Manual Técnico.

Universidad Nacional Autónoma de México.

Facultad de Ingeniería.

Laboratorio de computación gráfica e interacción humano computadora.

Grupo: 04.

Profesor:

Ing. Roman Balbuena Carlos Aldair.

Manual técnico.

Alumnos:

Cruz Ramírez Cesar Alejandro

Rojas Ruiz Luis Enrique

Semestre 2021-2.

Contenido

Objetivos.

Mediante este proyecto los alumnos deberán aplicar y demostrar los conocimientos adquiridos durante todo el curso.

Alcance del proyecto.

Para este proyecto escogimos una fachada y un espacio para hacer una recreación 3D en OpenGL.

Imagen de referencia de la fachada.

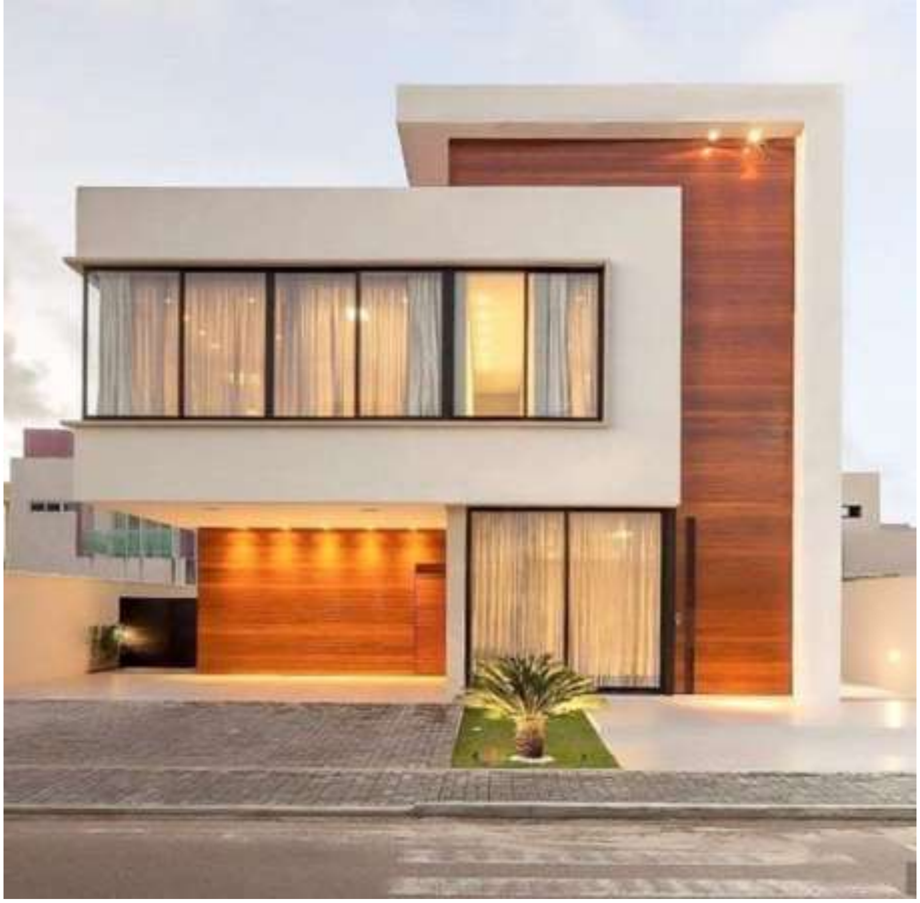
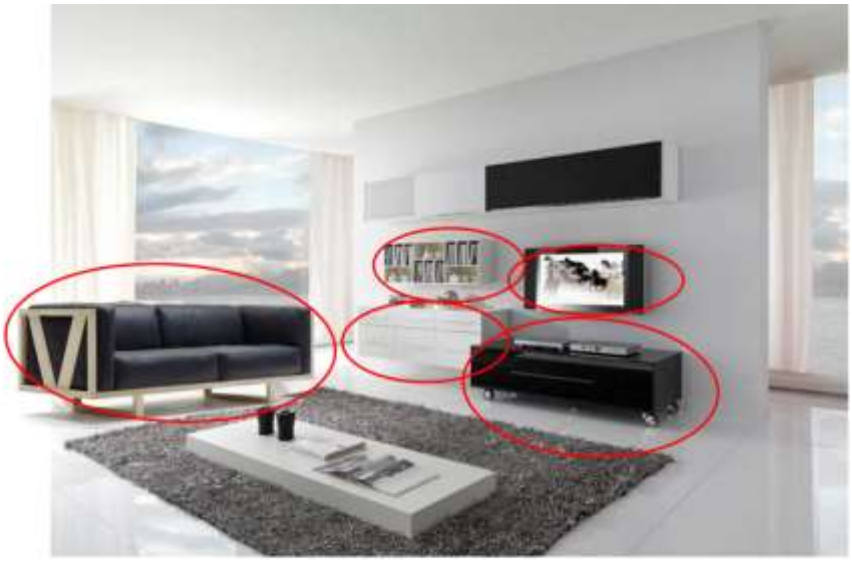
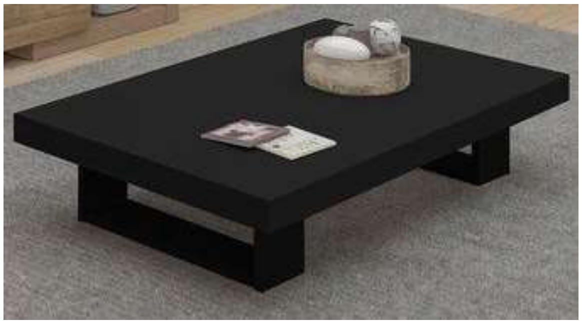


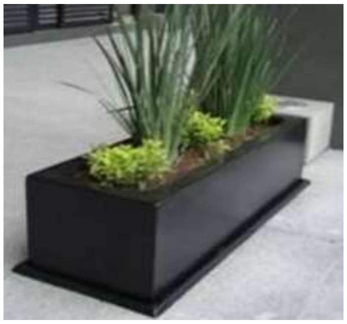
Imagen de referencia de espacio a recrear.



En la imagen anterior se muestran 5 de los objetos que serán recreados, pero para este proyecto deben ser mínimo 7 los objetos recreados por lo que decidimos agregar dos objetos más que no se encuentran en la imagen anterior.

Objetos que serán agregados al espacio a recrear.





Para este proyecto se buscó que todos los objetos tuvieran una buena geometría además de tener un buen texturizado.

También se busco crear 3 animaciones sencillas y 2 animaciones complejas.

Plan de trabajo.

Actividad.	S1	S2	S3	S4	S5	S6	S7.
Propuesta de proyecto.	🟥						
Búsqueda de modelos a utilizar.	🟦	🟦					
Texturizado de modelos		🟪	🟪				
Carga de modelos en OpenGL				🟥	🟥	🟥	🟥
Creación del Skybox				🟧	🟧		
Creación de animaciones				🟪	🟪		
Corrección de errores.						🟩	🟩
Entrega de proyecto							🟩

Calendario.

Semana 1: 15 de marzo – 19 de marzo.

Semana 2: 21 de junio – 25 de junio.

Semana 3: 28 de junio – 02 de julio.

Semana 4: 05 de julio – 09 de julio.

Semana 5: 12 de julio – 16 de julio.

Semana 6: 19 de julio – 23 de julio.

Semana 7: 26 de julio – 30 de julio.

Herramientas de trabajo.

Autodesk Maya.

Github.

Visual Studio 2019.

Google Meet.

Telegram.

Limitantes.

Durante el desarrollo del proyecto hubo diversos factores que limitaron el trabajo en este proyecto.

Al ser un semestre en línea los participantes del equipo trabajamos de manera virtual utilizando las herramientas mencionadas en el punto anterior.

Los participantes del equipo no cuentan con un equipo actualizado en cuanto a hardware por lo que en algunas de las herramientas no se podía realizar un trabajo fluido.

Durante este semestre se presento un paró de labores por lo que el semestre se comprimió y algunos de los conceptos no fueron comprendidos por completo, por lo que se presentaron algunos problemas durante el desarrollo del proyecto.

Resultados.

En las siguientes imágenes podremos apreciar una comparación de las imágenes de referencia contra el modelo 3D generado.

Imagen de referencia.

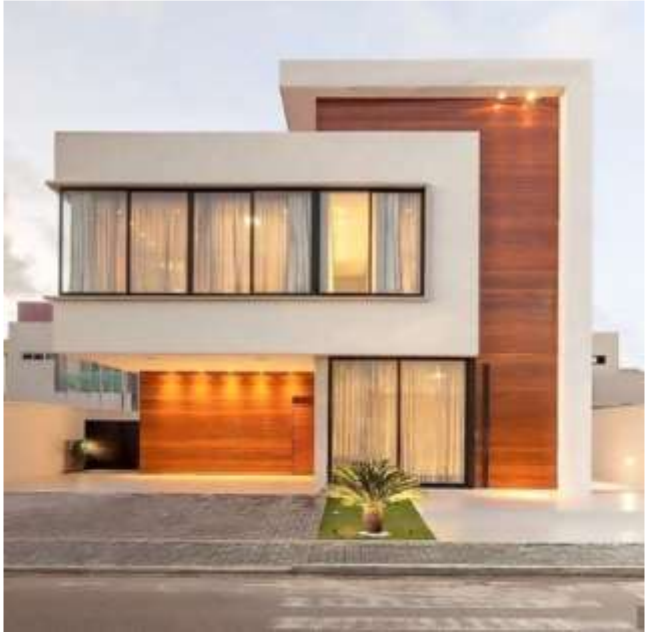


Imagen de modelo 3D.

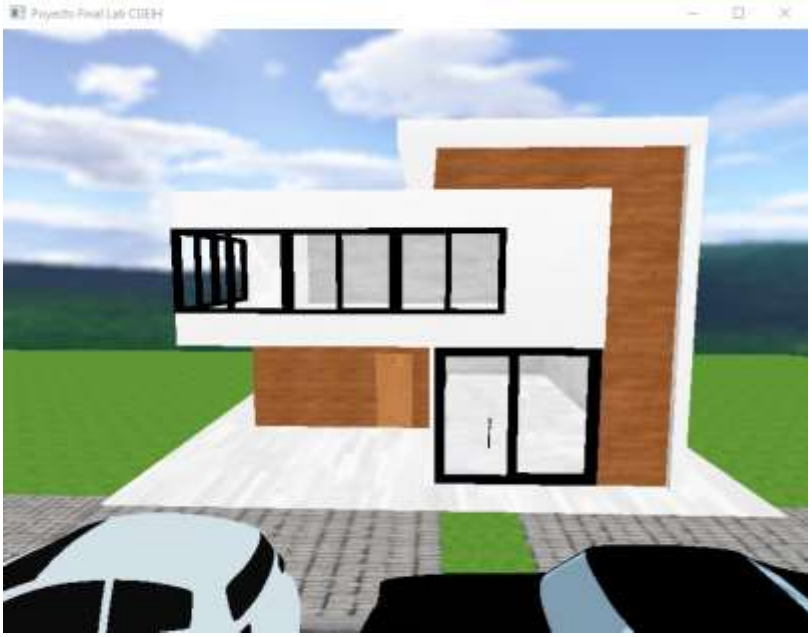


Imagen de referencia.

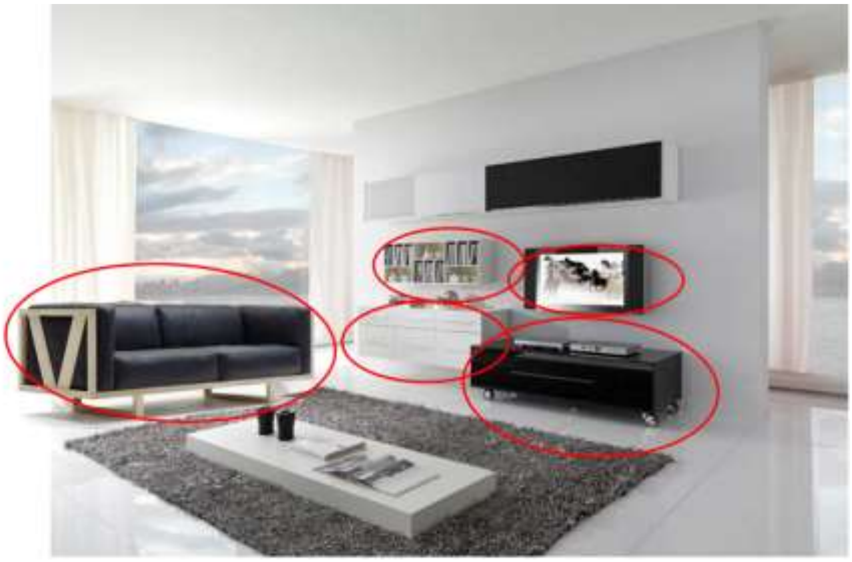
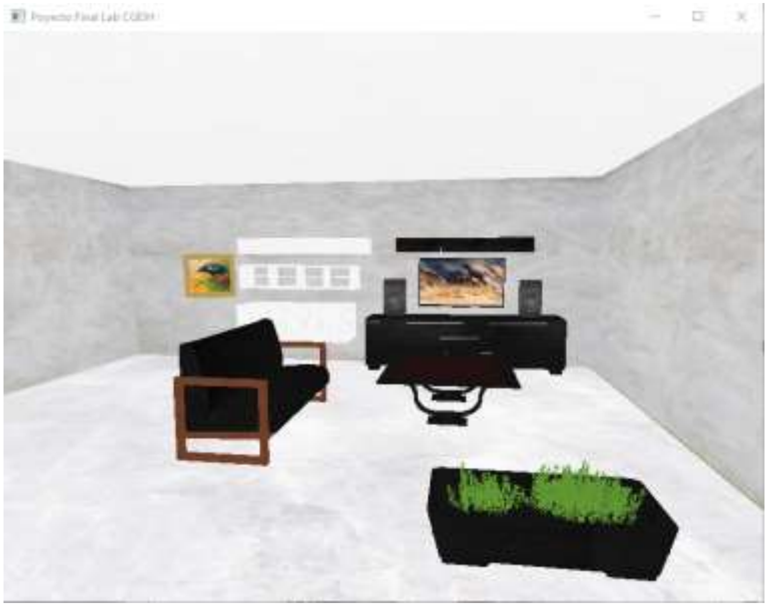


Imagen de modelo con los 7 objetos de referencia 3D.



Documentación del código.

```
#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 animacion();

// 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;
float range = 0.0f;
float rot = 0.0f;



// Light attributes
glm::vec3 lightPos(0.0f, 0.0f, 1.0f);
glm::vec3 PosIni(-95.0f, 1.0f, -45.0f);
bool active;

//SkyBox Variables.
int iluminacion = 0;
bool sky = false;
int numsky;

// Variables para animación de pantalla.
float TvX = 2.0;
float TvY = -7.0;
float TvZ = 4.0;

bool movimientoTv = false;
bool movimiento1 = true;
bool movimiento2 = false;
bool movimiento3 = false;

//Variables para animacion de puerta

float rotpuerta = 180.0f;

bool movimientoPuerta = false;
bool movP1 = true;
bool movP2 = false;
bool movP3 = false;

//Variables para animación de auto 1.
//Animación del coche
float movKitX = 2.0;
float movKitZ = 4.0;
float rotKit = 0.0;

bool circuito = false;
bool recorrido1 = true;
bool recorrido2 = false;
bool recorrido3 = false;
bool recorrido4 = false;
bool recorrido5 = false;
bool recorrido6 = false;
bool recorrido7 = false;

float movKitX2 = 2.0;
float movKitZ2 = 4.0;
float rotKit2 = 0.0;

bool circuito2 = false;
bool recorridoA = true;
bool recorridoB = false;
bool recorridoC = false;
bool recorridoD = false;
bool recorridoE = false;
bool recorridoF = false;
bool recorridoG = false;



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

// Positions of the point lights
glm::vec3 pointLightPositions[] = {
	/*glm::vec3(posX,posY,posZ),*/
	glm::vec3(0,0,0),
	glm::vec3(0,0,0),
	glm::vec3(0,0,0)
};

glm::vec3 LightP1;



int main()
{
	// Init GLFW
	glfwInit();



	// Create a GLFWwindow object that we can use for GLFW's functions
	GLFWwindow* window = glfwCreateWindow(WIDTH, HEIGHT, "Poyecto Final Lab CGEIH", 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);
	printf("%f", glfwGetTime());

	// 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/lighting.frag");
Shader lampShader("Shaders/lamp.vs", "Shaders/lamp.frag");
Shader SkyBoxshader("Shaders/SkyBox.vs", "Shaders/SkyBox.frag");

// Setup and compile our shaders
Shader shader("Shaders/modelLoading.vs", "Shaders/modelLoading.frag");

Model Casa((char*)"Models/Casa/Casa.obj");
Model Repisa((char*)"Models/Repisa/Repisa.obj");
Model Repisa2((char*)"Models/Repisa_2/Repisa2.obj");
Model Repisa3((char*)"Models/Repisa_3/Repisa_3.obj");
Model Bocina((char*)"Models/Bocina/Audio.obj");
Model Cajonera((char*)"Models/Cajonera/Cajonera.obj");
Model Cuadro((char*)"Models/Cuadro/Cuadro.obj");
Model MesaTv((char*)"Models/Mesa/MesaTV.obj");
Model MesaCentro((char*)"Models/Mesa_2/Mesa_2.obj");
Model Tv((char*)"Models/TV/SmartTV.obj");
Model Calle((char*)"Models/Calle/Calle.obj");
Model Auto((char*)"Models/Auto/Auto.obj");
Model Llantasdelanteras((char*)"Models/Auto/Llantasdelanteras.obj");
Model Llantastraseras((char*)"Models/Auto/Llantastraseras.obj");
Model Auto2((char*)"Models/Auto2/Auto2.obj");
Model Llantasdelanteras2((char*)"Models/Auto2/Llantasdelanteras2.obj");
Model Llantastraseras2((char*)"Models/Auto2/Llantastraseras2.obj");
Model Puerta((char*)"Models/Puerta/Puerta.obj");
Model Marcopuerta((char*)"Models/Puerta/Marcopuerta.obj");
Model Sillon((char*)"Models/Sillon/Sillon.obj");
Model Maceta((char*)"Models/Maceta/Maceta.obj");
Model Comedor((char*)"Models/Comedor/Comedor.obj");
```

```
// 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
        -10.0f,  10.0f, -10.0f,
        -10.0f, -10.0f, -10.0f,
         10.0f, -10.0f, -10.0f,
         10.0f, -10.0f, -10.0f,
         10.0f,  10.0f, -10.0f,
        -10.0f,  10.0f, -10.0f,

        -10.0f, -10.0f,  10.0f,
        -10.0f, -10.0f, -10.0f,
        -10.0f,  10.0f, -10.0f,
        -10.0f,  10.0f, -10.0f,
        -10.0f,  10.0f,  10.0f,
        -10.0f, -10.0f,  10.0f,

        10.0f, -10.0f, -10.0f,
        10.0f, -10.0f,  10.0f,
        10.0f,  10.0f,  10.0f,
        10.0f,  10.0f,  10.0f,
        10.0f,  10.0f, -10.0f,
        10.0f, -10.0f, -10.0f,

        -10.0f, -10.0f,  10.0f,
        -10.0f,  10.0f,  10.0f,
         10.0f,  10.0f,  10.0f,
         10.0f,  10.0f,  10.0f,
         10.0f, -10.0f,  10.0f,
        -10.0f, -10.0f,  10.0f,

        -10.0f,  10.0f, -10.0f,
         10.0f,  10.0f, -10.0f,
         10.0f,  10.0f,  10.0f,
         10.0f,  10.0f,  10.0f,
        -10.0f,  10.0f,  10.0f,
        -10.0f,  10.0f, -10.0f,

        -10.0f, -10.0f, -10.0f,
        -10.0f, -10.0f,  10.0f,
         10.0f, -10.0f, -10.0f,
         10.0f, -10.0f, -10.0f,
        -10.0f, -10.0f,  10.0f,
         10.0f, -10.0f,  10.0f
};


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
};

// 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);



	//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/SkyRight.tga");
        faces.push_back("SkyBox/SkyLeft.tga");
        faces.push_back("SkyBox/SkyTop.tga");
        faces.push_back("SkyBox/SkyBottom.tga");
        faces.push_back("SkyBox/SkyBack.tga");
        faces.push_back("SkyBox/SkyFront.tga");

        vector<const GLchar*> faces2;
        faces2.push_back("SkyBox/SkyRight2.tga");
        faces2.push_back("SkyBox/SkyLeft2.tga");
        faces2.push_back("SkyBox/SkyTop2.tga");
        faces2.push_back("SkyBox/SkyBottom2.tga");
        faces2.push_back("SkyBox/SkyBack2.tga");
        faces2.push_back("SkyBox/SkyFront2.tga");

        vector<const GLchar*> mysky[] = { faces, faces2 };
        GLuint cubemapTexture = TextureLoading::LoadCubemap(mysky[0]);
        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. This can be done more code-friendly
		// by defining light types as classes and set their values in there, or by using a more efficient uniform approach
		// by using 'Uniform buffer objects', but that is something we discuss in the 'Advanced GLSL' tutorial.
		// == ==========================
		// Directional light
```

```
        glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.direction"), -
0.2f, -1.0f, -0.3f);
        switch (iluminacion)
        {
        case 0:
            glUniform3f(glGetUniformLocation(lightingShader.Program,
"dirLight.ambient"), 1.0f, 1.0f, 1.0f);
            break;
        case 1:
            glUniform3f(glGetUniformLocation(lightingShader.Program,
"dirLight.ambient"), 0.6f, 0.6f, 0.6f);
            break;
        default:
            break;
        }
        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"), pointLightPositions[0].x, pointLightPositions[0].y, pointLightPositions[0].z);
        glUniform3f(glGetUniformLocation(lightingShader.Program,
"pointLights[0].ambient"), 0.05f, 0.05f, 0.05f);
        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));
		glBindVertexArray(VAO);
		glm::mat4 tmp = glm::mat4(1.0f); //Temp

		 // Clear the colorbuffer
		glClearColor(0.0f, 0.0f, 0.5f, 1.0f);
		glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

		shader.Use();

		//glm::mat4 view = camera.GetViewMatrix();
		glUniformMatrix4fv(glGetUniformLocation(shader.Program, "projection"), 1, 
GL_FALSE, glm::value_ptr(projection));
		glUniformMatrix4fv(glGetUniformLocation(shader.Program, "view"), 1, GL_FALSE, 
glm::value_ptr(view));


		//Carga de modelos.

		//Repisa
		view = camera.GetViewMatrix();
		glm::mat4 model(1);
		model = glm::translate(model, glm::vec3(2.0f, -7.0f, 4.0f));
		model = glm::rotate(model, glm::radians(180.0f), glm::vec3(0.0f, 1.0f, 0.0f));
		model = glm::scale(model, glm::vec3(0.05f, 0.05f, 0.05f));
```

```
			glUniformMatrix4fv(glGetUniformLocation(shader.Program, "model"), 1,
GL_FALSE, glm::value_ptr(model));
			Repisa.Draw(lightingShader);


			//Repisa2
			view = camera.GetViewMatrix();
			model = glm::mat4(1);
			model = glm::translate(model, glm::vec3(2.0f, -7.0f, 4.0f));
			model = glm::rotate(model, glm::radians(180.0f), glm::vec3(0.0f, 1.0f, 0.0f));
			model = glm::scale(model, glm::vec3(0.05f, 0.05f, 0.05f));
			glUniformMatrix4fv(glGetUniformLocation(shader.Program, "model"), 1,
GL_FALSE, glm::value_ptr(model));
			Repisa2.Draw(lightingShader);


			//Repisa3
			view = camera.GetViewMatrix();
			model = glm::mat4(1);
			model = glm::translate(model, glm::vec3(2.0f, -7.0f, 4.0f));
			model = glm::rotate(model, glm::radians(180.0f), glm::vec3(0.0f, 1.0f, 0.0f));
			model = glm::scale(model, glm::vec3(0.05f, 0.05f, 0.05f));
			glUniformMatrix4fv(glGetUniformLocation(shader.Program, "model"), 1,
GL_FALSE, glm::value_ptr(model));
			Repisa3.Draw(lightingShader);


			//Bocina
			view = camera.GetViewMatrix();
			model = glm::mat4(1);
			model = glm::translate(model, glm::vec3(2.0f, -7.0f, 4.0f));
			model = glm::rotate(model, glm::radians(180.0f), glm::vec3(0.0f, 1.0f, 0.0f));
			model = glm::scale(model, glm::vec3(0.05f, 0.05f, 0.05f));
			glUniformMatrix4fv(glGetUniformLocation(shader.Program, "model"), 1,
GL_FALSE, glm::value_ptr(model));
			Bocina.Draw(lightingShader);


			//Cajonera
```

```
                view = camera.GetViewMatrix();
                model = glm::mat4(1);
                model = glm::translate(model, glm::vec3(2.0f, -7.0f, 4.0f));
                model = glm::rotate(model, glm::radians(180.0f), glm::vec3(0.0f, 1.0f, 0.0f));
                model = glm::scale(model, glm::vec3(0.05f, 0.05f, 0.05f));
                glUniformMatrix4fv(glGetUniformLocation(shader.Program, "model"), 1, 
        GL_FALSE, glm::value_ptr(model));
                Cajonera.Draw(lightingShader);


                //Cuadro
                view = camera.GetViewMatrix();
                model = glm::mat4(1);
                model = glm::translate(model, glm::vec3(2.0f, -7.0f, 4.0f));
                model = glm::rotate(model, glm::radians(180.0f), glm::vec3(0.0f, 1.0f, 0.0f));
                model = glm::scale(model, glm::vec3(0.05f, 0.05f, 0.05f));
                glUniformMatrix4fv(glGetUniformLocation(shader.Program, "model"), 1, 
        GL_FALSE, glm::value_ptr(model));
                Cuadro.Draw(lightingShader);


                //MesaTv
                view = camera.GetViewMatrix();
                model = glm::mat4(1);
                model = glm::translate(model, glm::vec3(2.0f, -7.0f, 4.0f));
                model = glm::rotate(model, glm::radians(180.0f), glm::vec3(0.0f, 1.0f, 0.0f));
                model = glm::scale(model, glm::vec3(0.05f, 0.05f, 0.05f));
                glUniformMatrix4fv(glGetUniformLocation(shader.Program, "model"), 1, 
        GL_FALSE, glm::value_ptr(model));
                MesaTv.Draw(lightingShader);


                //Mesa de centro.
                view = camera.GetViewMatrix();
                model = glm::mat4(1);
                model = glm::translate(model, glm::vec3(2.0f, -7.0f, 4.0f));
                model = glm::rotate(model, glm::radians(180.0f), glm::vec3(0.0f, 1.0f, 0.0f));
```

```
            model = glm::scale(model, glm::vec3(0.05f, 0.05f, 0.05f));

            glUniformMatrix4fv(glGetUniformLocation(shader.Program, "model"), 1,
GL_FALSE, glm::value_ptr(model));

            MesaCentro.Draw(lightingShader);


            //Pantalla.

            view = camera.GetViewMatrix();

            model = glm::mat4(1);

            model = glm::translate(model, glm::vec3(TvX, TvY, TvZ));

            model = glm::rotate(model, glm::radians(180.0f), glm::vec3(0.0f, 1.0f, 0.0f));

            model = glm::scale(model, glm::vec3(0.05f, 0.05f, 0.05f));

            glUniformMatrix4fv(glGetUniformLocation(shader.Program, "model"), 1,
GL_FALSE, glm::value_ptr(model));

            Tv.Draw(lightingShader);


            //Casa.

            view = camera.GetViewMatrix();

            model = glm::mat4(1);

            model = glm::translate(model, glm::vec3(0.0f, -7.0f, 4.0f));

            model = glm::rotate(model, glm::radians(180.0f), glm::vec3(0.0f, 1.0f, 0.0f));

            model = glm::scale(model, glm::vec3(0.05f, 0.05f, 0.05f));

            glUniformMatrix4fv(glGetUniformLocation(shader.Program, "model"), 1,
GL_FALSE, glm::value_ptr(model));

            Casa.Draw(lightingShader);


            //Calle

            view = camera.GetViewMatrix();

            model = glm::mat4(1);

            model = glm::translate(model, glm::vec3(2.0f, -7.0f, 4.0f));

            model = glm::rotate(model, glm::radians(180.0f), glm::vec3(0.0f, 1.0f, 0.0f));

            model = glm::scale(model, glm::vec3(0.05f, 0.05f, 0.05f));

            glUniformMatrix4fv(glGetUniformLocation(shader.Program, "model"), 1,
GL_FALSE, glm::value_ptr(model));

            Calle.Draw(lightingShader);
```

```
			//Auto

			view = camera.GetViewMatrix();

			model = glm::mat4(1);

			model = glm::translate(model, glm::vec3(movKitX, -7.0f, movKitZ));

			model = glm::rotate(model, glm::radians(rotKit), glm::vec3(0.0f, 1.0f, 0.0));

			model = glm::rotate(model, glm::radians(180.0f), glm::vec3(0.0f, 1.0f, 0.0f));

			model = glm::scale(model, glm::vec3(0.05f, 0.05f, 0.05f));

			glUniformMatrix4fv(glGetUniformLocation(shader.Program, "model"), 1,
GL_FALSE, glm::value_ptr(model));

			Auto.Draw(lightingShader);


			//Llatasdelanteras

			view = camera.GetViewMatrix();

			model = glm::mat4(1);

			model = glm::translate(model, glm::vec3(movKitX, -7.0f, movKitZ));

			model = glm::rotate(model, glm::radians(rotKit), glm::vec3(0.0f, 1.0f, 0.0));

			model = glm::rotate(model, glm::radians(180.0f), glm::vec3(0.0f, 1.0f, 0.0f));

			model = glm::scale(model, glm::vec3(0.05f, 0.05f, 0.05f));

			glUniformMatrix4fv(glGetUniformLocation(shader.Program, "model"), 1,
GL_FALSE, glm::value_ptr(model));

			Llantasdelanteras.Draw(lightingShader);


			//Llatastraseras

			view = camera.GetViewMatrix();

			model = glm::mat4(1);

			model = glm::translate(model, glm::vec3(movKitX, -7.0f, movKitZ));

			model = glm::rotate(model, glm::radians(rotKit), glm::vec3(0.0f, 1.0f, 0.0));

			model = glm::rotate(model, glm::radians(180.0f), glm::vec3(0.0f, 1.0f, 0.0f));

			model = glm::scale(model, glm::vec3(0.05f, 0.05f, 0.05f));

			glUniformMatrix4fv(glGetUniformLocation(shader.Program, "model"), 1,
GL_FALSE, glm::value_ptr(model));

			Llantastraseras.Draw(lightingShader);


			//Auto2
```

```
            view = camera.GetViewMatrix();
            model = glm::mat4(1);
            model = glm::translate(model, glm::vec3(movKitX2, -7.0f, movKitZ2));
            model = glm::rotate(model, glm::radians(180.0f), glm::vec3(0.0f, 1.0f, 0.0f));
            model = glm::scale(model, glm::vec3(0.05f, 0.05f, 0.05f));
            glUniformMatrix4fv(glGetUniformLocation(shader.Program, "model"), 1,
GL_FALSE, glm::value_ptr(model));
            Auto2.Draw(lightingShader);


            //Llatasdelanteras
            view = camera.GetViewMatrix();
            model = glm::mat4(1);
            model = glm::translate(model, glm::vec3(movKitX2, -7.0f, movKitZ2));
            model = glm::rotate(model, glm::radians(180.0f), glm::vec3(0.0f, 1.0f, 0.0f));
            model = glm::scale(model, glm::vec3(0.05f, 0.05f, 0.05f));
            glUniformMatrix4fv(glGetUniformLocation(shader.Program, "model"), 1,
GL_FALSE, glm::value_ptr(model));
            Llantasdelanteras2.Draw(lightingShader);


            //Llatastraseras2
            view = camera.GetViewMatrix();
            model = glm::mat4(1);
            model = glm::translate(model, glm::vec3(movKitX2, -7.0f, movKitZ2));
            model = glm::rotate(model, glm::radians(180.0f), glm::vec3(0.0f, 1.0f, 0.0f));
            model = glm::scale(model, glm::vec3(0.05f, 0.05f, 0.05f));
            glUniformMatrix4fv(glGetUniformLocation(shader.Program, "model"), 1,
GL_FALSE, glm::value_ptr(model));
            Llantastraseras2.Draw(lightingShader);


            //Marco puerta
            view = camera.GetViewMatrix();
            model = glm::mat4(1);
            model = glm::translate(model, glm::vec3(2.0f, -7.0f, 4.0f));
            model = glm::rotate(model, glm::radians(180.0f), glm::vec3(0.0f, 1.0f, 0.0f));
```

```
		model = glm::scale(model, glm::vec3(0.05f, 0.05f, 0.05f));
		glUniformMatrix4fv(glGetUniformLocation(shader.Program, "model"), 1, GL_FALSE, glm::value_ptr(model));
		Marcopuerta.Draw(lightingShader);


		//Puerta
		view = camera.GetViewMatrix();
		model = glm::mat4(1);
		model = glm::translate(model, glm::vec3(2.0f, -7.0f, 4.0f));
		model = glm::rotate(model, glm::radians(180.0f), glm::vec3(0.0f, 1.0f, 0.0f));
		model = glm::scale(model, glm::vec3(0.05f, 0.05f, 0.05f));
		glUniformMatrix4fv(glGetUniformLocation(shader.Program, "model"), 1, GL_FALSE, glm::value_ptr(model));
		Puerta.Draw(lightingShader);


		//Sillon
		view = camera.GetViewMatrix();
		model = glm::mat4(1);
		model = glm::translate(model, glm::vec3(2.0f, -7.0f, 4.0f));
		model = glm::rotate(model, glm::radians(180.0f), glm::vec3(0.0f, 1.0f, 0.0f));
		model = glm::scale(model, glm::vec3(0.05f, 0.05f, 0.05f));
		glUniformMatrix4fv(glGetUniformLocation(shader.Program, "model"), 1, GL_FALSE, glm::value_ptr(model));
		Sillon.Draw(lightingShader);


		//Maceta
		view = camera.GetViewMatrix();
		model = glm::mat4(1);
		model = glm::translate(model, glm::vec3(2.0f, -7.0f, 4.0f));
		model = glm::rotate(model, glm::radians(180.0f), glm::vec3(0.0f, 1.0f, 0.0f));
		model = glm::scale(model, glm::vec3(0.05f, 0.05f, 0.05f));
		glUniformMatrix4fv(glGetUniformLocation(shader.Program, "model"), 1, GL_FALSE, glm::value_ptr(model));
		Maceta.Draw(lightingShader);
		//Comedor
		view = camera.GetViewMatrix();
		model = glm::mat4(1);
		model = glm::translate(model, glm::vec3(2.0f, -7.0f, 4.0f));
		model = glm::rotate(model, glm::radians(180.0f), glm::vec3(0.0f, 1.0f, 0.0f));
		model = glm::scale(model, glm::vec3(0.05f, 0.05f, 0.05f));
		glUniformMatrix4fv(glGetUniformLocation(shader.Program, "model"), 1, GL_FALSE, glm::value_ptr(model));
		Comedor.Draw(lightingShader);


		glBindVertexArray(0);



		// 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);



        // 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);
        GLuint cubemapTexture = TextureLoading::LoadCubemap(mysky[numsky]);
        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;
}

void animacionPuerta()
{
    if (movimientoPuerta)
    {
        if (movP1)
        {
            rotpuerta += 1.0f;
            if (rotpuerta > 295.0f)
            {
                movP1 = false;
                movP2 = true;
            }
        }

        if (movP2)
        {
            rotpuerta -= 1.0f;
            if (rotpuerta < 180.0f)
            {
                movP2 = false;
                movP3 = true;
            }
        }

        if (movP3)
        {
            rotpuerta = 180.0f;
            if (rotpuerta = 180.0f)
            {
                movP3 = false;
                movP1 = true;
            }
        }
    }
}

void animacion()
{
    if (movimientoTv)
    {

        if (movimiento1)
        {
            TvZ += 0.1;
            if (TvZ > 6.5)
            {
                movimiento1 = false;
                movimiento2 = true;
            }
        }

        if (movimiento2)
        {
            TvZ -= 0.1;
            if (TvZ < 4.0)
            {
                movimiento2 = false;
                movimiento3 = true;
            }
        }

        if (movimiento3)
        {
            TvZ = 4.0;
            if (TvZ = 4.0)
            {
                movimiento3 = false;
                movimiento1 = true;
            }
        }

    }

    if (circuito)
    {
        if (recorrido1)
        {
            movKitZ += 0.5f;
            if (movKitZ > 10)
            {
                recorrido1 = false;
                recorrido2 = true;
        }

}

if (recorrido2)
{
        //rotKit = 45;
        movKitZ -= 0.5f;
        movKitX -= 0.5f;
        if (movKitX < -10)
        {
                recorrido2 = false;
                recorrido3 = true;
        }
}

if (recorrido3)
{
        rotKit = 0.0;
        movKitZ -= 5.0;

        if (movKitZ < -150)
        {
                recorrido3 = false;
                recorrido4 = true;
        }
}

if (recorrido4)
{
        //rotKit = 90;
        movKitX -= 5.0;
```

```
    if (movKitX < -165)
    {
        recorrido4 = false;
        recorrido5 = true;
    }
}

if (recorrido5)
{
    //rotKit = 180;
    movKitZ += 5.0;

    if (movKitZ > 9)
    {
        recorrido5 = false;
        recorrido6 = true;
    }
}

if (recorrido6)
{
    //rotKit = 270;
    movKitX += 5.0;

    if (movKitX > 2)
    {
        recorrido6 = false;
        recorrido7 = true;
    }
}

if (recorrido7)
{
        rotKit = 0.0;
        movKitZ -= 0.2;

        if (movKitZ < 7)
        {
            recorrido7 = false;
            recorrido1 = true;
        }
    }
}

if (circuito2)
{
    if (recorridoA)
    {
        movKitZ2 -= 5.0f;
        if (movKitZ2 < -130 )
        {
            recorridoA = false;
            recorridoB = true;
        }

    }

    if (recorridoB)
    {
        //rotKit2 = 90;
        movKitX2 -= 5.0f;
        if (movKitX2 < -175)
        {
            recorridoB = false;
            recorridoC = true;
        }
```

```
}

if (recorridoC)
{
        //rotKit2 = 180;
        movKitZ2 += 5.0;

        if (movKitZ2 > 30)
        {
                recorridoC = false;
                recorridoD = true;
        }
}

if (recorridoD)
{
        //rotKit2 = 270;
        movKitX2 += 5.0;

        if (movKitX2 > -18)
        {
                recorridoD = false;
                recorridoE = true;
        }
}

if (recorridoE)
{
        //rotKit2 = 0.0;
        movKitZ2 -= 0.5;



        if (movKitZ2 < 0)
```

```
}
            {
                recorridoE = false;
                recorridoF = true;
            }
        }

        if (recorridoF)
        {
            //rotKit2 = -45.0;
            movKitZ2 -= 0.5;
            movKitX2 += 0.5;

            if (movKitX2 > 1)
            {
                recorridoF = false;
                recorridoG = true;
            }
        }

        if (recorridoG)
        {
            //rotKit2 = 0.0;
            movKitZ2 += 1.0;

            if (movKitZ2 > 4)
            {
                recorridoG = false;
                recorridoA = true;
            }
        }
    }
}
```

```
// 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;
                }
        }

        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);
        }

        if (keys[GLFW_KEY_Q])
        {
```

```
        sky = !sky;
        if (sky)
        {
            numsky = 1;


        }
        else
        {
            numsky = 0;
        }

        iluminacion = !iluminacion;
        if (iluminacion)
        {
            iluminacion = 1;


        }
        else
        {
            iluminacion = 0;
        }
    }
}

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/alters the camera positions based on user input
void DoMovement()
{
        if (keys[GLFW_KEY_Y])
        {

                movimientoTv = true;

        }

        if (keys[GLFW_KEY_H])
        {
                movimientoTv = false;


        }
        if (keys[GLFW_KEY_I])
        {

                circuito = true;
```

```
}

if (keys[GLFW_KEY_K])
{
	circuito = false;



}

if (keys[GLFW_KEY_U])
{


	circuito2 = true;


}

if (keys[GLFW_KEY_J])
{
	circuito2 = false;



}

// 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);
    }

}
```

Anexo.

Repositorio de GitHub: https://github.com/CesarCruz001/Lab-CGEIH