Cabrera Beltrán Héctor Eduardo
Grupo: 04

Proyecto Final

Objetivo

Aplicar los conocimientos adquiridos en la asignatura de Computación Gráfica para recrear un espacio virtual basado en una vivienda real o ficticia, además de dos habitaciones dentro de dicha vivienda.

Alcance

Diseñar y representar un espacio virtual con base en la casa de *Los Simpson* utilizando técnicas de modelado 3D con ayuda del software *Autodesk Maya*.

Recrear dentro del ambiente virtual dos habitaciones ya existentes en la vivienda; la sala, que se encuentra junto a la entrada de la casa, y el cuarto con la televisión serán las estancias por modelar.

Representar doce muebles dentro de ambos cuartos para contribuir a la fidelidad del realismo de la vivienda, por lo que esta debe contar con un sillón, un sofá, una chimenea, un piano, una mesa de centro y un par de lámparas, burós, una televisión, la caja del cable, un librero, dos bocinas y una lámpara de piso.

Obtener todos los muebles, a excepción de la mesa, usando el software mencionado anteriormente. Por otro lado, diseñar la mesa por medio de código dentro del entorno de desarrollo de *MS Visual Studio* utilizando *OpenGL.*

Texturizar todos los objetos que se usarán en el espacio virtual.

Añadir distintas animaciones a los muebles; desplazar el sillón, abrir y cerrar los cajones, consumir y suministrar la madera de la chimenea, abrir y cerrar la puerta frontal, cambiar la posición de la antena que se encuentra sobre la caja del cable, al igual que trasladar los libros que se encuentran en el librero.

Introducir todos los elementos anteriores en un proyecto de *MS Visual Studio*. Asimismo, implementar una cámara sintética para moverse por el espacio virtual recreado.

Complementar el ambiente virtual con un *skybox texture*, simulando un entorno rodeado por un cielo en pleno día.

Generar un archivo ejecutable junto con sus complementos, a partir del proyecto realizado, que sea capaz de correrse en una computadora distinta de la que lo generó.

Requerimientos

- Elaboración de un manual de usuario.
- El espacio virtual recreado deberá tener una buena fidelidad respecto a la casa usada como referencia.

Cabrera Beltrán Héctor Eduardo
Grupo: 04

- El proyecto deberá ser elaborado en el entorno de desarrollo de *MS Visual Studio.*
- Modelado de, al menos, 5 elementos existentes por habitación dentro de la vivienda. Todos ellos deben estar texturizados.
- Existencia de 4 animaciones dentro del ambiente virtual.
- Manejo de cámara sintética para recorrer el espacio recreado.
- Creación y correcto funcionamiento de un archivo ejecutable.

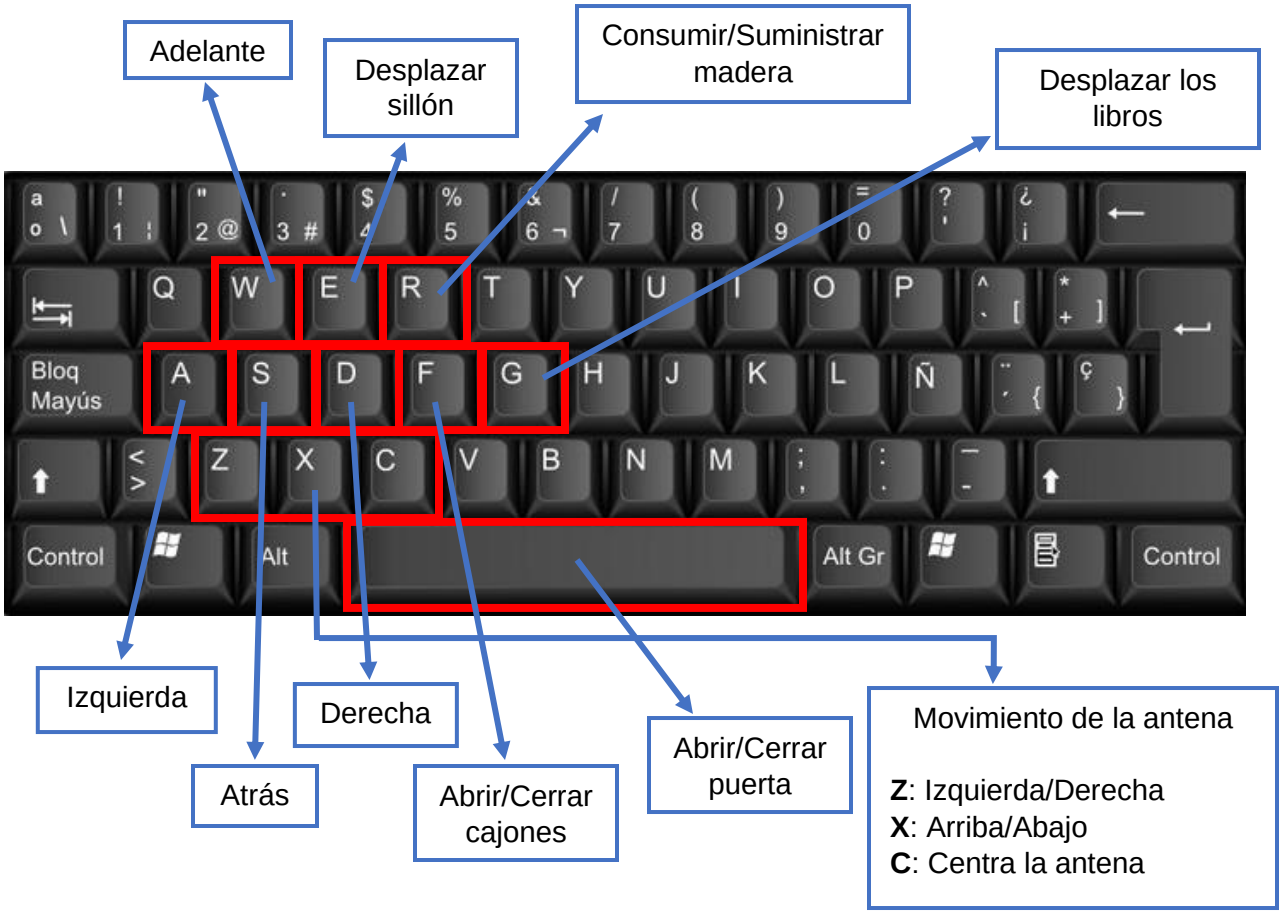
Cronograma de actividades (Diagrama de Gantt)

Actividades	Semana 1 (13/04 – 19/04)				Semana 2 (20/04 – 26/04)				Semana 3 (27/04 – 03/05)				Semana 4 (11/05 – 17/05)				Semana 5 (18/05 – 21/05)	
	L/M	M/J	V/S	D	L/M	M/J	V/S	D	L/M	M/J	V/S	D	L/M	M/J	V/S	D	L/M	M/J
Investigación sobre el uso de *Autodesk Maya*																		
Diseño y modelado de la fachada (exterior)																		
Texturizado y corrección de imperfecciones																		
Diseño y modelado de la fachada (interior)																		
Creación y modificación de los muebles																		
Texturizado y escalado de los muebles																		
Integración de los modelos en *MS Visual Studio*																		
Generación del código de la mesa																		
Elaboración de las animaciones																		
Implementación de un *skybox texture*																		
Investigación sobre creación del archivo ejecutable																		
Diseño y modelado de la segunda habitación																		
Creación, modificación y texturizado de los muebles restantes																		
Implementación de los nuevos muebles en *MS Visual Studio*																		
Elaboración las animaciones restantes																		
Creación de la documentación																		
Diseño del análisis de costos																		

Cabrera Beltrán Héctor Eduardo
Grupo: 04

Manual de usuario

Asignación de teclas



Movimiento

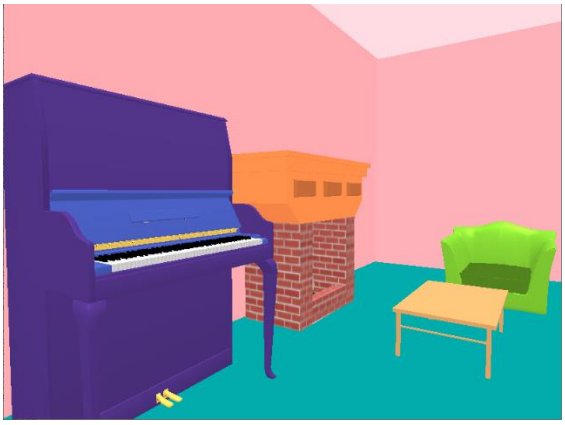
Para desplazarse por el escenario se utilizan las teclas **W**, **A**, **S**, **D**.

W	Adelante
A	Izquierda
S	Atrás
D	Derecha

Cabrera Beltrán Héctor Eduardo
Grupo: 04

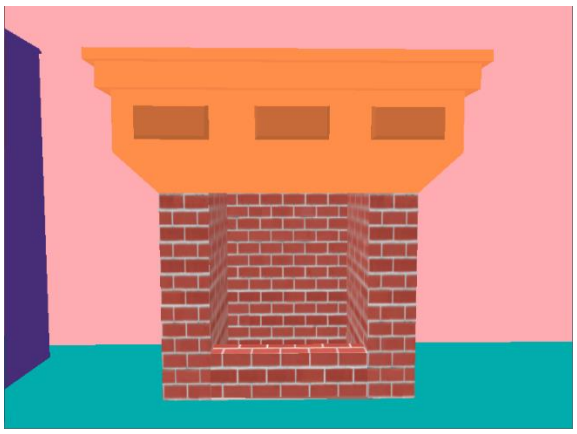
Movimiento de la cámara

Usar el mouse para mirar a los alrededores del escenario.



Consumir/Suministrar madera

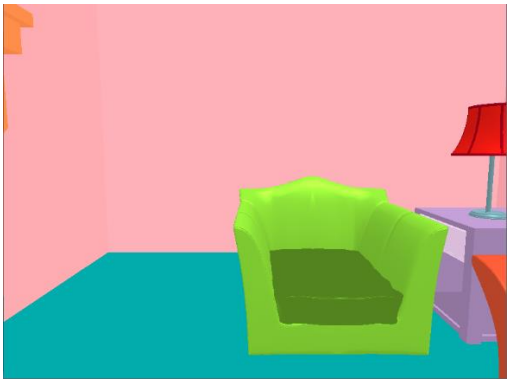
Cuando la madera se encuentre colocada en la chimenea, al presionar la tecla **R** se consumirá la madera.



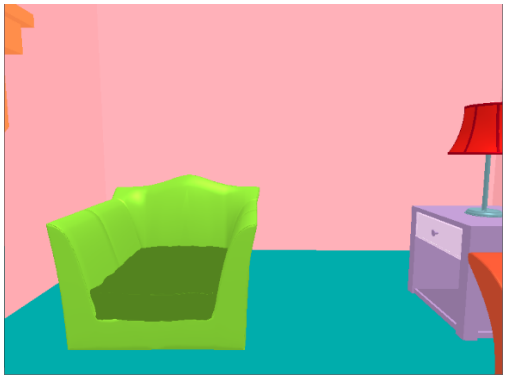
Cuando la madera esté consumida, al presionar la tecla **R** se abastecerá la madera.

Desplazar sillón

Utilizar la tecla **E** para trasladar al sillón verde a otra posición. Si el sillón ya está desplazado, al presionar nuevamente la tecla **E** volverá a su posición original.

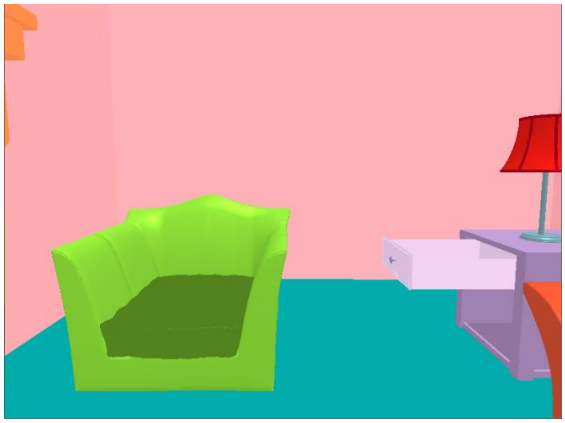


Posición original



Sillón desplazado

Nota: Si los cajones están abiertos, no se podrá regresar el sillón a su posición original.



Cabrera Beltrán Héctor Eduardo
Grupo: 04

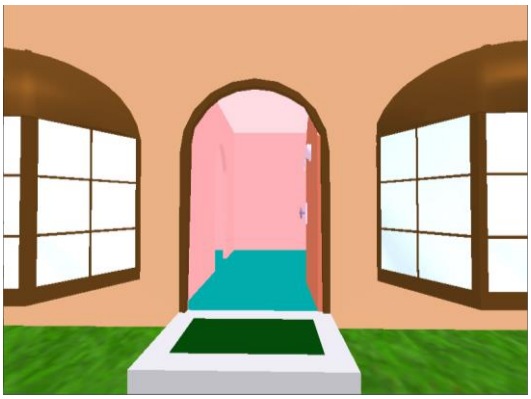
Abrir/Cerrar cajones

Al presionar la tecla **F**, los cajones se abrirán o cerrarán, dependiendo de la posición que tuvieran anteriormente.


Nota: Si el sillón verde está en su posición original, los cajones no podrán abrirse.

Abrir/Cerrar puerta

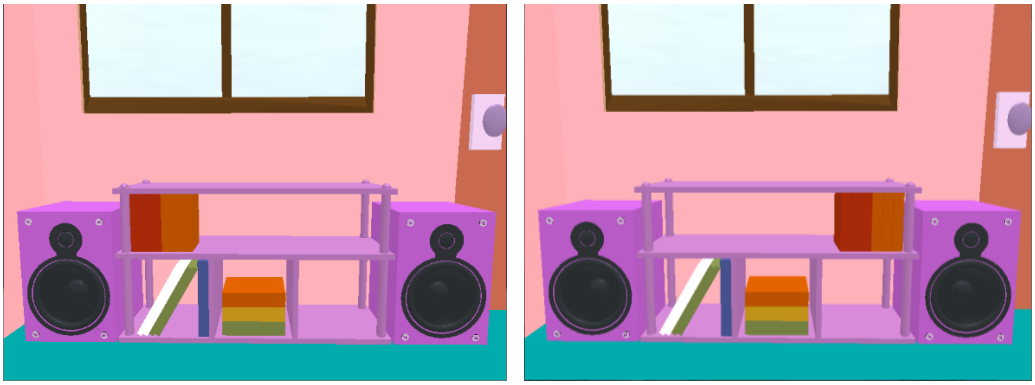
Si la puerta está cerrada, al presionar la tecla **espacio** se abrirá. En cambio, si la puerta está abierta, al oprimir la tecla **espacio** se cerrará.

Cabrera Beltrán Héctor Eduardo
Grupo: 04

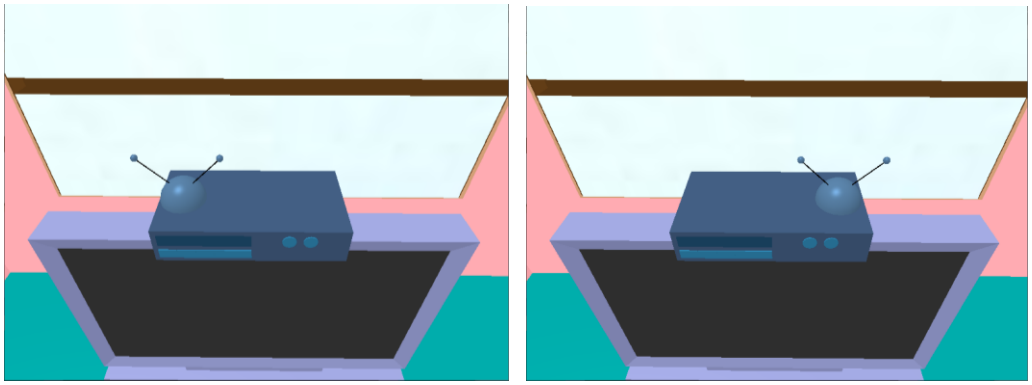
Desplazar los libros del librero

Al oprimir la tecla **G**, los libros en el segundo estante del librero se moverán hacia la izquierda o hacia la derecha, dependiendo del lugar donde se encontraban anteriormente.

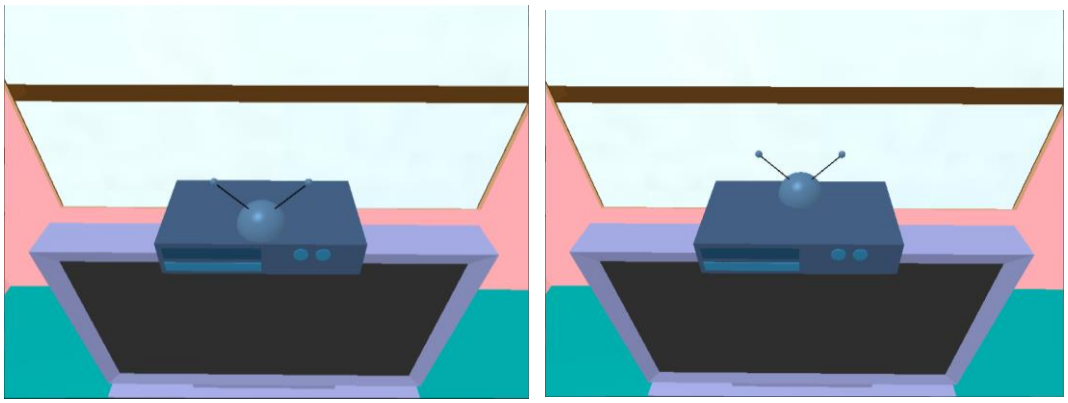


Cambiar la posición de la antena sobre la caja del cable

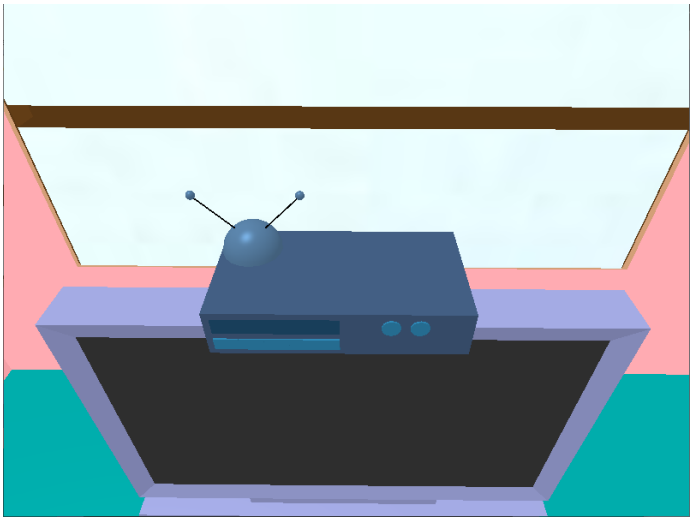
Al presionar la tecla **Z**, se puede desplazar la antena de izquierda a derecha, o viceversa.



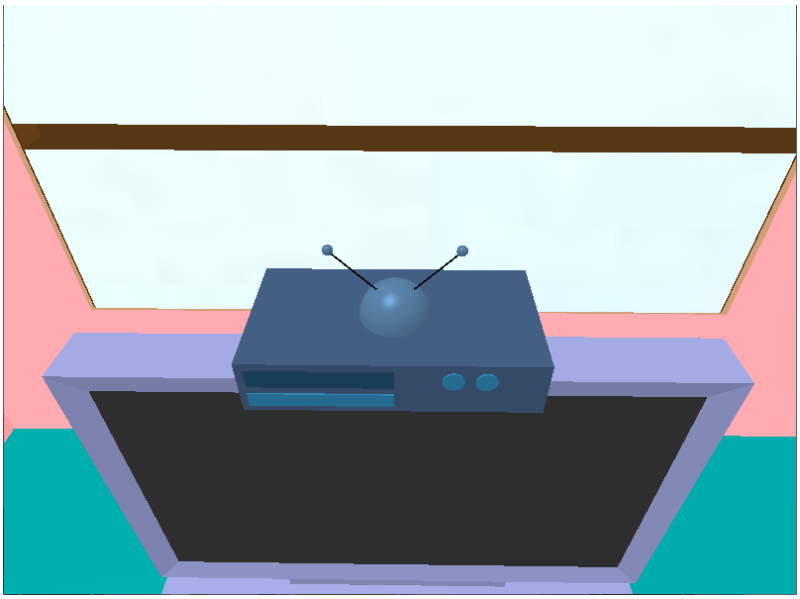
Con la tecla **X**, la antena se moverá hacia abajo o hacia arriba, dependiendo de su posición anterior.



Incluso, se pueden combinar ambos movimientos; se puede desplazar arriba/abajo e izquierda/derecha, por lo que la antena se puede posicionar en cualquiera de las cuatro esquinas de la caja del cable.



Para regresar la antena a su posición inicial, es decir, para centrarla, se presiona la tecla **C**.



Cabrera Beltrán Héctor Eduardo
Grupo: 04

Funciones principales del código fuente

- int main()

Configura todas las opciones necesarias para GLFW.

Crea un objeto ventana donde se visualizará el espacio virtual.

Define las dimensiones del viewport.

Establece las funciones callback requeridas.

Deshabilita el cursor dentro de la ventana.

Declara los objetos de tipo Shader para la iluminación y SkyBox.

Declara los modelos a partir de los archivos OBJ.

Define los vértices para formar un cubo con sus respectivas coordenadas de textura.

Define los vértices para crear el SkyBox.

Define los índices para indicar el orden para formar un cubo.

Establece el VAO y VBO.

Establece la posición en la que se encuentran los vértices, las normales y las coordenadas de textura.

Configura y carga las texturas necesarias para el SkyBox.

Carga la textura que se ocupará en la mesa y establece las unidades de textura.

Crea un ciclo, el cual permanecerá activo durante el tiempo en que esté abierta la ventana.

Dentro del ciclo:

Calcula el tiempo entre el frame actual y el frame anterior.

Verifica si algún evento ha sido activado, como presionar alguna tecla, mover el mouse o realizar alguna animación, y se manda a llamar a la función correspondiente.

Utiliza el Shader de iluminación para establecer los parámetros con los que se dibujarán los objetos.

Crea una Directional light.

Crea una Point light.

Define las propiedades de los materiales.

Modela la mesa, dibujando la base, después cada una de las patas y, por último, los soportes que van en cada par de patas. Cada una de las piezas se traslada, rota y/o escala para acomodarla en el lugar que corresponde dentro del ambiente virtual.

Carga cada uno de los modelos, definidos previamente, y les aplica sus respectivas transformaciones básicas para ubicarlos dentro del espacio virtual.

Para concluir lo que abarca el ciclo, dibuja el SkyBox.

Limpia todos los recursos asignados por GLFW.

Valor de retorno es 0.

- void KeyCallback(GLFWwindow *window, int key, int scancode, int action, int mode)

Se manda a llamar a esta función cada vez que se presiona/libera una tecla. Indica la tecla que fue oprimida, la cual se verificará si corresponde con alguna acción en la función DoMovement().

Valida si se presiona la tecla *Esc*. De ser así, cierra la ventana.

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

Obtiene el desplazamiento del mouse y actualiza la posición de la cámara con base en el movimiento del mouse.

- void DoMovement()

Altera (mueve) la posición de la cámara, basado en la tecla que oprima el usuario.

Cuando detecta que se ha presionado la tecla *W/arriba, S/abajo, A/izquierda* o *D/derecha*, realiza el movimiento correspondiente con la cámara para recorrer el espacio.

Se modifican las siguientes variables de acuerdo con la tecla que haya sido oprimida:

Tecla	Variable	Nuevo valor	Descripción
R	activo	true	Activa la animación del dispensador de madera.
F	activoC	true	Activa la animación de los cajones.
E	activoS	true	Activa la animación del sillón.
Espacio	activoP	true	Activa la animación de la puerta.
G	libAct	true	Activa la animación de los libros.

X	antActX	true	Activa el desplazamiento de la antena en el eje X.
Z	antActZ	true	Activa el desplazamiento de la antena en el eje Z.
C	antActC	true	Activa la animación de centrado de la antena.

- void animacionMadera()

Dependiendo de la posición anterior de la madera, esta se consumirá o será suministrada. Para realizar cualquiera de las acciones, primero debe ser activada la animación.

Variables:

seleccion = true, la madera fue consumida.

seleccion = false, la madera fue suministrada.

movCaida, cuando alcance el valor máximo para considerarse suministrada o consumida, se cambiará el valor de seleccion y la animación será desactivada. Si aún no se alcanzan dichos valores, movCaida disminuirá 0.05.

Variable	Valor anterior	Nuevo valor
activo	true	false
seleccion	true	false
	false	true

- void animacionPuerta()

Cuando la animación esté activa, la puerta se abrirá o cerrará.

Variables:

abierta = true, la puerta está abierta.

abierta = false, la puerta está cerrada.

rotP, indica los grados con los que irá rotando la puerta. Estos aumentarán o disminuirán 1.0 y, cuando la puerta esté completamente cerrada o abierta, se desactivará la animación y se indicará su estado.

Variable	Valor anterior	Nuevo valor
activoP	true	false
abierta	true	false
	false	true

- void animacionSillon()

Si la animación está activa, el sillón será desplazado de su posición original o regresará a esta si ya se había movido. Sin embargo, para regresar a su posición original, la variable dentro debe ser true.

Variables:

desplazado = true, sillón no está en su posición original.

desplazado = false, sillón está en su posición original.

desplazamiento, aumentará o disminuirá 0.05 dependiendo si el sillón está desplazado o no. Cuando el sillón llegue a su posición final, se desactivará la animación y se indicará si está en su posición original o no.

Variable	Valor anterior	Nuevo valor
activoS	true	false
desplazado	true	false
	false	true

- void animacionCajon()

Mientras esté activa la animación, los cajones se meterán, si estaban fuera, o saldrán, si se estaban dentro. No obstante, para que puedan salir, la variable desplazado debe ser true.

Variables:

dentro = true, los cajones están dentro del buró.

dentro = false, los cajones están fuera del buró.

movi, aumentará o disminuirá 0.05 dependiendo de la posición en la que estén los cajones. Cuando estos estén completamente dentro o fuera, se desactivará la animación y se indicará su posición final.

Variable	Valor anterior	Nuevo valor
activoC	true	false
dentro	true	false
	false	true

- void animacionAntena()

Cuando la animación esté activa, verificará en qué eje se desplazará o si se centrará la antena. Si ya se había movido en un respectivo eje y se vuelve a invocar la animación en el mismo eje, se desplazará hacia el lado contrario sobre el mismo eje.

Variables:

antDx = true, indica que la antena se movió hacia abajo.

antDx = false, indica que la antena se movió hacia arriba.

antDz = true, indica que la antena se movió hacia la izquierda.

antDz = false, indica que la antena se movió hacia la derecha.

antMov, es un vector de 3 dimensiones, donde el componente x y el componente z aumentarán o disminuirán 0.01 dependiendo del movimiento anterior realizado sobre su respectivo eje. Cuando se alcance el desplazamiento máximo en el eje que activó la animación, se desactivará esta y se indicará si la antena está arriba/abajo e izquierda/derecha. Cuando la animación activa sea para centrar la antena, se verificará la posición de la antena en el eje x y en el eje z, y posteriormente se moverá hacia el centro respecto a su ubicación actual; la animación se desactivará cuando se haya movido lo suficiente, tanto en x como en z, para estar en el centro de la caja del cable.

Variable	Valor anterior	Nuevo valor
antActX	true	false
antActZ		
antDx	true	false
	false	true
antDz	true	false
	false	true

Variable	Valor anterior	Nuevo valor
antActC	True	false
antDx	true	false
	false	false
antDz	true	false
	false	false

- void animacionLibros()

Con la animación activa, se moverán los libros de izquierda a derecha, o viceversa.

Variables:

libDes = true, indica que los libros se desplazaron a la derecha.

libDes = false, indica que los libros se desplazaron a la izquierda.

libMov, cuando los libros estén en un extremo u otro, se desactivará la animación y se indicará si se encuentran a la izquierda o a la derecha. Si aún no están en el extremo, libMov aumentará o disminuirá 0.05.

Variable	Valor anterior	Nuevo valor
libAct	true	false
libDes	true	false
	false	true

Cabrera Beltrán Héctor Eduardo
Grupo: 04

Código fuente

```
#include <iostream>
#include <cmath>

// GLEW
#include <GL/glew.h>

// GLFW
#include <GLFW/glfw3.h>

// Other Libs
#include "stb_image.h"

// GLM Mathematics
#include <glm/glm.hpp>
#include <glm/gtc/matrix_transform.hpp>
#include <glm/gtc/type_ptr.hpp>

//Load Models
#include "SOIL2/SOIL2.h"


// Other includes
#include "Shader.h"
#include "Camera.h"
#include "Model.h"
#include "Texture.h"

// Function prototypes
void KeyCallback(GLFWwindow *window, int key, int scancode, int action, int mode);
void MouseCallback(GLFWwindow *window, double xPos, double yPos);
void DoMovement();
void animacionMadera();
void animacionPuerta();
void animacionSillon();
void animacionCajon();
void animacionAntena();
void animacionLibros();

// Window dimensions
const GLuint WIDTH = 800, HEIGHT = 600;
int SCREEN_WIDTH, SCREEN_HEIGHT;


// Camera
Camera  camera(glm::vec3(-100.0f, 2.0f, -45.0f));
GLfloat lastX = WIDTH / 2.0;
GLfloat lastY = HEIGHT / 2.0;
bool keys[1024];
bool firstMouse = true;

//Animación del dispensador de madera
float movCaida = -2.5f; //Movimiento de caída de dispensador o consumo de la madera
bool activo = false; //Indica si se activa la animación
bool seleccion = false; //Indica qué movimiento se realizará

//Animación de la puerta
float rotP = 180.0f; //Rotación que realiza
bool activoP = false; //Indica si se activa la animación
bool abierta = false; //Indica si la puerta está abierta o cerrada

//Animación del sillón verde
float desplazamiento = -100.0f; //Desplazamiento
bool activoS = false; //Indica si se activa la animación
bool desplazado = false; //Indica si ya se movió

//Animación de los cajones
float movi = -106.7f; //Desplazamiento
bool activoC = false; //Indica si se activa la animación
bool dentro = true; //Indica si el cajón está adentro o afuera
```

```
//Animación de la antena
glm::vec3 antMov(-105.3f, -2.09f, -85.2f); //Desplazamiento
bool antActX = false; //Indica si se activa la animación para moverla en X
bool antActZ = false; //Indica si se activa la animación para moverla en Z
bool antActC = false; //Indica si se activa la animación para centrarla
bool antDX = false; //Indica si ya se movió en X
bool antDZ = false; //Indica si ya se movió en Z

//Animación de los libros
float libMov = -106.77f; //Desplazamiento
bool libAct = false; //Indica si se activa la animación
bool libDes = false; //Indica si ya se movieron los libros


// Deltatime
GLfloat deltaTime = 0.0f;   // Time between current frame and last frame
GLfloat lastFrame = 0.0f;   // Time of last frame

int main()
{
    // Init GLFW
    glfwInit();
    // Set all the required options for GLFW
    glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3);
    glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
    glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);
    glfwWindowHint(GLFW_OPENGL_FORWARD_COMPAT, GL_TRUE);
    glfwWindowHint(GLFW_RESIZABLE, GL_FALSE);

    // Create a GLFWwindow object that we can use for GLFW's functions
    GLFWwindow* window = glfwCreateWindow(WIDTH, HEIGHT, "Proyecto Final", nullptr, nullptr);

    if (nullptr == window)
    {
        std::cout << "Failed to create GLFW window" << std::endl;
        glfwTerminate();

        return EXIT_FAILURE;
    }

    glfwMakeContextCurrent(window);

    glfwGetFramebufferSize(window, &SCREEN_WIDTH, &SCREEN_HEIGHT);

    // Set the required callback functions
    glfwSetKeyCallback(window, KeyCallback);
    glfwSetCursorPosCallback(window, MouseCallback);

    // GLFW Options
    glfwSetInputMode(window, GLFW_CURSOR, GLFW_CURSOR_DISABLED);


   // Set this to true so GLEW knows to use a modern approach to retrieving function pointers and extensions
   glewExperimental = GL_TRUE;
   // Initialize GLEW to setup the OpenGL Function pointers
   if (GLEW_OK != glewInit())
   {
       std::cout << "Failed to initialize GLEW" << std::endl;
       return EXIT_FAILURE;
   }

   // Define the viewport dimensions
   glViewport(0, 0, SCREEN_WIDTH, SCREEN_HEIGHT);

   // OpenGL options
   glEnable(GL_DEPTH_TEST);
   glEnable(GL_BLEND);
   glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);

   Shader lightingShader("Shaders/lighting.vs", "Shaders/lighting4.frag");
   Shader SkyBoxShader("Shaders/Skybox.vs", "Shaders/Skybox.frag");
```

```
Model Casa((char*)"Models/Casa/casa.obj");
Model Puerta((char*)"Models/Puerta/puerta.obj");
Model Buro((char*)"Models/Buro/buro.obj");
Model Cajon((char*)"Models/Cajon/cajon.obj");
Model Chimenea((char*)"Models/Chimenea/chimenea.obj");
Model Madera((char*)"Models/Madera/madera.obj");
Model Piano((char*)"Models/Piano/piano.obj");
Model SillonVerde((char*)"Models/SillonVerde/sillonVerde.obj");
Model SillonCafe((char*)"Models/SillonCafe/sillonCafe.obj");
Model Lampara((char*)"Models/Lampara/lampara.obj");
Model LamparaP((char*)"Models/LamparaPiso/lamparaP.obj");
Model TV((char*)"Models/TV/TV.obj");
Model CajaCable((char*)"Models/Caja del cable/cajaCable.obj");
Model Antena((char*)"Models/Antena/antena.obj");
Model Bocina((char*)"Models/Bocina/bocina.obj");
Model Librero((char*)"Models/Librero/librero.obj");
Model Libros((char*)"Models/Libros/libros.obj");

// Build and compile our shader program
```

```
// Set up vertex data (and buffer(s)) and attribute pointers
GLfloat vertices[] =
{
    // Positions           // Normals              // Texture Coords
    -0.5f, -0.5f, -0.5f,    0.0f,  0.0f, -1.0f,     0.0f,  0.0f,
    0.5f, -0.5f, -0.5f,     0.0f,  0.0f, -1.0f,     1.0f,  0.0f,
    0.5f,  0.5f, -0.5f,     0.0f,  0.0f, -1.0f,     1.0f,  1.0f,
    0.5f,  0.5f, -0.5f,     0.0f,  0.0f, -1.0f,     1.0f,  1.0f,
    -0.5f,  0.5f, -0.5f,    0.0f,  0.0f, -1.0f,     0.0f,  1.0f,
    -0.5f, -0.5f, -0.5f,    0.0f,  0.0f, -1.0f,     0.0f,  0.0f,

    -0.5f, -0.5f,  0.5f,    0.0f,  0.0f,  1.0f,     0.0f,  0.0f,
    0.5f, -0.5f,  0.5f,     0.0f,  0.0f,  1.0f,     1.0f,  0.0f,
    0.5f,  0.5f,  0.5f,     0.0f,  0.0f,  1.0f,     1.0f,  1.0f,
    0.5f,  0.5f,  0.5f,     0.0f,  0.0f,  1.0f,     1.0f,  1.0f,
    -0.5f,  0.5f,  0.5f,    0.0f,  0.0f,  1.0f,     0.0f,  1.0f,
    -0.5f, -0.5f,  0.5f,    0.0f,  0.0f,  1.0f,     0.0f,  0.0f,

    -0.5f,  0.5f,  0.5f,    -1.0f,  0.0f,  0.0f,    1.0f,  0.0f,
    -0.5f,  0.5f, -0.5f,    -1.0f,  0.0f,  0.0f,    1.0f,  1.0f,
    -0.5f, -0.5f, -0.5f,    -1.0f,  0.0f,  0.0f,    0.0f,  1.0f,
    -0.5f, -0.5f, -0.5f,    -1.0f,  0.0f,  0.0f,    0.0f,  1.0f,
    -0.5f, -0.5f,  0.5f,    -1.0f,  0.0f,  0.0f,    0.0f,  0.0f,
    -0.5f,  0.5f,  0.5f,    -1.0f,  0.0f,  0.0f,    1.0f,  0.0f,

    0.5f,  0.5f,  0.5f,     1.0f,  0.0f,  0.0f,     1.0f,  0.0f,
    0.5f,  0.5f, -0.5f,     1.0f,  0.0f,  0.0f,     1.0f,  1.0f,
    0.5f, -0.5f, -0.5f,     1.0f,  0.0f,  0.0f,     0.0f,  1.0f,
    0.5f, -0.5f, -0.5f,     1.0f,  0.0f,  0.0f,     0.0f,  1.0f,
    0.5f, -0.5f,  0.5f,     1.0f,  0.0f,  0.0f,     0.0f,  0.0f,
    0.5f,  0.5f,  0.5f,     1.0f,  0.0f,  0.0f,     1.0f,  0.0f,

    -0.5f, -0.5f, -0.5f,    0.0f, -1.0f,  0.0f,     0.0f,  1.0f,
    0.5f, -0.5f, -0.5f,     0.0f, -1.0f,  0.0f,     1.0f,  1.0f,
    0.5f, -0.5f,  0.5f,     0.0f, -1.0f,  0.0f,     1.0f,  0.0f,
    0.5f, -0.5f,  0.5f,     0.0f, -1.0f,  0.0f,     1.0f,  0.0f,
    -0.5f, -0.5f,  0.5f,    0.0f, -1.0f,  0.0f,     0.0f,  0.0f,
    -0.5f, -0.5f, -0.5f,    0.0f, -1.0f,  0.0f,     0.0f,  1.0f,

    -0.5f,  0.5f, -0.5f,    0.0f,  1.0f,  0.0f,     0.0f,  1.0f,
    0.5f,  0.5f, -0.5f,     0.0f,  1.0f,  0.0f,     1.0f,  1.0f,
    0.5f,  0.5f,  0.5f,     0.0f,  1.0f,  0.0f,     1.0f,  0.0f,
    0.5f,  0.5f,  0.5f,     0.0f,  1.0f,  0.0f,     1.0f,  0.0f,
    -0.5f,  0.5f,  0.5f,    0.0f,  1.0f,  0.0f,     0.0f,  0.0f,
    -0.5f,  0.5f, -0.5f,    0.0f,  1.0f,  0.0f,     0.0f,  1.0f
};
```

```
GLfloat skyboxVertices[] = {
    // Positions
    -1.0f,  1.0f, -1.0f,
    -1.0f, -1.0f, -1.0f,
    1.0f, -1.0f, -1.0f,
    1.0f, -1.0f, -1.0f,
    1.0f,  1.0f, -1.0f,
    -1.0f,  1.0f, -1.0f,

    -1.0f, -1.0f,  1.0f,
    -1.0f, -1.0f, -1.0f,
    -1.0f,  1.0f, -1.0f,
    -1.0f,  1.0f, -1.0f,
    -1.0f,  1.0f,  1.0f,
    -1.0f, -1.0f,  1.0f,

    1.0f, -1.0f, -1.0f,
    1.0f, -1.0f,  1.0f,
    1.0f,  1.0f,  1.0f,
    1.0f,  1.0f,  1.0f,
    1.0f,  1.0f, -1.0f,
    1.0f, -1.0f, -1.0f,

    -1.0f, -1.0f,  1.0f,
    -1.0f,  1.0f,  1.0f,
    1.0f,  1.0f,  1.0f,
    1.0f,  1.0f,  1.0f,
    1.0f, -1.0f,  1.0f,
    -1.0f, -1.0f,  1.0f,


    -1.0f,  1.0f, -1.0f,
    1.0f,  1.0f, -1.0f,
    1.0f,  1.0f,  1.0f,
    1.0f,  1.0f,  1.0f,
    -1.0f,  1.0f,  1.0f,
    -1.0f,  1.0f, -1.0f,

    -1.0f, -1.0f, -1.0f,
    -1.0f, -1.0f,  1.0f,
    1.0f, -1.0f, -1.0f,
    1.0f, -1.0f, -1.0f,
    -1.0f, -1.0f,  1.0f,
    1.0f, -1.0f,  1.0f
};

GLuint indices[] =
{  // Note that we start from 0!
    0,1,2,3,
    4,5,6,7,
    8,9,10,11,
    12,13,14,15,
    16,17,18,19,
    20,21,22,23,
    24,25,26,27,
    28,29,30,31,
    32,33,34,35
};

// 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
glEnableVertexAttribArray(0);
glBindVertexArray(0);

//SkyBox
GLuint skyboxVBO, skyboxVAO;
glGenVertexArrays(1, &skyboxVAO);
glGenBuffers(1, &skyboxVBO);
glBindVertexArray(skyboxVAO);
glBindBuffer(GL_ARRAY_BUFFER, skyboxVBO);
glBufferData(GL_ARRAY_BUFFER, sizeof(skyboxVertices), &skyboxVertices, GL_STATIC_DRAW);
glEnableVertexAttribArray(0);
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(GLfloat), (GLvoid *)0);

// Load textures
vector<const GLchar*> faces;
faces.push_back("SkyBox/derecha.tga");
faces.push_back("SkyBox/izquierda.tga");
faces.push_back("SkyBox/superior.tga");
faces.push_back("SkyBox/inferior.tga");
faces.push_back("SkyBox/atras.tga");
faces.push_back("SkyBox/frente.tga");

GLuint cubemapTexture = TextureLoading::LoadCubemap(faces);

GLuint texture1;
glGenTextures(1, &texture1);

int textureWidth, textureHeight, nrChannels;
stbi_set_flip_vertically_on_load(true);
unsigned char *image;
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR_MIPMAP_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST_MIPMAP_NEAREST);
// Diffuse map
image = stbi_load("Textura/mesa_cafe.png", &textureWidth, &textureHeight, &nrChannels, 0);
glBindTexture(GL_TEXTURE_2D, texture1);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, textureWidth, textureHeight, 0, GL_RGBA, GL_UNSIGNED_BYTE, image);
glGenerateMipmap(GL_TEXTURE_2D);
if (image)
{
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, textureWidth, textureHeight, 0, GL_RGBA, GL_UNSIGNED_BYTE, image);
    glGenerateMipmap(GL_TEXTURE_2D);
}
else
{
    std::cout << "Failed to load texture" << std::endl;
}
stbi_image_free(image);

// Set texture units
lightingShader.Use();
glUniform1i(glGetUniformLocation(lightingShader.Program, "material.diffuse"), 0);

glm::mat4 projection = glm::perspective(camera.GetZoom(), (GLfloat)SCREEN_WIDTH / (GLfloat)SCREEN_HEIGHT, 0.1f, 100.0f);
```

```
// 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();
    animacionMadera();
    animacionPuerta();
    animacionSillon();
    animacionCajon();
    animacionLibros();
    animacionAntena();

    // 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);

    // 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"), 0.4f, 0.4f, 0.4f);
    glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.specular"), 0.5f, 0.5f, 0.5f);
```

```
    // Point light 1
    glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].position"), 100.0f, 50.0f, 100.0f);
    glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].ambient"), 0.05f, 0.05f, 0.05f);
    glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].diffuse"), 1.0f, 1.0f, 1.0f);
    glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].specular"), 1.0f, 1.0f, 1.0f);
    glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[0].constant"), 1.0f);
    glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[0].linear"), 10.0f);
    glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[0].quadratic"), 20.0f);

    // 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
    glActiveTexture(GL_TEXTURE0);
    glBindTexture(GL_TEXTURE_2D, texture1);

    glBindVertexArray(VAO);
    glm::mat4 model(1);
```

```
//Mesa
model = glm::translate(model, glm::vec3(-106.8f, -1.47f, -76.2f));
model = glm::rotate(model, glm::radians(90.0f), glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::scale(model, glm::vec3(1.9f, 0.05f, 1.2f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glDrawArrays(GL_TRIANGLES, 0, 36);
```

```
//Pata 1
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-107.34f, -1.795f, -77.08f));
model = glm::scale(model, glm::vec3(0.05f, 0.6f, 0.05f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glDrawArrays(GL_TRIANGLES, 0, 36);

//Pata 2
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-106.276f, -1.795f, -77.08f));
model = glm::scale(model, glm::vec3(0.05f, 0.6f, 0.05f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glDrawArrays(GL_TRIANGLES, 0, 36);

//Pata 3
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-107.34f, -1.795f, -75.32f));
model = glm::scale(model, glm::vec3(0.05f, 0.6f, 0.05f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glDrawArrays(GL_TRIANGLES, 0, 36);

//Pata 4
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-106.276f, -1.795f, -75.32f));
model = glm::scale(model, glm::vec3(0.05f, 0.6f, 0.05f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glDrawArrays(GL_TRIANGLES, 0, 36);

//Soporte 1
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-106.81f, -1.69f, -77.08f));
model = glm::rotate(model, glm::radians(90.0f), glm::vec3(0.0f, 0.0f, 1.0f));
model = glm::scale(model, glm::vec3(0.02f, 1.02f, 0.02f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glDrawArrays(GL_TRIANGLES, 0, 36);
```

```
//Soporte 2
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-106.81f, -1.69f, -75.32f));
model = glm::rotate(model, glm::radians(90.0f), glm::vec3(0.0f, 0.0f, 1.0f));
model = glm::scale(model, glm::vec3(0.02f, 1.02f, 0.02f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glDrawArrays(GL_TRIANGLES, 0, 36);

//Carga de modelos
//Casa
view = camera.GetViewMatrix();
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-100.0f, -2.5f, -80.0f));
model = glm::rotate(model, glm::radians(180.0f), glm::vec3(0.0f, 1.0f, 0.0));
model = glm::scale(model, glm::vec3(0.02f, 0.02f, 0.02f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Casa.Draw(lightingShader);
```

```
//Piano
view = camera.GetViewMatrix();
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-105.7f, -2.11f, -76.9f));
model = glm::rotate(model, glm::radians(270.0f), glm::vec3(0.0f, 1.0f, 0.0));
model = glm::scale(model, glm::vec3(0.02f, 0.02f, 0.02f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Piano.Draw(lightingShader);

//Chimenea
view = camera.GetViewMatrix();
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-99.97f, -2.497f, -80.0f));
model = glm::rotate(model, glm::radians(180.0f), glm::vec3(0.0f, 1.0f, 0.0));
model = glm::scale(model, glm::vec3(0.02f, 0.02f, 0.02f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Chimenea.Draw(lightingShader);
```

```
//Sillón Verde
view = camera.GetViewMatrix();
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(desplazamiento, -2.51f, -80.8f));
model = glm::rotate(model, glm::radians(180.0f), glm::vec3(0.0f, 1.0f, 0.0));
model = glm::scale(model, glm::vec3(0.02f, 0.02f, 0.02f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
SillonVerde.Draw(lightingShader);

//Madera
view = camera.GetViewMatrix();
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-100.0f, movCaida, -79.99f));
model = glm::rotate(model, glm::radians(180.0f), glm::vec3(0.0f, 1.0f, 0.0));
model = glm::scale(model, glm::vec3(0.02f, 0.02f, 0.02f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Madera.Draw(lightingShader);

//Buro 1
view = camera.GetViewMatrix();
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-106.7f, -2.105f, -75.3f));
model = glm::rotate(model, glm::radians(270.0f), glm::vec3(0.0f, 1.0f, 0.0));
model = glm::scale(model, glm::vec3(0.02f, 0.02f, 0.02f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Buro.Draw(lightingShader);

//Cajón 1
view = camera.GetViewMatrix();
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(movi, -2.705f, -75.3f));
model = glm::rotate(model, glm::radians(270.0f), glm::vec3(0.0f, 1.0f, 0.0));
model = glm::scale(model, glm::vec3(0.02f, 0.02f, 0.02f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Cajon.Draw(lightingShader);
```

```
//Lámpara 1
view = camera.GetViewMatrix();
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-102.3f, -2.105f, -77.2f));
model = glm::scale(model, glm::vec3(0.02f, 0.02f, 0.02f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Lampara.Draw(lightingShader);
```

```
//Silón Café
view = camera.GetViewMatrix();
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-106.8f, -2.11f, -76.5f));
model = glm::rotate(model, glm::radians(270.0f), glm::vec3(0.0f, 1.0f, 0.0));
model = glm::scale(model, glm::vec3(0.02f, 0.02f, 0.02f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
SillonCafe.Draw(lightingShader);

//Buro 2
view = camera.GetViewMatrix();
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-106.7f, -2.105f, -69.2f));
model = glm::rotate(model, glm::radians(270.0f), glm::vec3(0.0f, 1.0f, 0.0));
model = glm::scale(model, glm::vec3(0.02f, 0.02f, 0.02f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Buro.Draw(lightingShader);

//Cajón 2
view = camera.GetViewMatrix();
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(movi, -2.705f, -69.2f));
model = glm::rotate(model, glm::radians(270.0f), glm::vec3(0.0f, 1.0f, 0.0));
model = glm::scale(model, glm::vec3(0.02f, 0.02f, 0.02f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Cajon.Draw(lightingShader);
```

```
//Lámpara 2
view = camera.GetViewMatrix();
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-102.3f, -2.105f, -71.1f));
model = glm::scale(model, glm::vec3(0.02f, 0.02f, 0.02f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Lampara.Draw(lightingShader);

//Puerta
view = camera.GetViewMatrix();
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-99.5533f, 0.0548f, -70.8918f));
model = glm::rotate(model, glm::radians(rotP), glm::vec3(0.0f, 1.0f, 0.0));
model = glm::scale(model, glm::vec3(0.02f, 0.02f, 0.02f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Puerta.Draw(lightingShader);

//Sillón Café (Segundo Cuarto)
view = camera.GetViewMatrix();
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-104.8f, -2.11f, -85.7f));
model = glm::rotate(model, glm::radians(270.0f), glm::vec3(0.0f, 1.0f, 0.0));
model = glm::scale(model, glm::vec3(0.02f, 0.02f, 0.02f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
SillonCafe.Draw(lightingShader);

//TV
view = camera.GetViewMatrix();
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-105.3f, -2.09f, -85.2f));
model = glm::rotate(model, glm::radians(270.0f), glm::vec3(0.0f, 1.0f, 0.0));
model = glm::scale(model, glm::vec3(0.02f, 0.02f, 0.02f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
TV.Draw(lightingShader);
```

```
//Caja del cable
view = camera.GetViewMatrix();
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-105.3f, -2.09f, -85.2f));
model = glm::rotate(model, glm::radians(270.0f), glm::vec3(0.0f, 1.0f, 0.0));
model = glm::scale(model, glm::vec3(0.02f, 0.02f, 0.02f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
CajaCable.Draw(lightingShader);

//Antena
view = camera.GetViewMatrix();
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(antMov.x, -2.09f, antMov.z));
model = glm::rotate(model, glm::radians(270.0f), glm::vec3(0.0f, 1.0f, 0.0));
model = glm::scale(model, glm::vec3(0.02f, 0.02f, 0.02f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Antena.Draw(lightingShader);

//Lámpara de piso
view = camera.GetViewMatrix();
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-104.81f, -2.099f, -85.8f));
model = glm::rotate(model, glm::radians(270.0f), glm::vec3(0.0f, 1.0f, 0.0));
model = glm::scale(model, glm::vec3(0.02f, 0.02f, 0.02f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
LamparaP.Draw(lightingShader);

//Bocina 1
view = camera.GetViewMatrix();
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-106.5f, -2.096f, -83.9f));
model = glm::rotate(model, glm::radians(270.0f), glm::vec3(0.0f, 1.0f, 0.0));
model = glm::scale(model, glm::vec3(0.02f, 0.02f, 0.02f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Bocina.Draw(lightingShader);
```

```
//Librero
view = camera.GetViewMatrix();
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-106.5f, -2.096f, -83.9f));
model = glm::rotate(model, glm::radians(270.0f), glm::vec3(0.0f, 1.0f, 0.0));
model = glm::scale(model, glm::vec3(0.02f, 0.02f, 0.02f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Librero.Draw(lightingShader);

//Libros
view = camera.GetViewMatrix();
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(libMov, -2.096f, -83.9f));
model = glm::rotate(model, glm::radians(270.0f), glm::vec3(0.0f, 1.0f, 0.0));
model = glm::scale(model, glm::vec3(0.02f, 0.02f, 0.02f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Libros.Draw(lightingShader);

//Bocina 2
view = camera.GetViewMatrix();
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-109.99f, -2.096f, -83.9f));
model = glm::rotate(model, glm::radians(270.0f), glm::vec3(0.0f, 1.0f, 0.0));
model = glm::scale(model, glm::vec3(0.02f, 0.02f, 0.02f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Bocina.Draw(lightingShader);

glBindVertexArray(0);

// Draw skybox as last
glDepthFunc(GL_LEQUAL);  // Change depth function so depth test passes when values are equal to depth buffer's content
SkyBoxShader.Use();
view = glm::mat4(glm::mat3(camera.GetViewMatrix()));    // Remove any translation component of the view matrix
glUniformMatrix4fv(glGetUniformLocation(SkyBoxShader.Program, "view"), 1, GL_FALSE, glm::value_ptr(view));
glUniformMatrix4fv(glGetUniformLocation(SkyBoxShader.Program, "projection"), 1, GL_FALSE, glm::value_ptr(projection));
```

```
        // skybox cube
        glBindVertexArray(skyboxVAO);
        glActiveTexture(GL_TEXTURE1);
        glBindTexture(GL_TEXTURE_CUBE_MAP, cubemapTexture);
        glDrawArrays(GL_TRIANGLES, 0, 36);
        glBindVertexArray(0);
        glDepthFunc(GL_LESS); // Set depth function back to default

        // Swap the screen buffers
        glfwSwapBuffers(window);
    }

    glDeleteVertexArrays(1, &VAO);
    glDeleteVertexArrays(1, &lightVAO);
    glDeleteBuffers(1, &VBO);
    glDeleteBuffers(1, &EBO);
    glDeleteVertexArrays(1, &skyboxVAO);
    glDeleteBuffers(1, &skyboxVBO);
    // Terminate GLFW, clearing any resources allocated by GLFW.
    glfwTerminate();



    return 0;
}

// Moves/alters the camera positions based on user input
void DoMovement()
{
    // Camera controls
    if (keys[GLFW_KEY_W] || keys[GLFW_KEY_UP])
    {
        camera.ProcessKeyboard(FORWARD, deltaTime);

    }

    if (keys[GLFW_KEY_S] || keys[GLFW_KEY_DOWN])
    {
        camera.ProcessKeyboard(BACKWARD, deltaTime);


    }

    if (keys[GLFW_KEY_A] || keys[GLFW_KEY_LEFT])
    {
        camera.ProcessKeyboard(LEFT, deltaTime);


    }

    if (keys[GLFW_KEY_D] || keys[GLFW_KEY_RIGHT])
    {
        camera.ProcessKeyboard(RIGHT, deltaTime);


    }

    //Activa la animación de la madera
    if (keys[GLFW_KEY_R])
    {
        activo = true;
    }

    //Activa la animación de los cajones
    if (keys[GLFW_KEY_F])
    {
        activoC = true;
    }

    //Activa la animación del sillón verde
    if (keys[GLFW_KEY_E])
    {
        activoS = true;
    }
```

```
    //Activa la animación de la puerta
    if (keys[GLFW_KEY_SPACE])
    {
        activoP = true;
    }

    //Activa la animación de los libros
    if (keys[GLFW_KEY_G])
    {
        libAct = true;
    }

    //Activa el desplazamiento de la antena en X
    if (keys[GLFW_KEY_X])
    {
        antActX = true;
    }

    //Activa el desplazamiento de la antena en Z
    if (keys[GLFW_KEY_Z])
    {
        antActZ = true;
    }

    //Activa el desplazamiento de la antena para centrarla
    if (keys[GLFW_KEY_C])
    {
        antActC = true;
    }
}

//Animación del dispensador de madera
void animacionMadera()
{
    if (activo)
    {
        if (seleccion)
        {

            if (movCaida < -2.5)
            {
                movCaida = -2.5f;
                activo = false;
                seleccion = false;
            }
            else
            {
                movCaida -= 0.05f;
            }
        }
        else
        {
            if (movCaida < -3.051)
            {
                movCaida = -0.823f;
                activo = false;
                seleccion = true;
            }
            else
            {
                movCaida -= 0.05f;
            }
        }
    }
}
```

```
//Animación Sillón Verde
void animacionSillon() {
    if (activoS)
    {
        if (desplazado)
        {
            if (dentro)
            {

                if (desplazamiento > -100)
                {
                    desplazamiento = -100.0f;
                    activoS = false;
                    desplazado = false;
                }
                else
                {
                    desplazamiento += 0.05f;
                }
            }
            else
            {
                activoS = false;
            }
        }
        else
        {
            if (desplazamiento < -101.68)
            {
                desplazamiento = -101.68f;
                activoS = false;
                desplazado = true;
            }
            else
            {
                desplazamiento -= 0.05f;
            }
        }
    }
}

//Animación de los cajones
void animacionCajon()
{
    if (activoC)
    {
        if (dentro)
        {

            if (desplazado)
            {
                if (movi < -107.64)
                {
                    movi = -107.64f;
                    activoC = false;
                    dentro = false;
                }
                else
                {
                    movi -= 0.05f;
                }
            }
            else
            {
                activoC = false;
            }
        }
```

```
        else
        {
            if (movi > -106.7)
            {
                movi = -106.7f;
                activoC = false;
                dentro = true;
            }
            else
            {
                movi += 0.05f;
            }
        }
    }
}

//Animación de la puerta
void animacionPuerta()
{
    if (activoP)
    {
        if (abierta)
        {
            if (rotP > 180)
            {
                rotP = 180.0f;
                activoP = false;
                abierta = false;
            }
            else
            {
                rotP += 1.0f;
            }
        }
        else
        {
            if (rotP < 90)
            {
                rotP = 90.0f;
                activoP = false;
                abierta = true;
            }
            else
            {
                rotP -= 1.0f;
            }
        }
    }
}

//Animación de los libros
void animacionLibros()
{
    if (libAct)
    {
        if (libDes)
        {
            if (libMov < -106.77)
            {
                libMov = -106.77f;
                libAct = false;
                libDes = false;
            }
            else
            {
                libMov -= 0.05f;
            }
        }
```

```
        else
        {
            if (libMov > -105.03)
            {
                libMov = -105.03f;
                libAct = false;
                libDes = true;
            }
            else
            {
                libMov += 0.05f;
            }
        }
    }
}

//Animación de la antena
void animacionAntena()
{
    if (antActX)
    {
        if (antDX)
        {
            if (antMov.x < -105.45)
            {
                antMov.x = -105.45f;
                antActX = false;
                antDX = false;
            }
            else
            {
                antMov.x -= 0.01f;
            }
        }
        else
        {
            if (antMov.x > -105.17)
            {
                antMov.x = -105.17f;
                antActX = false;
                antDX = true;
            }
            else
            {
                antMov.x += 0.01f;
            }
        }
    }
    else if (antActZ)
    {
        if (antDZ)
        {
```

```
            if (antMov.z < -85.59)
            {
                antMov.z = -85.59f;
                antActZ = false;
                antDZ = false;
            }
            else
            {
                antMov.z -= 0.01f;
            }
        }
        else
        {
            if (antMov.z > -84.79)
            {
                antMov.z = -84.79f;
                antActZ = false;
                antDZ = true;
            }
            else
            {
                antMov.z += 0.01f;
            }
        }
    }
    else if (antActC)
    {
        if (antMov.x == -105.3f && antMov.z == -85.2f)
        {
            antActC = false;
        }
        else
        {
            if (antDX)
            {

                if (antMov.x < -105.3)
                {
                    antMov.x = -105.3f;
                    antDX = false;
                }
                else
                {
                    antMov.x -= 0.01f;
                }
            }
            else
            {
                if (antMov.x > -105.3)
                {
                    antMov.x = -105.3f;
                    antDX = false;
                }
                else
                {
                    antMov.x += 0.01f;
                }
            }

            if (antDZ)
            {
                if (antMov.z < -85.2)
                {
                    antMov.z = -85.2f;
                    antDZ = false;
                }
```

```
                else
                {
                    antMov.z -= 0.01f;
                }
            }
            else
            {

                if (antMov.z > -85.2)
                {
                    antMov.z = -85.2f;
                    antDZ = false;
                }
                else
                {
                    antMov.z += 0.01f;
                }
            }
        }
    }
}

// Is called whenever a key is pressed/released via GLFW
void KeyCallback(GLFWwindow *window, int key, int scancode, int action, int mode)
{
    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;
        }
    }
}


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);
}
```