerta_azul.obj");
244)            Model techo((char*)"Models/Techo/techo.obj");
245)            Model Jukebox((char*)"Models/Jukebox/Jukebox.obj");
246)            Model Disque((char*)"Models/Jukebox/Jukebox_disque.obj");
247)            Model Luz_techo((char*)"Models/Luz_techo/Luz_techo.obj");
248)            Model Barra((char*)"Models/Barra/Barra.obj");
249)            Model Stool((char*)"Models/Stool/Stool.obj");
250)            Model Monkey((char*)"Models/Changuito/Monkey.obj");
251)            Model Monkey_izq((char*)"Models/Changuito/Monkey_Izq.obj");
252)            Model Monkey_der((char*)"Models/Changuito/Monkey_Der.obj");
253)            Model Diana((char*)"Models/Diana/Diana.obj");
254)            Model ball((char*)"Models/Disco/ball.obj");
255)            Model Chain((char*)"Models/Disco/Chain.obj");
256)            Model Barril((char*)"Models/Barriles/Barril.obj");
257)
258)            // Estudio
259)            Model baul((char*)"Models/estudio/paredes/baul.obj");
260)            Model closet((char*)"Models/estudio/paredes/closet.obj");
261)            Model escalera((char*)"Models/estudio/paredes/escalera.obj");
262)            Model escritorio((char*)"Models/estudio/paredes/escritorio.obj");
263)            Model flor((char*)"Models/estudio/paredes/flor.obj");
264)            Model lampara((char*)"Models/estudio/paredes/lampara.obj");
265)            Model libro((char*)"Models/estudio/paredes/libro.obj");
266)            Model mesa((char*)"Models/estudio/paredes/mesa.obj");
267)            Model papel((char*)"Models/estudio/paredes/papel.obj");
268)            Model pared((char*)"Models/estudio/paredes/pared.obj");
269)            Model piso2((char*)"Models/estudio/paredes/piso.obj");
270)            Model reloj((char*)"Models/estudio/paredes/reloj.obj");
271)            Model silla((char*)"Models/estudio/paredes/silla.obj");
272)            Model silla2((char*)"Models/estudio/paredes/silla2.obj");
273)            Model sofa((char*)"Models/estudio/paredes/sofa.obj");
274)            Model techo2((char*)"Models/estudio/paredes/techo.obj");
275)
276)
277)            // Build and compile our shader program
278)
279)            //Inicialización de KeyFrames - KeyFrame Initialize
280)            // Cada Keyframe podra guardar la posicion o rotacion de la figura por cada
   indice
281)            // Saves each frame the times the key has been used
282)            for(int i=0; i<MAX_FRAMES; i++)
283)            {
284)                  KeyFrame[i].posX = 0;
285)                  KeyFrame[i].incX = 0;
286)                  KeyFrame[i].incY = 0;
287)                  KeyFrame[i].incZ = 0;
288)                  KeyFrame[i].rotManIzq = 0;
289)                  KeyFrame[i].rotManDer = 0;
290)            }
```

```
291)
292)
293)            // Establece las posiciones y normales de algunos apuntadores
294)            // Set up vertex data (and buffer(s)) and attribute pointers
295)            GLfloat vertices[] =
296)            {
297)                  // Positions            // Normals              // Texture Coords
298)                  -0.5f, -0.5f, -0.5f,    0.0f,  0.0f, -1.0f,     0.0f,  0.0f,
299)                  0.5f, -0.5f, -0.5f,     0.0f,  0.0f, -1.0f,     1.0f,  0.0f,
300)                  0.5f,  0.5f, -0.5f,     0.0f,  0.0f, -1.0f,     1.0f,  1.0f,
301)                  0.5f,  0.5f, -0.5f,     0.0f,  0.0f, -1.0f,     1.0f,  1.0f,
302)                  -0.5f,  0.5f, -0.5f,    0.0f,  0.0f, -1.0f,     0.0f,  1.0f,
303)                  -0.5f, -0.5f, -0.5f,    0.0f,  0.0f, -1.0f,     0.0f,  0.0f,
304)
305)                  -0.5f, -0.5f,  0.5f,    0.0f,  0.0f,  1.0f,     0.0f,  0.0f,
306)                  0.5f, -0.5f,  0.5f,     0.0f,  0.0f,  1.0f,     1.0f,  0.0f,
307)                  0.5f,  0.5f,  0.5f,     0.0f,  0.0f,  1.0f,     1.0f,  1.0f,
308)                  0.5f,  0.5f,  0.5f,     0.0f,  0.0f,  1.0f,        1.0f,  1.0f,
309)                  -0.5f,  0.5f,  0.5f,    0.0f,  0.0f,  1.0f,     0.0f,  1.0f,
310)                  -0.5f, -0.5f,  0.5f,    0.0f,  0.0f,  1.0f,     0.0f,  0.0f,
311)
312)                  -0.5f,  0.5f,  0.5f,    -1.0f,  0.0f,  0.0f,    1.0f,  0.0f,
313)                  -0.5f,  0.5f, -0.5f,    -1.0f,  0.0f,  0.0f,    1.0f,  1.0f,
314)                  -0.5f, -0.5f, -0.5f,    -1.0f,  0.0f,  0.0f,    0.0f,  1.0f,
315)                  -0.5f, -0.5f, -0.5f,    -1.0f,  0.0f,  0.0f,    0.0f,  1.0f,
316)                  -0.5f, -0.5f,  0.5f,    -1.0f,  0.0f,  0.0f,    0.0f,  0.0f,
317)                  -0.5f,  0.5f,  0.5f,    -1.0f,  0.0f,  0.0f,    1.0f,  0.0f,
318)
319)                  0.5f,  0.5f,  0.5f,     1.0f,  0.0f,  0.0f,     1.0f,  0.0f,
320)                  0.5f,  0.5f, -0.5f,     1.0f,  0.0f,  0.0f,     1.0f,  1.0f,
321)                  0.5f, -0.5f, -0.5f,     1.0f,  0.0f,  0.0f,     0.0f,  1.0f,
322)                  0.5f, -0.5f, -0.5f,     1.0f,  0.0f,  0.0f,     0.0f,  1.0f,
323)                  0.5f, -0.5f,  0.5f,     1.0f,  0.0f,  0.0f,     0.0f,  0.0f,
324)                  0.5f,  0.5f,  0.5f,     1.0f,  0.0f,  0.0f,     1.0f,  0.0f,
325)
326)                  -0.5f, -0.5f, -0.5f,    0.0f, -1.0f,  0.0f,     0.0f,  1.0f,
327)                  0.5f, -0.5f, -0.5f,     0.0f, -1.0f,  0.0f,     1.0f,  1.0f,
328)                  0.5f, -0.5f,  0.5f,     0.0f, -1.0f,  0.0f,     1.0f,  0.0f,
329)                  0.5f, -0.5f,  0.5f,     0.0f, -1.0f,  0.0f,     1.0f,  0.0f,
330)                  -0.5f, -0.5f,  0.5f,    0.0f, -1.0f,  0.0f,     0.0f,  0.0f,
331)                  -0.5f, -0.5f, -0.5f,    0.0f, -1.0f,  0.0f,     0.0f,  1.0f,
332)
333)                  -0.5f,  0.5f, -0.5f,    0.0f,  1.0f,  0.0f,     0.0f,  1.0f,
334)                  0.5f,  0.5f, -0.5f,     0.0f,  1.0f,  0.0f,     1.0f,  1.0f,
335)                  0.5f,  0.5f,  0.5f,     0.0f,  1.0f,  0.0f,     1.0f,  0.0f,
336)                  0.5f,  0.5f,  0.5f,     0.0f,  1.0f,  0.0f,     1.0f,  0.0f,
337)                  -0.5f,  0.5f,  0.5f,    0.0f,  1.0f,  0.0f,     0.0f,  0.0f,
338)                  -0.5f,  0.5f, -0.5f,    0.0f,  1.0f,  0.0f,     0.0f,  1.0f
339)            };
340)
341)
342)            //Se crean las caras del skybox para añadir el fondo - skybox vertex arrays are
    created
343)            GLfloat skyboxVertices[] = {
344)                  // Positions
345)                  -1.0f,  1.0f, -1.0f,
346)                  -1.0f, -1.0f, -1.0f,
347)                  1.0f, -1.0f, -1.0f,
348)                  1.0f, -1.0f, -1.0f,
349)                  1.0f,  1.0f, -1.0f,
350)                  -1.0f,  1.0f, -1.0f,
351)
352)                  -1.0f, -1.0f,  1.0f,
353)                  -1.0f, -1.0f, -1.0f,
```

```
354)                    -1.0f,  1.0f, -1.0f,
355)                    -1.0f,  1.0f, -1.0f,
356)                    -1.0f,  1.0f,  1.0f,
357)                    -1.0f, -1.0f,  1.0f,
358)
359)                    1.0f, -1.0f, -1.0f,
360)                    1.0f, -1.0f,  1.0f,
361)                    1.0f,  1.0f,  1.0f,
362)                    1.0f,  1.0f,  1.0f,
363)                    1.0f,  1.0f, -1.0f,
364)                    1.0f, -1.0f, -1.0f,
365)
366)                    -1.0f, -1.0f,  1.0f,
367)                    -1.0f,  1.0f,  1.0f,
368)                    1.0f,  1.0f,  1.0f,
369)                    1.0f,  1.0f,  1.0f,
370)                    1.0f, -1.0f,  1.0f,
371)                    -1.0f, -1.0f,  1.0f,
372)
373)                    -1.0f,  1.0f, -1.0f,
374)                    1.0f,  1.0f, -1.0f,
375)                    1.0f,  1.0f,  1.0f,
376)                    1.0f,  1.0f,  1.0f,
377)                    -1.0f,  1.0f,  1.0f,
378)                    -1.0f,  1.0f, -1.0f,
379)
380)                    -1.0f, -1.0f, -1.0f,
381)                    -1.0f, -1.0f,  1.0f,
382)                    1.0f, -1.0f, -1.0f,
383)                    1.0f, -1.0f, -1.0f,
384)                    -1.0f, -1.0f,  1.0f,
385)                    1.0f, -1.0f,  1.0f
386)            };
387)
388)
389)            GLuint indices[] =
390)            {  // Note that we start from 0!
391)                  0,1,2,3,
392)                  4,5,6,7,
393)                  8,9,10,11,
394)                  12,13,14,15,
395)                  16,17,18,19,
396)                  20,21,22,23,
397)                  24,25,26,27,
398)                  28,29,30,31,
399)                  32,33,34,35
   };
   // Positions all containers
   glm::vec3 cubePositions[] = {
         glm::vec3(0.0f,  0.0f,  0.0f),
         glm::vec3(2.0f,  5.0f, -15.0f),
         glm::vec3(-1.5f, -2.2f, -2.5f),
         glm::vec3(-3.8f, -2.0f, -12.3f),
         glm::vec3(2.4f, -0.4f, -3.5f),
         glm::vec3(-1.7f,  3.0f, -7.5f),
         glm::vec3(1.3f, -2.0f, -2.5f),
         glm::vec3(1.5f,  2.0f, -2.5f),
         glm::vec3(1.5f,  0.2f, -1.5f),
         glm::vec3(-1.3f,  1.0f, -1.5f)
   };


   // First, set the container's VAO (and VBO)
   GLuint VBO, VAO, EBO;
```

```
	glGenVertexArrays(1, &VAO);
	glGenBuffers(1, &VBO);
	glGenBuffers(1, &EBO);

	glBindVertexArray(VAO);
	glBindBuffer(GL_ARRAY_BUFFER, VBO);
	glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW);

	glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EBO);
	glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(indices), indices, GL_STATIC_DRAW);

	// Position attribute
	glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (GLvoid *)0);
	glEnableVertexAttribArray(0);
	// Normals attribute
	glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (GLvoid *)(3 *
sizeof(GLfloat)));
	glEnableVertexAttribArray(1);
	// Texture Coordinate attribute
	glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (GLvoid *)(6 *
sizeof(GLfloat)));
	glEnableVertexAttribArray(2);
	glBindVertexArray(0);

	// Then, we set the light's VAO (VBO stays the same. After all, the vertices are the same
for the light object (also a 3D cube))
	GLuint lightVAO;
	glGenVertexArrays(1, &lightVAO);
	glBindVertexArray(lightVAO);
	// We only need to bind to the VBO (to link it with glVertexAttribPointer), no need to fill
it; the VBO's data already contains all we need.
	glBindBuffer(GL_ARRAY_BUFFER, VBO);
	// Set the vertex attributes (only position data for the lamp))
	glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (GLvoid *)0); // Note
that we skip over the other data in our buffer object (we don't need the normals/textures, only
positions).
	glEnableVertexAttribArray(0);
	glBindVertexArray(0);



	//Establecemos los frames en cada indice de KeyFrames - Set Monkey movement for every inder
in KeyFrames function
	rotManDer = -10;
	rotManIzq = 10;
	saveFrame();
	rotManDer = 18;
	rotManIzq = -18;
	saveFrame();
	rotManDer = -10;
	rotManIzq = 10;
	saveFrame();
	rotManDer = 18;
	rotManIzq = -18;
	saveFrame();
	rotManDer = -10;
	rotManIzq = 10;
	saveFrame();

	// Carga las texturas de nuestro fondo - Load textures of the skybox
	vector<const GLchar*> faces;
	faces.push_back("SkyBox/right.tga");
	faces.push_back("SkyBox/left.tga");
	faces.push_back("SkyBox/top.tga");
```

```
	faces.push_back("SkyBox/bottom.tga");
	faces.push_back("SkyBox/back.tga");
	faces.push_back("SkyBox/front.tga");

	GLuint cubemapTexture = TextureLoading::LoadCubemap(faces);

	glm::mat4 projection = glm::perspective(camera.GetZoom(), (GLfloat)SCREEN_WIDTH /
(GLfloat)SCREEN_HEIGHT, 0.1f, 1000.0f);

	// Game loop
	while (!glfwWindowShouldClose(window))
	{


		// Calculate deltatime of current frame
		GLfloat currentFrame = glfwGetTime();
		deltaTime = currentFrame - lastFrame;
		lastFrame = currentFrame;

		// Check if any events have been activiated (key pressed, mouse moved etc.) and call
corresponding response functions
		glfwPollEvents();
		DoMovement();
		animacion();


		// Clear the colorbuffer
		glClearColor(0.1f, 0.1f, 0.1f, 1.0f);
		glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);


		// Use cooresponding shader when setting uniforms/drawing objects
		lightingShader.Use();
		GLint viewPosLoc = glGetUniformLocation(lightingShader.Program, "viewPos");
		glUniform3f(viewPosLoc, camera.GetPosition().x, camera.GetPosition().y,
camera.GetPosition().z);
		// Set material properties
		glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"),
32.0f);
		// == ===========================
		// Here we set all the uniforms for the 5/6 types of lights we have. We have to set
them manually and index
		// the proper PointLight struct in the array to set each uniform variable.
		// // == ===========================
		// Aqui debemos de establecer los 5 de 6 tipos de luces que tenemos. Lo hacemos
manualmente para establecer cada una de las variables.
		// == ===========================
		// Directional light
		glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.direction"), -0.2f,
-1.0f, -0.3f);
		glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.ambient"), 1.0f,
1.0f, 1.0f);
		glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.diffuse"), 0.2f,
0.2f, 0.2f);
		glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.specular"), 0.5f,
0.5f, 0.5f);


		// Point light 1
		glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].position"),
pointLightPositions[0].x, pointLightPositions[0].y, pointLightPositions[0].z);
		glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].ambient"),
0.5f, 0.5f, 0.5f);
```

```
            glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].diffuse"),
LightP1.x, LightP1.y, LightP1.z);
            glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].specular"),
LightP1.x, LightP1.y, LightP1.z);
            glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[0].constant"),
1.0f);
            glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[0].linear"),
0.09f);
            glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[0].quadratic"),
0.032f);



            // Point light 2
            glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[1].position"),
pointLightPositions[1].x, pointLightPositions[1].y, pointLightPositions[1].z);
            glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[1].ambient"),
0.05f, 0.05f, 0.05f);
            glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[1].diffuse"),
1.0f, 1.0f, 0.0f);
            glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[1].specular"),
1.0f, 1.0f, 0.0f);
            glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[1].constant"),
1.0f);
            glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[1].linear"),
0.09f);
            glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[1].quadratic"),
0.032f);

            // Point light 3
            glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[2].position"),
pointLightPositions[2].x, pointLightPositions[2].y, pointLightPositions[2].z);
            glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[2].ambient"),
0.05f, 0.05f, 0.05f);
            glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[2].diffuse"),
0.0f, 1.0f, 1.0f);
            glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[2].specular"),
0.0f, 1.0f, 1.0f);
            glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[2].constant"),
1.0f);
            glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[2].linear"),
0.09f);
            glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[2].quadratic"),
0.032f);

            // Point light 4
            glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[3].position"),
pointLightPositions[3].x, pointLightPositions[3].y, pointLightPositions[3].z);
            glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[3].ambient"),
0.05f, 0.05f, 0.05f);
            glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[3].diffuse"),
1.0f, 0.0f, 1.0f);
            glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[3].specular"),
1.0f, 0.0f, 1.0f);
            glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[3].constant"),
1.0f);
            glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[3].linear"),
0.09f);
            glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[3].quadratic"),
0.032f);

            // SpotLight
            glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight.position"),
camera.GetPosition().x, camera.GetPosition().y, camera.GetPosition().z);
```

```
            glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight.direction"),
camera.GetFront().x, camera.GetFront().y, camera.GetFront().z);
            glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight.ambient"), 0.0f,
0.0f, 0.0f);
            glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight.diffuse"), 0.0f,
0.0f, 0.0f);
            glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight.specular"), 0.0f,
0.0f, 0.0f);
            glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight.constant"), 1.0f);
            glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight.linear"), 0.09f);
            glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight.quadratic"),
0.032f);
            glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight.cutOff"),
glm::cos(glm::radians(12.5f)));
            glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight.outerCutOff"),
glm::cos(glm::radians(15.0f)));

            // Set material properties
            glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"),
32.0f);

            // Create camera transformations
            glm::mat4 view;
            view = camera.GetViewMatrix();


            // Get the uniform locations
            GLint modelLoc = glGetUniformLocation(lightingShader.Program, "model");
            GLint viewLoc = glGetUniformLocation(lightingShader.Program, "view");
            GLint projLoc = glGetUniformLocation(lightingShader.Program, "projection");

            // Pass the matrices to the shader
            glUniformMatrix4fv(viewLoc, 1, GL_FALSE, glm::value_ptr(view));
            glUniformMatrix4fv(projLoc, 1, GL_FALSE, glm::value_ptr(projection));

            // Bind diffuse map
            //glBindTexture(GL_TEXTURE_2D, texture1);*/

            // Bind specular map
            /*glActiveTexture(GL_TEXTURE1);
            glBindTexture(GL_TEXTURE_2D, texture2);*/


            glBindVertexArray(VAO);
   1)              glm::mat4 tmp = glm::mat4(1.0f); //Temp


//Carga de modelo
            // == ==========================
            // En esta parte se hace la carga de todos los modelos de nuestro proyecto. Les
aplicamos transformaciones basicas como el escalado por 2 para que sean el doble de su tamaño y se
aprecien bien.
            // Las traslaciones para que aparezcan en el punto exacto dentro o fuera de la fachada
y por último las rotaciones para que estén bien orientados.
            // == ==========================
            // In this section we load all the  models for our project. In here I had the duty to
transform this figures with the functions of scale, rotate or translate to get them
            // in the right position inside or outside the building.
            // == ==========================

            //Guante
            view = camera.GetViewMatrix();
            glm::mat4 model(1);
            tmp = model = glm::translate(model, glm::vec3(52, 10.3, -53));
```

```
model = glm::translate(model, glm::vec3(posX, posY, posZ));
model = glm::scale(model, glm::vec3(2));
model = glm::rotate(model, glm::radians(2 * rot), glm::vec3(1.0f, 0.0f, 0.0));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Guante.Draw(lightingShader);


//Mesa
view = camera.GetViewMatrix();
model = glm::mat4(1);
tmp = model = glm::translate(model, glm::vec3(85, 0.2, -80));
model = glm::translate(model,glm::vec3(posX,posY,posZ));
model = glm::scale(model, glm::vec3(5));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Mesa_circular.Draw(lightingShader);

//Mesa 2
view = camera.GetViewMatrix();
model = glm::mat4(1);
tmp = model = glm::translate(model, glm::vec3(52, 0.2, -53));
model = glm::translate(model, glm::vec3(posX, posY, posZ));
model = glm::scale(model, glm::vec3(5));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Mesa_circular.Draw(lightingShader);

//Mesa 3
view = camera.GetViewMatrix();
model = glm::mat4(1);
tmp = model = glm::translate(model, glm::vec3(110, 0.4, -45));
model = glm::translate(model, glm::vec3(posX, posY, posZ));
model = glm::scale(model, glm::vec3(5));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Mesa_circular.Draw(lightingShader);

//Monkey
view = camera.GetViewMatrix();
model = glm::mat4(1);
tmp = model = glm::translate(model, glm::vec3(110, 13.2, -45));
model = glm::translate(model, glm::vec3(posX, posY, posZ));
model = glm::scale(model, glm::vec3(.2));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Monkey.Draw(lightingShader);

//Monkey_izq
view = camera.GetViewMatrix();
model = glm::mat4(1);
tmp = model = glm::translate(model, glm::vec3(110, 13.2, -45));
model = glm::translate(model, glm::vec3(posX, posY, posZ));
model = glm::rotate(model, glm::radians(rotManIzq), glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::scale(model, glm::vec3(.2));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Monkey_izq.Draw(lightingShader);

//Monkey_der
view = camera.GetViewMatrix();
model = glm::mat4(1);
tmp = model = glm::translate(model, glm::vec3(110, 13.2, -45));
model = glm::translate(model, glm::vec3(posX, posY, posZ));
model = glm::rotate(model, glm::radians(rotManDer), glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::scale(model, glm::vec3(.2));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Monkey_der.Draw(lightingShader);

//Puerta2
```

```
view = camera.GetViewMatrix();
model = glm::mat4(1);

model = glm::translate(model, glm::vec3(posX, posY, posZ));
model = glm::translate(model, glm::vec3(-4.8, 0, 23.6));
model = glm::rotate(model, glm::radians(3 * rot), glm::vec3(0.0f, 1.0f, 0.0));
model = glm::rotate(model, glm::radians(rotpuerta), glm::vec3(0.0f, 1.0, 0.0f));
model = glm::scale(model, glm::vec3(2));//d

glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Puerta2.Draw(lightingShader);

//Diana
view = camera.GetViewMatrix();
model = glm::mat4(1);
tmp = model = glm::translate(model, glm::vec3(0, 2, -56));
model = glm::translate(model, glm::vec3(posX, posY, posZ));
model = glm::translate(model, PosIni + glm::vec3(90, movKitY, movKitZ));
model = glm::rotate(model, glm::radians(2 * rot), glm::vec3(0.0f, 1.0f, 0.0));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Diana.Draw(lightingShader);
//Stool 1
view = camera.GetViewMatrix();
model = glm::mat4(1);
tmp = model = glm::translate(model, glm::vec3(60, 4, -50));
model = glm::translate(model, glm::vec3(posX, posY, posZ));
model = glm::scale(model, glm::vec3(0.6));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Stool.Draw(lightingShader);

//Stool 2
view = camera.GetViewMatrix();
model = glm::mat4(1);
tmp = model = glm::translate(model, glm::vec3(50, 4, -70));
model = glm::translate(model, glm::vec3(posX, posY, posZ));
model = glm::scale(model, glm::vec3(0.6));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Stool.Draw(lightingShader);

//Stool 3
view = camera.GetViewMatrix();
model = glm::mat4(1);
tmp = model = glm::translate(model, glm::vec3(73, 4, -80));
model = glm::translate(model, glm::vec3(posX, posY, posZ));
model = glm::scale(model, glm::vec3(0.6));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Stool.Draw(lightingShader);

//Stool 4
view = camera.GetViewMatrix();
model = glm::mat4(1);
tmp = model = glm::translate(model, glm::vec3(98, 4.2, -45));
model = glm::translate(model, glm::vec3(posX, posY, posZ));
model = glm::scale(model, glm::vec3(0.6));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Stool.Draw(lightingShader);

//Stool 5
view = camera.GetViewMatrix();
model = glm::mat4(1);
tmp = model = glm::translate(model, glm::vec3(100, 4.2, -83));
model = glm::translate(model, glm::vec3(posX, posY, posZ));
model = glm::scale(model, glm::vec3(0.6));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
```

```
Stool.Draw(lightingShader);


//Barra
view = camera.GetViewMatrix();
model = glm::mat4(1);
tmp = model = glm::translate(model, glm::vec3(77, -0.2, -96));
model = glm::translate(model, glm::vec3(posX, posY, posZ));
model = glm::rotate(model, glm::radians(3 * rot), glm::vec3(0.0f, 1.0f, 0.0));
model = glm::scale(model, glm::vec3(2.7));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Barra.Draw(lightingShader);

//Hookah
view = camera.GetViewMatrix();
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(85, 11.3, -80));
model = glm::translate(model, glm::vec3(posX, posY, posZ));
model = glm::scale(model, glm::vec3(7));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Hookah.Draw(lightingShader);

//Sofa
view = camera.GetViewMatrix();
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(65, 0, 13));
model = glm::translate(model, glm::vec3(posX, posY, posZ));
model = glm::rotate(model, glm::radians(2*rot), glm::vec3(0.0f, 1.0f, 0.0));
model = glm::scale(model, glm::vec3(8.6));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Sofa.Draw(lightingShader);

//Sofa 2
view = camera.GetViewMatrix();
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(106, 0, 13));
model = glm::translate(model, glm::vec3(posX, posY, posZ));
model = glm::rotate(model, glm::radians(2*rot), glm::vec3(0.0f, 1.0f, 0.0));
model = glm::scale(model, glm::vec3(8.6));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Sofa.Draw(lightingShader);

//Sofa 3
view = camera.GetViewMatrix();
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(0, 0, -49.2));
model = glm::translate(model, glm::vec3(posX, posY, posZ));
model = glm::rotate(model, glm::radians(1*rot), glm::vec3(0.0f, 1.0f, 0.0));
model = glm::scale(model, glm::vec3(8.5));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Sofa.Draw(lightingShader);

//Jukebox
view = camera.GetViewMatrix();
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(20, 0, -11));
model = glm::translate(model, glm::vec3(posX, posY, posZ));
model = glm::rotate(model, glm::radians(2 * rot), glm::vec3(0.0f, 1.0f, 0.0));
model = glm::scale(model, glm::vec3(.6));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Jukebox.Draw(lightingShader); //Disque
view = camera.GetViewMatrix();
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(20.9, 1, 18.6));
```

```
model = glm::translate(model, glm::vec3(posX, posY, posZ));

model = glm::rotate(model, glm::radians(rotJuke), glm::vec3(0.0f, 1.0, 0.0f));
model = glm::scale(model, glm::vec3(.58));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Disque.Draw(lightingShader);

//palco
view = camera.GetViewMatrix();
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(5, 0, 0));
model = glm::translate(model, glm::vec3(posX, posY, posZ));
model = glm::rotate(model, glm::radians(3 * rot), glm::vec3(0.0f, 1.0f, 0.0));
model = glm::scale(model, glm::vec3(2));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
palco.Draw(lightingShader);

//Barriles
view = camera.GetViewMatrix();
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(5, 0, 0));
model = glm::translate(model, glm::vec3(posX, posY, posZ));
model = glm::rotate(model, glm::radians(3 * rot), glm::vec3(0.0f, 1.0f, 0.0));
model = glm::scale(model, glm::vec3(2));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Barril.Draw(lightingShader);

//Luces
view = camera.GetViewMatrix();
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(5, 0, 0));
model = glm::translate(model, glm::vec3(posX, posY, posZ));
model = glm::rotate(model, glm::radians(3 * rot), glm::vec3(0.0f, 1.0f, 0.0));
model = glm::scale(model, glm::vec3(2));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Luces.Draw(lightingShader);

//Chain
view = camera.GetViewMatrix();
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(posX, posY, posZ));
model = glm::translate(model, glm::vec3(60, 0, -70));
model = glm::rotate(model, glm::radians(rotdisco), glm::vec3(0.0f, 1.0, 0.0f));
model = glm::scale(model, glm::vec3(2));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Chain.Draw(lightingShader);

//ball
view = camera.GetViewMatrix();
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(posX, posY, posZ));
model = glm::translate(model, glm::vec3(60, 0, -70));
model = glm::rotate(model, glm::radians(rotdisco), glm::vec3(0.0f, 1.0, 0.0f));
model = glm::scale(model, glm::vec3(2));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
ball.Draw(lightingShader);

//tuberias
view = camera.GetViewMatrix();
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(5, 0, 0));
model = glm::translate(model, glm::vec3(posX, posY, posZ));
model = glm::rotate(model, glm::radians(3 * rot), glm::vec3(0.0f, 1.0f, 0.0));
model = glm::scale(model, glm::vec3(2));
```

```
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
tuberias.Draw(lightingShader);

//wall
view = camera.GetViewMatrix();
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(5, 0, 0));
model = glm::translate(model, glm::vec3(posX, posY, posZ));
model = glm::rotate(model, glm::radians(3 * rot), glm::vec3(0.0f, 1.0f, 0.0));
model = glm::scale(model, glm::vec3(2));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
wall.Draw(lightingShader);

//wall1
view = camera.GetViewMatrix();
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(5, 0, 0));
model = glm::translate(model, glm::vec3(posX, posY, posZ));
model = glm::rotate(model, glm::radians(3 * rot), glm::vec3(0.0f, 1.0f, 0.0));
model = glm::scale(model, glm::vec3(2));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
wall1.Draw(lightingShader);

//wall2
view = camera.GetViewMatrix();
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(5, 0, 0));
model = glm::translate(model, glm::vec3(posX, posY, posZ));
model = glm::rotate(model, glm::radians(3 * rot), glm::vec3(0.0f, 1.0f, 0.0));
model = glm::scale(model, glm::vec3(2));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
wall2.Draw(lightingShader);

//wall3
view = camera.GetViewMatrix();
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(5, 0, 0));
model = glm::translate(model, glm::vec3(posX, posY, posZ));
model = glm::rotate(model, glm::radians(3 * rot), glm::vec3(0.0f, 1.0f, 0.0));
model = glm::scale(model, glm::vec3(2));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
wall3.Draw(lightingShader);

//wallcita
view = camera.GetViewMatrix();
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(5, 0, 0));
model = glm::translate(model, glm::vec3(posX, posY, posZ));
model = glm::rotate(model, glm::radians(3 * rot), glm::vec3(0.0f, 1.0f, 0.0));
model = glm::scale(model, glm::vec3(2));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
wallcita.Draw(lightingShader);

//altar
view = camera.GetViewMatrix();
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(5, 0, 0));
model = glm::translate(model, glm::vec3(posX, posY, posZ));
model = glm::rotate(model, glm::radians(3 * rot), glm::vec3(0.0f, 1.0f, 0.0));
model = glm::scale(model, glm::vec3(2));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
altar.Draw(lightingShader);

//beer
```

```
view = camera.GetViewMatrix();
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(5, 0, 0));
model = glm::translate(model, glm::vec3(posX, posY, posZ));
model = glm::rotate(model, glm::radians(3 * rot), glm::vec3(0.0f, 1.0f, 0.0));
model = glm::scale(model, glm::vec3(2));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
beer.Draw(lightingShader);

//ventana
view = camera.GetViewMatrix();
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(5, 0, 0));
model = glm::translate(model, glm::vec3(posX, posY, posZ));
model = glm::rotate(model, glm::radians(3 * rot), glm::vec3(0.0f, 1.0f, 0.0));
model = glm::scale(model, glm::vec3(2));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
ventana.Draw(lightingShader);

//piso
view = camera.GetViewMatrix();
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(5, 0, 0));
model = glm::translate(model, glm::vec3(posX, posY, posZ));
model = glm::rotate(model, glm::radians(3 * rot), glm::vec3(0.0f, 1.0f, 0.0));
model = glm::scale(model, glm::vec3(2));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
piso.Draw(lightingShader);


//Vapor
view = camera.GetViewMatrix();
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(5, 0, 0));
model = glm::translate(model, glm::vec3(posX, posY, posZ));
model = glm::rotate(model, glm::radians(3 * rot), glm::vec3(0.0f, 1.0f, 0.0));
model = glm::scale(model, glm::vec3(2));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
vapor.Draw(lightingShader);

//techo
view = camera.GetViewMatrix();
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(5, 0, 0));
model = glm::translate(model, glm::vec3(posX, posY, posZ));
model = glm::rotate(model, glm::radians(3 * rot), glm::vec3(0.0f, 1.0f, 0.0));
model = glm::scale(model, glm::vec3(2));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
techo.Draw(lightingShader);

/////////Sección Proyecto Final de Teoría
//pared
view = camera.GetViewMatrix();
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(5, 0, 0));
model = glm::translate(model, glm::vec3(posX, posY, posZ));
model = glm::rotate(model, glm::radians(3 * rot), glm::vec3(0.0f, 1.0f, 0.0));
model = glm::scale(model, glm::vec3(2));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
pared.Draw(lightingShader); //pared
view = camera.GetViewMatrix();
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(5, 0, 0));
model = glm::translate(model, glm::vec3(posX, posY, posZ));
```

```
model = glm::rotate(model, glm::radians(3 * rot), glm::vec3(0.0f, 1.0f, 0.0));
model = glm::scale(model, glm::vec3(2));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
pared.Draw(lightingShader);

//baul
view = camera.GetViewMatrix();
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(5, 0, 0));
model = glm::translate(model, glm::vec3(posX, posY, posZ));
model = glm::rotate(model, glm::radians(3 * rot), glm::vec3(0.0f, 1.0f, 0.0));
model = glm::scale(model, glm::vec3(2));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
baul.Draw(lightingShader);

//closet
view = camera.GetViewMatrix();
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(5, 0, 0));
model = glm::translate(model, glm::vec3(posX, posY, posZ));
model = glm::rotate(model, glm::radians(3 * rot), glm::vec3(0.0f, 1.0f, 0.0));
model = glm::scale(model, glm::vec3(2));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
closet.Draw(lightingShader);

//escalera
view = camera.GetViewMatrix();
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(5, 0, 0));
model = glm::translate(model, glm::vec3(posX, posY, posZ));
model = glm::rotate(model, glm::radians(3 * rot), glm::vec3(0.0f, 1.0f, 0.0));
model = glm::scale(model, glm::vec3(2));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
escalera.Draw(lightingShader);

//escritorio
view = camera.GetViewMatrix();
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(5, 0, 0));
model = glm::translate(model, glm::vec3(posX, posY, posZ));
model = glm::rotate(model, glm::radians(3 * rot), glm::vec3(0.0f, 1.0f, 0.0));
model = glm::scale(model, glm::vec3(2));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
escritorio.Draw(lightingShader);

//flor
view = camera.GetViewMatrix();
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(5, 0, 0));
model = glm::translate(model, glm::vec3(posX, posY, posZ));
model = glm::rotate(model, glm::radians(3 * rot), glm::vec3(0.0f, 1.0f, 0.0));
model = glm::scale(model, glm::vec3(2));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
flor.Draw(lightingShader);

//lampara
view = camera.GetViewMatrix();
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(5, 0, 0));
model = glm::translate(model, glm::vec3(posX, posY, posZ));
model = glm::rotate(model, glm::radians(3 * rot), glm::vec3(0.0f, 1.0f, 0.0));
model = glm::scale(model, glm::vec3(2));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
lampara.Draw(lightingShader);
```

```
//libro
view = camera.GetViewMatrix();
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(5, 0, 0));
model = glm::translate(model, glm::vec3(posX, posY, posZ));
model = glm::rotate(model, glm::radians(3 * rot), glm::vec3(0.0f, 1.0f, 0.0));
model = glm::scale(model, glm::vec3(2));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
libro.Draw(lightingShader);


//mesa
view = camera.GetViewMatrix();
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(5, 0, 0));
model = glm::translate(model, glm::vec3(posX, posY, posZ));
model = glm::rotate(model, glm::radians(3 * rot), glm::vec3(0.0f, 1.0f, 0.0));
model = glm::scale(model, glm::vec3(2));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
mesa.Draw(lightingShader);

//papel
view = camera.GetViewMatrix();
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(5, 0, 0));
model = glm::translate(model, glm::vec3(posX, posY, posZ));
model = glm::rotate(model, glm::radians(3 * rot), glm::vec3(0.0f, 1.0f, 0.0));
model = glm::scale(model, glm::vec3(2));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
papel.Draw(lightingShader);

//pared
view = camera.GetViewMatrix();
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(5, 0, 0));
model = glm::translate(model, glm::vec3(posX, posY, posZ));
model = glm::rotate(model, glm::radians(3 * rot), glm::vec3(0.0f, 1.0f, 0.0));
model = glm::scale(model, glm::vec3(2));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
pared.Draw(lightingShader);

//piso2
view = camera.GetViewMatrix();
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(5, 0, 0));
model = glm::translate(model, glm::vec3(posX, posY, posZ));
model = glm::rotate(model, glm::radians(3 * rot), glm::vec3(0.0f, 1.0f, 0.0));
model = glm::scale(model, glm::vec3(2));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
piso2.Draw(lightingShader);

//reloj
view = camera.GetViewMatrix();
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(5, 0, 0));
model = glm::translate(model, glm::vec3(posX, posY, posZ));
model = glm::rotate(model, glm::radians(3 * rot), glm::vec3(0.0f, 1.0f, 0.0));
model = glm::scale(model, glm::vec3(2));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
reloj.Draw(lightingShader);


//silla
```

```
		view = camera.GetViewMatrix();
		model = glm::mat4(1);
		model = glm::translate(model, glm::vec3(5, 0, 0));
		model = glm::translate(model, glm::vec3(posX, posY, posZ));
		model = glm::rotate(model, glm::radians(3 * rot), glm::vec3(0.0f, 1.0f, 0.0));
		model = glm::scale(model, glm::vec3(2));
		glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
		silla.Draw(lightingShader);


		//silla2
		view = camera.GetViewMatrix();
		model = glm::mat4(1);
		model = glm::translate(model, glm::vec3(5, 0, 0));
		model = glm::translate(model, glm::vec3(posX, posY, posZ));
		model = glm::rotate(model, glm::radians(3 * rot), glm::vec3(0.0f, 1.0f, 0.0));
		model = glm::scale(model, glm::vec3(2));
		glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
		silla2.Draw(lightingShader);


		//sofa
		view = camera.GetViewMatrix();
		model = glm::mat4(1);
		model = glm::translate(model, glm::vec3(5, 0, 0));
		model = glm::translate(model, glm::vec3(posX, posY, posZ));
		model = glm::rotate(model, glm::radians(3 * rot), glm::vec3(0.0f, 1.0f, 0.0));
		model = glm::scale(model, glm::vec3(2));
		glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
		sofa.Draw(lightingShader);


		//techo2
		view = camera.GetViewMatrix();
		model = glm::mat4(1);
		model = glm::translate(model, glm::vec3(5, 0, 0));
		model = glm::translate(model, glm::vec3(posX, posY, posZ));
		model = glm::rotate(model, glm::radians(3 * rot), glm::vec3(0.0f, 1.0f, 0.0));
		model = glm::scale(model, glm::vec3(2));
		glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
		techo2.Draw(lightingShader);

		// Also draw the lamp object, again binding the appropriate shader
		lampShader.Use();
		// Get location objects for the matrices on the lamp shader (these could be different
on a different shader)
		modelLoc = glGetUniformLocation(lampShader.Program, "model");
		viewLoc = glGetUniformLocation(lampShader.Program, "view");
		projLoc = glGetUniformLocation(lampShader.Program, "projection");

		// Set matrices
		glUniformMatrix4fv(viewLoc, 1, GL_FALSE, glm::value_ptr(view));
		glUniformMatrix4fv(projLoc, 1, GL_FALSE, glm::value_ptr(projection));
		model = glm::mat4(1);
		model = glm::translate(model, lightPos);
		//model = glm::scale(model, glm::vec3(0.2f)); // Make it a smaller cube
		glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
		// Draw the light object (using light's vertex attributes)
		glBindVertexArray(lightVAO);
		for (GLuint i = 0; i < 4; i++)
		{
			model = glm::mat4(1);
			model = glm::translate(model, pointLightPositions[i]);
			model = glm::scale(model, glm::vec3(0.2f)); // Make it a smaller cube
```

```
                glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
                glDrawArrays(GL_TRIANGLES, 0, 36);
            }
            glBindVertexArray(0);
            // Swap the screen buffers
            glfwSwapBuffers(window);
        }



        glfwTerminate();




        return 0;
}


void animacion()
{

            //Movimiento del changuito - Monkey's movement
            // Cuando play sea verdadero nos dejará entrar a su proceso donde a través de la
interpolacion se puede animar el movimiento del objeto.
            //Play will be true when we press L in our keyboard, and it'll use interpolation to
make the movement from one site to another.
            if (play)
            {
                if (i_curr_steps >= i_max_steps) //end of animation between frames?
                {
                    playIndex++;
                    if (playIndex>FrameIndex - 2)   //end of total animation?
                    {
                        printf("termina anim\n");
                        playIndex = 0;
                        play = false;
                    }
                    else //Next frame interpolations
                    {
                        i_curr_steps = 0; //Reset counter
                                                //Interpolation
                        interpolation();
                    }
                }
                else
                {
                    //Dibujamos la animación - Draw animation
                    posX += KeyFrame[playIndex].incX;
                    posY += KeyFrame[playIndex].incY;
                    posZ += KeyFrame[playIndex].incZ;

                    rotRodIzq += KeyFrame[playIndex].rotInc;
                    rotManIzq += KeyFrame[playIndex].rotInc3;
                    rotManDer += KeyFrame[playIndex].rotInc4;


                    i_curr_steps++;
                }

            }

            /* Si circuito está activo entonces se producirá el movimiento de la Diana que se
actualiza cuando rebase cierto rango para avanzar en otro eje.
```

```
               If it's active then our Bullseye will move in a square shape*/
            if (circuito)
            {
                  //Ejercicio 1 Movimiento sobre el rectangulo
                  if (recorrido1)
                  {
                        movKitY += 0.25f; //Mueve la diana en el eje Y  -  Moves the bullseye in
Y axis
                        if (movKitY > 11.5)
                        {
                              recorrido1 = false; //Desactivamos el estado para poder entrar al
siguiente de inmediato
                              recorrido2 = true; // We deactivate the states to change the
direction in which it continues moving.
                        }
                  }
                  if (recorrido2)
                  {
                        movKitZ -= 0.25f; //Mueve la diana en el eje Z  -  Moves the bullseye in
Z axis
                        if (movKitZ < -11)
                        {
                              recorrido2 = false;
                              recorrido3 = true;

                        }
                  }

                  if (recorrido3)
                  {
                        movKitY -= 0.25f; //Mueve la diana en el eje Y  -  Moves the bullseye in
Y axis
                        if (movKitY < 3.2)
                        {
                              recorrido3 = false;
                              recorrido4 = true;
                        }
                  }

                  if (recorrido4)
                  {
                        movKitZ += 0.25f; //Mueve la diana en el eje Z  -  Moves the bullseye in
Z axis
                        if (movKitZ > 11)
                        {
                              recorrido4 = false;
                              recorrido1 = true;
                        }
                  }
            }

            //Si la puerta está activada entonces oscilará su valor de rotación para que simule
que está abriendose
            // If the door is activated the movement between one and othe state is declared
            if (puerta)
            {
                  if (55 > rotpuerta) {
                        rotpuerta += 0.25f;
                  }
                  if (55 <= rotpuerta) {
                        aux = true;
                        puerta = false;
                  }
            }
```

```
		if (aux) {
				rotpuerta -= 0.25f;
				if (0 > rotpuerta) {
					puerta = true;
					aux = false;
				}
			}


		// Para la esfera de disco se aumentará en razon de .3 para que gira lentamente.
		// If activated it while spin by itself
		if (disco) {
			rotdisco += 0.3f;
		}



		// Para el disco dentro de la rocola se aumentará en razon de .3 para que gira
lentamente,´nuevamente.
		// If activated it while spin by itself

		if (Juke) {
			rotJuke += 0.3f;
		}
	}


// Hace todo lo que le indique mientras se esten doblan sos piernas. -  Is called whenever a key is
pressed/released via GLFW
void KeyCallback(GLFWwindow *window, int key, int scancode, int action, int mode)
{
	//Luego de presionar la L se hace la interpolación para mover la figura en un eje.
	//After activating L the interpolation is done to make it rotate in the same axis.

	if (keys[GLFW_KEY_L])
	{
		if (play == false && (FrameIndex > 1))
		{

			resetElements();
			//First Interpolation
			interpolation();

			play = true;
			playIndex = 0;
			i_curr_steps = 0;
		}
		else
		{
			play = false;
		}

	}


	//Si presionamos la tecla cambiará el valor de su boleano por defecto. En este caso pasará
de false a true
	//When pushing the Key it will activate the movement detailed on the animation function
	if (keys[GLFW_KEY_K])
	{
		puerta = not puerta;

	}
```

```
	if (keys[GLFW_KEY_P])
	{
		disco = not disco;

	}

	if (keys[GLFW_KEY_U])
	{
		Juke = not Juke;

	}

	//Debido a esta función cada vez que presionamos la tecla solo cuenta como si se apretara una vez, gracias a esto si mantenemos
	//la tecla oprimida aún así solo se activará una vez el comando
	//This instructions let us push a key without updating more than once the value of the variable while still pressing the key.
	if (GLFW_KEY_ESCAPE == key && GLFW_PRESS == action)
	{
		glfwSetWindowShouldClose(window, GL_TRUE);
	}

	if (key >= 0 && key < 1024)
	{
		if (action == GLFW_PRESS)
		{
			keys[key] = true;
		}
		else if (action == GLFW_RELEASE)
		{
			keys[key] = false;
		}
	}
	// Prende la luz 1
	//Turns on the light 1
	if (keys[GLFW_KEY_SPACE])
	{
		active = !active;
		if (active)
			LightP1 = glm::vec3(1.0f, 0.0f, 0.0f);
		else
			LightP1 = glm::vec3(0.0f, 0.0f, 0.0f);
	}
}

void MouseCallback(GLFWwindow *window, double xPos, double yPos)
{

	if (firstMouse)
	{
		lastX = xPos;
		lastY = yPos;
		firstMouse = false;
	}

	GLfloat xOffset = xPos - lastX;
	GLfloat yOffset = lastY - yPos;  // Reversed since y-coordinates go from bottom to left

	lastX = xPos;
	lastY = yPos;

	camera.ProcessMouseMovement(xOffset, yOffset);
}
```

```
// Moves/changes the camera positions based on user input
void DoMovement()
{

	if (keys[GLFW_KEY_1])
	{

		movCamera = 2.5f;//Manda una velocidad de 0.01 a la camara automatica

	}

	// Inicia el movimiento para la diana
	// Activates the animation for the bullseye
	if (keys[GLFW_KEY_I])
	{
		circuito = true;
	}

	//Detiene la diana en cualquier punto
	// Deactivates the animation for the bullseye
	if (keys[GLFW_KEY_O])
	{
		circuito = false;
	}

	// Controles de la camara
	// Camera controls
	if (keys[GLFW_KEY_W] || keys[GLFW_KEY_UP]) //Permite que la camara se mueva en vertical - 
Let's the camera move up
	{
		camera.ProcessKeyboard(FORWARD, deltaTime);

	}

	if (keys[GLFW_KEY_S] || keys[GLFW_KEY_DOWN]) //Permite que la camara se mueva en vertical, 
hacia abajo - Let's the camera move down
	{
		camera.ProcessKeyboard(BACKWARD, deltaTime);


	}

	if (keys[GLFW_KEY_A] || keys[GLFW_KEY_LEFT]) //Permite que la camara se mueva hacia la 
izquierda - Let's the camera move left
	{
		camera.ProcessKeyboard(LEFT, deltaTime);


	}

	if (keys[GLFW_KEY_D] || keys[GLFW_KEY_RIGHT]) //Permite que la camara se mueva a la derecha 
- Let's the camera move to the right
	{
		camera.ProcessKeyboard(RIGHT, deltaTime);
	}


}
```

Conclusiones y comentarios finales:

Se crearon los elementos para poblar las habitaciones creadas. Se buscó complementar el proyecto de tal manera que cumpliera con la cantidad de elementos solicitados. De esta forma y siguiendo la temática de la serie, luego de unas escaleras podemos encontrar el estudio que se creó.

En general, se aplicaron todas las técnicas de modelado y animación vistas en el curso con el fin de demostrar el conocimiento aprendido o reforzarlo en todo caso. Me quedo satisfecho con el esfuerzo que le metí al proyecto porque fueron demasiadas horas y pese a que me encontré bastantes adversidades a la hora de desarrollarlo se puede decir que llegué a la meta, cumplí con la entrega, además de que pese a que no es exacto a las imágenes de referencia me aproximé a ello. La habilidades que he adquirido en Maya, GIMP, GitHub y conocimientos para programar en Visual me asombran porque logré dominarlo en un solo semestre.

Me habría gustado conocer desde el principio sobre animación compleja para poder definir desde ese momento que objetos animar porque si se reducen las posibilidades una vez que ya fijaste una temática y en las imágenes de referencia no existen más objetos con movilidad.

En conclusión, logré animar los objetos en cuestión y se modeló el espacio de forma exitosa.