UNIVERSIDAD NACIONAL
AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Laboratorio Computación Gráfica e Interacción Humano Computadora

Manual Técnico

Profesor: Ing. Carlos Aldair Román Balbuena

Alumno: Corona Carrillo Emmanuel

Grupo 04

Contenido

Objetivos

Aplicar los conocimientos adquiridos a lo largo del laboratorio de Computación Gráfica e Interacción Humano Máquina, mediante el desarrollo de un proyecto de modelado de un entorno propuesto.

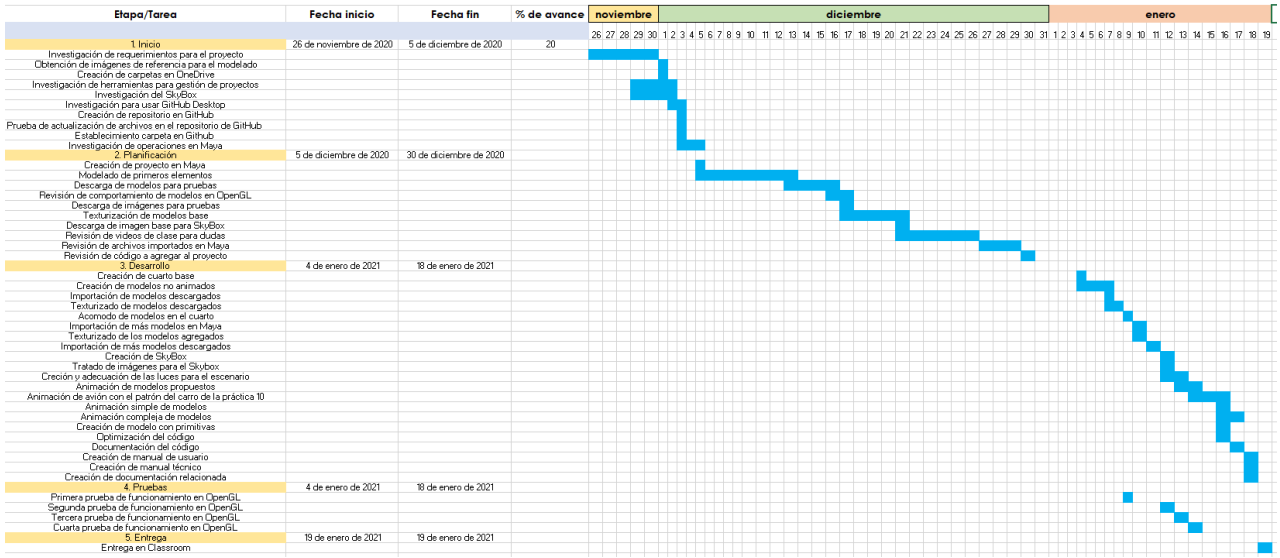
Se usarán herramientas de modelado de software como Maya para generar objetos en tres dimensiones con sus respectivas texturas. Además de apoyarse del código proporcionado durante el Laboratorio para generar dicho entorno en Visual Studio, implementado en OpenGL.

Alcance del proyecto

El proyecto servirá como introducción al ambiente de la animación y el modelado de objetos. Así mismo se pretende retomar conocimientos y metodologías de materias pasadas que son fundamentales al ejercer la profesión, ya que la buena gestión del proyecto evitará gastos extra y evitará salirse del presupuesto del proyecto.

Dichas metodologías son acompañadas de buenas prácticas de programación como lo son la documentación del código, el cual es otro punto esencial para la formación profesional como ingeniero, debido a que permite llevar un buen registro de cada parte del proyecto. Evitando retrasos innecesarios, ya sea por dudas en las funciones o al no entender alguna parte del código.

Diagrama de Gantt



Documentación del código

```
#include <iostream>
#include <cmath>
#include <string>
// 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(0.0f, 10.0f, -10.0f)); //Coordenadas de la cámara
GLfloat lastX = WIDTH / 2.0;//Variable para mantener centrada la ventana
GLfloat lastY = HEIGHT / 2.0;//Variable para mantener centrada la ventana
bool keys[1024];

bool firstMouse = true;

// Light attributes
glm::vec3 lightPos(0.0f, 5.0f, 0.0f);
glm::vec3 PosIni(16.0f, 11.0f, -18.0f); //Posicion inicial del carro
bool active;

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

//Variables globales
float posX = PosIni.x, posY = PosIni.y, posZ = PosIni.z;
float puerta_rot = 0;//Variable de rotación para la puerta
float ventana_rot = 0;//Variable de rotación para la ventana
float cajon_mov = 0;//Variable de movimiento para el cajón
float rel_rot = 0.0;//Variable de rotación para el reloj
```

```
int i_max_steps = 190;
int i_curr_steps = 0;

// Positions of the point lights
glm::vec3 pointLightPositions[] = {
	glm::vec3(posX,posY,posZ),//0
	glm::vec3(-8.1f,5.25f,-8.0f),//Lampara de escritorio
	glm::vec3(16.55f, 7.0f, 1.85f),//Lampara de noche
	glm::vec3(0.95f,5.17f,-19.0f)// Lampara de tocador
};

glm::vec3 LightP1;
//Animación del coche
float movKitX = 0.0;//Movimiento del modelo en el eje X
float movKitZ = 0.0;//Movimiento del modelo en el eje Z
float rotKit = 0.0;//Rotación del modelo en cierto eje
int i = 0; //Variable para iterar en un ciclo FOR

//Variables para el circuito del carro animado
bool circuito = false;
bool recorrido1 = true;
bool recorrido2 = false;
bool recorrido3 = false;
bool recorrido4 = false;
bool recorrido5 = false;

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);
	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);
	//Carga de Shaders
	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");


	//Se definen todos los objetos para su manipulación
	Model proyectFinal((char*)"Models/proyectFinal.obj");//Cuarto
	Model puerta((char*)"Models/puerta.obj");//Puerta
	Model ventana((char*)"Models/ventana.obj");//Ventana
	Model avion((char*)"Models/avion.obj");//Avion
	Model cajon_Esc((char*)"Models/deskDrawer.obj");//Cajon animado para el escritorio
```

```
	Model reloj((char*)"Models/clock.obj");//Reloj animado para la
comoda
	//Carga de modelos para el carro animado
	Model Carroseria((char*)"Models/Carro/Carroseria.obj");
	Model LLanta((char*)"Models/Carro/Wheel.obj");
	Model Piso((char*)"Models/Carro/Piso.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
        -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
};

// 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);

	// Carga de texturas del Skybox The Simpsons
	vector<const GLchar*> faces;
	faces.push_back("SkyBox/right.tga");
	faces.push_back("SkyBox/left.tga");
	faces.push_back("SkyBox/top.tga");
	faces.push_back("SkyBox/bottom.tga");
	faces.push_back("SkyBox/back.tga");
	faces.push_back("SkyBox/front.tga");

	GLuint cubemapTexture = TextureLoading::LoadCubemap(faces);

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

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

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

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

```
            // 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);
            glUniform3f(glGetUniformLocation(lightingShader.Program,
"dirLight.ambient"), 0.5f, 0.5f, 0.5f);
            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"), 0.5f, 0.5f, 0.5f);
            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
            //Carga de modelo
            //Modelo del cuarto base
            view = camera.GetViewMatrix();
            glm::mat4 model(1);
            model = glm::translate(model, glm::vec3(1.0, 1.0, 1.0));
            model = glm::scale(model, glm::vec3(2.0f, 2.0f, 2.0f));
            model = glm::rotate(model, glm::radians(180.0f),
glm::vec3(0.0f, 1.0f, 0.0f));
            glUniformMatrix4fv(modelLoc, 1, GL_FALSE,
glm::value_ptr(model));
            proyectFinal.Draw(lightingShader);
            //Modelo de la puerta
            model = glm::translate(tmp, glm::vec3(11.5f, 5.0f, -20.4f));
            model = glm::scale(model, glm::vec3(2.0f, 2.0f, 2.0f));
            //Se suma la variable puerta_rot para darle movimiento a la
puerta
            model = glm::rotate(model, glm::radians(puerta_rot+180.0f),
glm::vec3(0.0f, 1.0f, 0.0));
            glUniformMatrix4fv(modelLoc, 1, GL_FALSE,
glm::value_ptr(model));
            puerta.Draw(lightingShader);
```

```
            //Modelo de la ventana
            model = glm::translate(tmp, glm::vec3(11.3f, 9.15f, 7.7f));
            model = glm::scale(model, glm::vec3(2.0f, 2.0f, 2.0f));
            //Se suma la variable ventana_rot para darle movimiento a la
ventana
            model = glm::rotate(model, glm::radians(180.0f+ ventana_rot),
glm::vec3(1.0f, 0.0f, 0.0f));
            glUniformMatrix4fv(modelLoc, 1, GL_FALSE,
glm::value_ptr(model));
            ventana.Draw(lightingShader);
            //Modelo del avión
            model = glm::translate(tmp,glm::vec3(16.5f, 9.2f, -12.0f));
            model = glm::scale(model, glm::vec3(0.3f, 0.3f, 0.3f));
            model = glm::rotate(model, glm::radians(270.0f),
glm::vec3(0.0f, 1.0f, 0.0));
            glUniformMatrix4fv(modelLoc, 1, GL_FALSE,
glm::value_ptr(model));
            avion.Draw(lightingShader);
            //Modelo cajón del escritorio con movimiento
            //Se suma la variable cajon_mov para darle movimiento al
cajon
            model = glm::translate(tmp, glm::vec3(-8.54f + cajon_mov,
1.075f, -5.25f));
            model = glm::scale(model, glm::vec3(2.0f, 2.0f, 2.0f));
            model = glm::rotate(model, glm::radians(90.0f),
glm::vec3(0.0f, 1.0f, 0.0));
            glUniformMatrix4fv(modelLoc, 1, GL_FALSE,
glm::value_ptr(model));
            cajon_Esc.Draw(lightingShader);
            //Modelo del reloj
            model = glm::translate(tmp, glm::vec3(16.5f, 4.8f, -15.3f));
            model = glm::scale(model, glm::vec3(0.3f, 0.3f, 0.3f));
            //Se suma la variable rel_rot para darle movimiento al reloj
```

```
            model = glm::rotate(model, glm::radians(45.0f+ rel_rot),
glm::vec3(0.0f, 1.0f, 0.0));

            glUniformMatrix4fv(modelLoc, 1, GL_FALSE,
glm::value_ptr(model));

            reloj.Draw(lightingShader);

/////////////CARGA DE MODELOS PARA EL CARRO MOVIL/////////////////

            //Carga de modelo

            //Carroceria

            view = camera.GetViewMatrix();

            model = glm::mat4(1);

            //Las variables movKitX y movKitZ se suman para el movimiento
del carro animado

            model = glm::translate(model, PosIni + glm::vec3(movKitX, 0,
movKitZ));

            //La variable rotKit sse usa para darle rotación al carro y a
sus partes

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

            model = glm::scale(model, glm::vec3(0.008f, 0.008f, 0.008f));

            glUniformMatrix4fv(modelLoc, 1, GL_FALSE,
glm::value_ptr(model));

            Carroseria.Draw(lightingShader);


            //Llanta Delantera Der

            view = camera.GetViewMatrix();

            model = glm::mat4(1);

            //Las variables movKitX y movKitZ se suman para el movimiento
del carro animado

            model = glm::translate(model, PosIni + glm::vec3(movKitX-
1.1f, 0-0.35f, movKitZ-1.55f));

            //La variable rotKit sse usa para darle rotación al carro y a
sus partes

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

```
            model = glm::translate(model, glm::vec3(1.7f, 0.5f, 2.6f));;
            model = glm::scale(model, glm::vec3(0.008f, 0.008f, 0.008f));
            glUniformMatrix4fv(modelLoc, 1, GL_FALSE,
glm::value_ptr(model));
            LLanta.Draw(lightingShader);


            //Llanta Trasera Der
            view = camera.GetViewMatrix();
            model = glm::mat4(1);
            //Las variables movKitX y movKitZ se suman para el movimiento
del carro animado
            model = glm::translate(model, PosIni + glm::vec3(movKitX-
1.1f, 0-0.25f, movKitZ+1.75f));
            //La variable rotKit sse usa para darle rotación al carro y a
sus partes
            model = glm::rotate(model, glm::radians(rotKit),
glm::vec3(0.0f, 1.0f, 0.0));
            model = glm::translate(model, glm::vec3(1.7f, 0.5f, -2.9f));
            model = glm::scale(model, glm::vec3(0.008f, 0.008f, 0.008f));
            glUniformMatrix4fv(modelLoc, 1, GL_FALSE,
glm::value_ptr(model));
            LLanta.Draw(lightingShader);



            //Llanta Delantera Izq
            view = camera.GetViewMatrix();
            model = glm::mat4(1);
            //Las variables movKitX y movKitZ se suman para el movimiento
del carro animado
            model = glm::translate(model, PosIni +
glm::vec3(movKitX+1.1f, 0-0.5f, movKitZ-1.55f));
            //La variable rotKit sse usa para darle rotación al carro y a
sus partes
```

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

            model = glm::translate(model, glm::vec3(-1.7f, 0.8f, 2.6f));

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

            model = glm::scale(model, glm::vec3(0.008f, 0.008f, 0.008f));

            glUniformMatrix4fv(modelLoc, 1, GL_FALSE,
glm::value_ptr(model));

            LLanta.Draw(lightingShader);


            //Llanta Trasera Izq

            view = camera.GetViewMatrix();

            model = glm::mat4(1);

            //Las variables movKitX y movKitZ se suman para el movimiento
del carro animado

            model = glm::translate(model, PosIni +
glm::vec3(movKitX+1.1f, 0-0.4f, movKitZ+1.7f));

            //La variable rotKit sse usa para darle rotación al carro y a
sus partes

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

            model = glm::translate(model, glm::vec3(-1.7f, 0.8f, -2.9f));

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

            model = glm::scale(model, glm::vec3(0.008f, 0.008f, 0.008f));

            glUniformMatrix4fv(modelLoc, 1, GL_FALSE,
glm::value_ptr(model));

            LLanta.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);
            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;
}



// Is called whenever a key is pressed/released via GLFW
void KeyCallback(GLFWwindow* window, int key, int scancode, int action, int mode)
{	//Accion para encender luces
	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;
		}
	}
	//Tecla para abrir puerta
```

```
if (keys[GLFW_KEY_Z])
{
	for (i = 0; i < 90; i++)
	{
		puerta_rot +=1.0;
		if (puerta_rot > 90)
		{
			break;
		}
	}

}
//Tecla para cerrar puerta
if (keys[GLFW_KEY_X])
{
	for (i = 0; i < 90; i++)
	{
		puerta_rot -= 1.0;

	}
}

//Tecla para abrir ventana
if (keys[GLFW_KEY_C])
{
	for (i = 0; i < 90; i++)
	{
		ventana_rot += 1.0;
		if (ventana_rot > 90)
		{
```

```
                  break;
            }
      }

}
//Tecla para cerrar ventana
if (keys[GLFW_KEY_V])
{
      for (i = 0; i < 90; i++)
      {
            ventana_rot -= 1.0;

      }
}
//Tecla para rotar reloj
if (keys[GLFW_KEY_O])
{
      for (i = 0; i < 45; i++)
      {
            rel_rot += 1.0;
            if (rel_rot > 90)
            {
                  break;
            }
      }

}
//Tecla para rotar reloj
if (keys[GLFW_KEY_P])
{
```

```
	for (i = 0; i < 45; i++)
	{
		rel_rot -= 1.0;

	}

}

//Tecla para abrir el cajon
if (keys[GLFW_KEY_1])
{
	for (i=0;i<4;i++)
	{

		cajon_mov +=0.5 ;
	}



}
//Tecla para cerrar el cajon
if (keys[GLFW_KEY_2])
{
	for (i = 0; i < 4; i++)
	{

		cajon_mov -= 0.5;
	}



}
```

```
	//Animacion importante para la radio y las luces

	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_3])
	{
		active = !active;
		if (active)
			LightP1 = glm::vec3(1.0f, 0.0f, 0.0f);
		else
			LightP1 = glm::vec3(0.0f, 0.0f, 0.0f);
	}

}

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

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

```
	}


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

	lastX = xPos;
	lastY = yPos;

	camera.ProcessMouseMovement(xOffset, yOffset);
}


// Moves/alters the camera positions based on user input
void DoMovement()
{
	//Movimiento carro
	if (keys[GLFW_KEY_N])
	{
		circuito = true;


	}
	if (keys[GLFW_KEY_M])
	{
		circuito = 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);
	}
}


void animacion()
{
	//Movimiento del coche
	if (circuito)
	{
		if (recorrido1)
		{
			movKitX += 0.1f;
			rotKit = 90;
			if (movKitX > 90)
			{
```

```
            recorrido1 = false;
            recorrido4 = true;
      }
}
if (recorrido2)
{
      rotKit = 180;
      movKitZ -= 0.1f;
      if (movKitZ < 0)
      {
            recorrido2 = false;
            recorrido5 = true;

      }
}

if (recorrido4)
{
      rotKit = -45;
      movKitX -= 0.1f;
      movKitZ += 0.1f;

      if (movKitZ > 90)
      {
            recorrido4 = false;
            recorrido2 = true;
      }
}
if (recorrido5)
{
```

```
                rotKit = 90;
                movKitZ -= 0.1f;
                if (movKitZ < 0)
                {
                      recorrido5 = false;
                      recorrido1 = true;
                }
          }


      }
}
```