Universidad Nacional Autónoma de México

Facultad de Ingeniería

Ingeniería en Computación

Asignatura: Laboratorio de Computación Gráfica e Interacción Humano Computadora

Grupo: 8

Proyecto Final: Espacio recreado en open GL

Profesor: Ing. Carlos Aldair Román Balbuena

Nombre del alumno: Jiménez Gutiérrez Miguel

CONTENIDO

Objetivos

Aplicar los conocimientos adquiridos en la materia de Computación grafica e interacción humano computadora. Se realizará un proyecto que incluya todos los conceptos de las clases de laboratorio, así como de teoría, todo esto mediante el uso del lenguaje c++, OpenGL, visual studio y algún software de modelado como los son Maya, 3Dmax o Blender, el proyecto consistirá en un ambiente gráfico en que se recreara una fachada principal, así como el modelado de un cuarto y el uso de animaciones sencilla y complejas.

Diagrama de Gantt

Actividad	Semana 1	Semana 2	Semana 3	Semana 4	Semana 5
Análisis de requerimientos de proyecto					
Búsqueda de mágenes para propuesta					
Búsqueda de animaciones para propuesta					
Búsqueda de herramientas para proyecto					
Redacción de propuesta para proyecto					
Creación de Gantt de actividades					
Creación de repositorio en GitHub					
Creación de Tablero Kanban en Trello					
Listado de mobiliario para el interior					
Búsqueda de mobiliario					
Modelado interior					
Listado de elementos del exterior					
Búsqueda de elementos exteriores					
Modelado exterior					
Análisis de animaciones					
Creación de animaciones					
Adaptación de animaciones					
Listado de texturas a utilizar					
Búsqueda de texturas					
Adaptación de texturas					
Adaptación de cámara sintética					
Pruebas y mantenimiento finales					
Recopilación de elementos para documentación					
Creación del manual de usuario					
Ajustes finales					
Entrega del proyecto					

Alcance del proyecto

El presente proyecto tiene aplicaciones principalmente del área de cómputo gráfico, el alcance es limitado por el momento, debido a que son conceptos básicos, aunque es un impulsor a la creatividad y la futura mejora de este.

Como proyecto es una actividad que sin duda es una muestra de lo que el mundo laboral se pide. Y personalmente es un avance en cuanto experiencia se refiere.

Como alcance a futuro, si decido invertirle más tiempo, e incluso recursos monetarios, el proyecto puede mejorar estética y técnicamente.

Documentación del código

En el código existen diferentes variables que se emplearon y que son importantes de mencionar, no se mencionarán todas, únicamente las más empleadas

Variable	Descripción
active	Variable booleana que activa una serie de luces
activanim	Variable booleana que activa una animación mediante una entrada del teclado
activanim2	Variable booleana que activa una animación mediante una entrada del teclado
flag	Variable flotante que recibe un valor de acuerdo con otra variable de nombre rot, y evalúan condición en un ciclo if
flag2	Variable flotante auxiliar de la anterior variable
anim	Variable booleana que activa una animación mediante una entrada del teclado
model nombre((char*)/ruta);	Función que castea y que recibe como parámetro una ruta, para cargar un modelo
(keys[GLFW_KEY_nombre)	Recibe una entrada del teclado de acuerdo al nombre la variable o tecla
glm::mat4 model(1);	Crea un matriz de modelo
model = glm::mat4(1);	“Setea” un matriz de modelo, hace una copia en términos simples
Nombre.Draw(lightingShader);	Dibuja un modelo, y recibe como parámetro el shader, puesto que le tiene que avisar
model = glm::translate(model, glm::vec3(-3.5, 0.0, 0.0));	Realiza una transformación básica, en este caso mueve o traslada el objeto
model = glm::rotate(model, glm::radians(90), glm::vec3(1.0f, 0.0f, 0.0f));	Realiza una transformación básica, en este caso rota el objeto, tiene como parámetro una rotación sobre x en 90°, contiene 3 respectivos valores para x,y,z
model = glm::scale(model, glm::vec3(-15.0, 0.0, 6.0));	Realiza una transformación básica, en este caso escala el objeto, contiene 3 respectivos valores para x,y,z

Se anexan de igual forma funciones para las animaciones por key frames, si en un futuro se decide aplicar alguna mejora, de igual forma existen funciones para la iluminación, todo esto se puede observar en el código adjunto

En el caso de los key frames se puede utilizar las siguientes instrucciones

1- Declarar en keyFrames : variable inicializada en 0

2- Declarar en struct_frame: un variable / un incremento

3-Declarar en Inicialización de keyframes

4-Funcion animación en Draw animation

5-Funcion void Interpolation

6-Funcion void resetElements

7-Funcion saveframe

8-En la zona de modelos, asignar la variable donde se animará debajo de rotate

9-En la función doMovement : Asignar teclas a la animación

```
#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(-3.5f, 4.5f, 17.5f));//-4.0f, 4.0f, 10.0f
GLfloat lastX = WIDTH / 2.0;
```

```
GLfloat lastY = HEIGHT / 2.0;
bool keys[1024];
bool firstMouse = true;
float range = 0.0f;
float rot = 0.0f;
float rot2 = 0.0f;

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

bool activanim;
bool activanim2;

float flag = 0.0f;
float flag2 = 0.0f;

//Moto animacion
bool anim = false;
float posicion = 0.0;
float rotM = 0.0f;
bool recorrido1 = true;
bool recorrido2 = false;
bool recorrido3 = false;
bool recorrido4 = false;
bool recorrido5 = false;
float movx = 0.0f;
```

```
float movz = 0.0f;
float rotvehiculo = 0.0;
float vol;
float cuerpo;
float llantad;
float llantat;


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

// Keyframes
float posX =PosIni.x, posY = PosIni.y, posZ = PosIni.z, rotRodIzq = 0, rotRodDer=0;//=0
float rotBraIzq=0;
float rotBraDer=0;

#define MAX_FRAMES 9
int i_max_steps = 190;
int i_curr_steps = 0;
typedef struct _frame
{
	//Variables para GUARDAR Key Frames
	float posX;		//Variable para PosicionX
	float posY;		//Variable para PosicionY
	float posZ;		//Variable para PosicionZ
	float incX;		//Variable para IncrementoX
	float incY;		//Variable para IncrementoY
```

```
	float incZ;		//Variable para IncrementoZ
	float rotRodIzq;
	float rotRodDer;
	float rotInc;
	float rotInc2;
	float rotBraIzq;
	float rotBraDer;
	float rotInc3;
	float rotInc4;

}FRAME;

FRAME KeyFrame[MAX_FRAMES];
int FrameIndex = 0;			//introducir datos
bool play = false;
int playIndex = 0;

// Positions of the point lights
glm::vec3 pointLightPositions[] = {
	glm::vec3(-9.5f,5.0f,7.0f),//
	glm::vec3(2.0f,5.0f,7.0f),
	glm::vec3(-11.3f,11.0f,8.0f),
	glm::vec3(4.4f,11.0f,8.0f)
};

glm::vec3 Light1 = glm::vec3(0);
glm::vec3 LightP1;
```

```
float vertices[] = {
        // positions
        -0.5f, -0.5f, -0.5f,
         0.5f, -0.5f, -0.5f,
         0.5f,  0.5f, -0.5f,
         0.5f,  0.5f, -0.5f,
        -0.5f,  0.5f, -0.5f,
        -0.5f, -0.5f, -0.5f,

        -0.5f, -0.5f,  0.5f,
         0.5f, -0.5f,  0.5f,
         0.5f,  0.5f,  0.5f,
         0.5f,  0.5f,  0.5f,
        -0.5f,  0.5f,  0.5f,
        -0.5f, -0.5f,  0.5f,

        -0.5f,  0.5f,  0.5f,
        -0.5f,  0.5f, -0.5f,
        -0.5f, -0.5f, -0.5f,
        -0.5f, -0.5f, -0.5f,
        -0.5f, -0.5f,  0.5f,
        -0.5f,  0.5f,  0.5f,

         0.5f,  0.5f,  0.5f,
         0.5f,  0.5f, -0.5f,
         0.5f, -0.5f, -0.5f,
         0.5f, -0.5f, -0.5f,
```

```
         0.5f, -0.5f,  0.5f,
         0.5f,  0.5f,  0.5f,

        -0.5f, -0.5f, -0.5f,
         0.5f, -0.5f, -0.5f,
         0.5f, -0.5f,  0.5f,
         0.5f, -0.5f,  0.5f,
        -0.5f, -0.5f,  0.5f,
        -0.5f, -0.5f, -0.5f,

        -0.5f,  0.5f, -0.5f,
         0.5f,  0.5f, -0.5f,
         0.5f,  0.5f,  0.5f,
         0.5f,  0.5f,  0.5f,
        -0.5f,  0.5f,  0.5f,
        -0.5f,  0.5f, -0.5f
};


void saveFrame(void)
{

        printf("frameindex %d\n", FrameIndex);

        KeyFrame[FrameIndex].posX = posX;
        KeyFrame[FrameIndex].posY = posY;
        KeyFrame[FrameIndex].posZ = posZ;
```

```
	KeyFrame[FrameIndex].rotRodIzq = rotRodIzq;
	KeyFrame[FrameIndex].rotRodDer= rotRodDer;

	KeyFrame[FrameIndex].rotBraIzq = rotBraIzq;
	KeyFrame[FrameIndex].rotBraDer = rotBraDer;

	FrameIndex++;
}

void resetElements(void)
{
	posX = KeyFrame[0].posX;
	posY = KeyFrame[0].posY;
	posZ = KeyFrame[0].posZ;

	rotRodIzq = KeyFrame[0].rotRodIzq;
	rotRodDer = KeyFrame[0].rotRodDer;
	rotBraIzq = KeyFrame[0].rotBraIzq;
	rotBraDer = KeyFrame[0].rotBraDer;
}

void interpolation(void)
{

	KeyFrame[playIndex].incX = (KeyFrame[playIndex + 1].posX - KeyFrame[playIndex].posX) / i_max_steps;
	KeyFrame[playIndex].incY = (KeyFrame[playIndex + 1].posY - KeyFrame[playIndex].posY) / i_max_steps;
```

```
	KeyFrame[playIndex].incZ = (KeyFrame[playIndex + 1].posZ - KeyFrame[playIndex].posZ) / i_max_steps;


	KeyFrame[playIndex].rotInc = (KeyFrame[playIndex + 1].rotRodIzq - KeyFrame[playIndex].rotRodIzq) / i_max_steps;
	KeyFrame[playIndex].rotInc2 = (KeyFrame[playIndex + 1].rotRodDer - KeyFrame[playIndex].rotRodDer) / i_max_steps;
	KeyFrame[playIndex].rotInc3 = (KeyFrame[playIndex + 1].rotBraIzq - KeyFrame[playIndex].rotBraIzq) / i_max_steps;
	KeyFrame[playIndex].rotInc4 = (KeyFrame[playIndex + 1].rotBraDer - KeyFrame[playIndex].rotBraDer) / i_max_steps;
}




int main()
{
	// Init GLFW
	glfwInit();
	// Set all the required options for GLFW
	/*(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 CGeHC", 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");



Model Piso((char*)"Models/Piso.obj");
Model Casa((char*)"Models/Canceleria/CASAW.obj");
Model Puertaf((char*)"Models/Canceleria/PF.obj");
Model Puertat((char*)"Models/Canceleria/PT.obj");
Model Ventana1((char*)"Models/Canceleria/VPF.obj");
Model Ventana2((char*)"Models/Canceleria/VPT.obj");
Model Ventana3((char*)"Models/Canceleria/VFIZQ.obj");
Model Ventana4((char*)"Models/Canceleria/VFDER.obj");
Model Ventana5((char*)"Models/Canceleria/VA.obj");
Model Ventana6((char*)"Models/Canceleria/VP.obj");
Model cerca((char*)"Models/fence/cerca.obj");
Model arbolp((char*)"Models/trees/arbol_p.obj");
```

```
Model arbolm((char*)"Models/trees/arbol_m.obj");
Model arbolg((char*)"Models/trees/arbol_g.obj");
Model moto((char*)"Models/bike/moto1.obj");
Model motolld((char*)"Models/bike/llanta_d.obj");
Model motollt((char*)"Models/bike/llanta_t.obj");
Model motoc((char*)"Models/bike/cuerpo.obj");
Model motov((char*)"Models/bike/volante.obj");

// Build and compile our shader program

//Inicialización de KeyFrames

for (int i = 0; i < MAX_FRAMES; i++)
{
	KeyFrame[i].posX = 0;
	KeyFrame[i].incX = 0;
	KeyFrame[i].incY = 0;
	KeyFrame[i].incZ = 0;
	KeyFrame[i].rotRodIzq = 0;
	KeyFrame[i].rotRodDer = 0;
	KeyFrame[i].rotInc = 0;
	KeyFrame[i].rotInc2 = 0;
	KeyFrame[i].rotBraIzq = 0;
	KeyFrame[i].rotBraDer = 0;
	KeyFrame[i].rotInc3 = 0;
	KeyFrame[i].rotInc4 = 0;

}
```

```
// 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);

// Load textures
vector<const GLchar*> faces;
faces.push_back("SkyBox3/right.tga");
faces.push_back("SkyBox3/left.tga");
faces.push_back("SkyBox3/top.tga");
faces.push_back("SkyBox3/bottom.tga");
faces.push_back("SkyBox3/back.tga");
faces.push_back("SkyBox3/front.tga");

GLuint cubemapTexture = TextureLoading::LoadCubemap(faces);

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

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

    // Calculate deltatime of current frame
```

```
GLfloat currentFrame = glfwGetTime();
deltaTime = currentFrame - lastFrame;
lastFrame = currentFrame;

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



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



// Use cooresponding shader when setting uniforms/drawing objects
lightingShader.Use();
GLint viewPosLoc = glGetUniformLocation(lightingShader.Program, "viewPos");
glUniform3f(viewPosLoc, camera.GetPosition().x, camera.GetPosition().y, camera.GetPosition().z);
// Set material properties
glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 32.0f);
// == ==========================
// Here we set all the uniforms for the 5/6 types of lights we have. We have to set them manually and index
// the proper PointLight struct in the array to set each uniform variable. 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"), 0.3f, 0.3f, 0.3f);//0.6f, 0.6f, 0.6f
        glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.diffuse"), 0.1f, 0.1f, 0.1f);
        glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.specular"), 0.0f, 0.0f, 0.0f);
        // 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"), 0.0f, 0.0f, 0.0f);
        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"), 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"), 0.0f, 0.0f, 0.0f);
        glUniform3f(glGetUniformLocation(lightingShader.Program,
"pointLights[1].diffuse"), 0.0f, 0.0f, 0.0f);
        glUniform3f(glGetUniformLocation(lightingShader.Program,
"pointLights[1].specular"), 1.0f, 1.0f, 1.0f);
        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, 0.0f, 0.0f);
        glUniform3f(glGetUniformLocation(lightingShader.Program,
"pointLights[2].specular"), 1.0f, 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"), 0.0f, 0.0f, 0.0f);

        glUniform3f(glGetUniformLocation(lightingShader.Program,
"pointLights[3].specular"), 1.0f, 1.0f, 1.0f);

        glUniform1f(glGetUniformLocation(lightingShader.Program,
"pointLights[3].constant"), 1.0f);

        glUniform1f(glGetUniformLocation(lightingShader.Program,
"pointLights[3].linear"), 0.09f);

        glUniform1f(glGetUniformLocation(lightingShader.Program,
"pointLights[3].quadratic"), 0.032f);



        // SpotLight

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

        glUniform3f(glGetUniformLocation(lightingShader.Program,
"spotLight.direction"), camera.GetFront().x, camera.GetFront().y,
camera.GetFront().z);

        glUniform3f(glGetUniformLocation(lightingShader.Program,
"spotLight.ambient"), 0.0f, 0.0f, 0.0f);

        glUniform3f(glGetUniformLocation(lightingShader.Program,
"spotLight.diffuse"), 0.0f, 0.0f, 0.0f);
```

```
        glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight.specular"), 0.0f, 0.0f, 0.0f);
        glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight.constant"), 1.0f);
        glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight.linear"), 0.09f);
        glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight.quadratic"), 0.032f);
        glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight.cutOff"), glm::cos(glm::radians(12.5f)));
        glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight.outerCutOff"), glm::cos(glm::radians(15.0f)));

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

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


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

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

```
glUniformMatrix4fv(projLoc, 1, GL_FALSE, glm::value_ptr(projection));

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

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


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

glm::mat4 model(1);

//Carga de modelo


//Piso
view = camera.GetViewMatrix();
model = glm::mat4(1);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Piso.Draw(lightingShader);


//Casa
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-3.5, 0.0, 0.0));
```

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



//Puerta Frontral
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-3.45, 5.6, 5.7));
model = glm::rotate(model, glm::radians(rot), glm::vec3(1.0f, 0.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
//glUniform1i(glGetUniformLocation(lightingShader.Program, "Transparencia"), 1.1);
Puertaf.Draw(lightingShader);

//Puerta trasera
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-3.38f, 5.4, -5.7));
model = glm::rotate(model, glm::radians(-rot2), glm::vec3(1.0f, 0.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Puertat.Draw(lightingShader);


//*****************************VENTANAS*********************************\\

//Ventana puerta frontal
glEnable(GL_BLEND);
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
model = glm::mat4(1);
//model = glm::scale(model, glm::vec3(-15.0, 0.0, 6.0));
```

```
		model = glm::translate(model, glm::vec3(-3.55, 5.4, 5.74));
		model = glm::rotate(model, glm::radians(rot), glm::vec3(1.0f, 0.0f, 0.0f));
		glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
		//glUniform1i(glGetUniformLocation(lightingShader.Program, "Transparencia"), 0.1);
		Ventana1.Draw(lightingShader);
		glDisable(GL_BLEND);
		glEnable(GL_DEPTH_TEST);

		//Ventana puerta trasera
		glEnable(GL_BLEND);
		glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
		model = glm::mat4(1);
		model = glm::translate(model, glm::vec3(-3.51f, 5.4, -5.7));
		model = glm::rotate(model, glm::radians(-rot2), glm::vec3(1.0f, 0.0f, 0.0f));
		glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
		//glUniform1i(glGetUniformLocation(lightingShader.Program, "Transparencia"), 1.1);
		Ventana2.Draw(lightingShader);
		glDisable(GL_BLEND);
		glEnable(GL_DEPTH_TEST);


		//Ventanas superiores izquierda y frontal
		glEnable(GL_BLEND);
		glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
		model = glm::mat4(1);
		model = glm::translate(model, glm::vec3(-3.52, -3.55, 0.01));
```

```
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Ventana3.Draw(lightingShader);
glDisable(GL_BLEND);
glEnable(GL_DEPTH_TEST);

//Ventana superior derecha
glEnable(GL_BLEND);
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-3.5, -3.4, 0.08));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Ventana4.Draw(lightingShader);
glDisable(GL_BLEND);
glEnable(GL_DEPTH_TEST);

//Ventana superior trasera
glEnable(GL_BLEND);
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-3.5, -3.5, 0.01));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Ventana5.Draw(lightingShader);
glDisable(GL_BLEND);
glEnable(GL_DEPTH_TEST);

//Ventana de la Pared
glEnable(GL_BLEND);
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
```

```
		model = glm::mat4(1);
		model = glm::translate(model, glm::vec3(0.56, 0.020, 0.030));
		glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
		Ventana6.Draw(lightingShader);
		glDisable(GL_BLEND);
		glEnable(GL_DEPTH_TEST);

		//*****************************TERMINA
VENTANAS*********************************\\



		model = glm::mat4(1);
		//model = glm::translate(model, glm::vec3(0.56, 0.020, 0.030));
		glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
		cerca.Draw(lightingShader);

		//*****************************Arboles*****************************\\
		model = glm::mat4(1);
		model = glm::translate(model, glm::vec3(-1.5, 0.0, 10.30));
		glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
		arbolp.Draw(lightingShader);

		model = glm::mat4(1);
		model = glm::translate(model, glm::vec3(18.0, 0.0, -15.30));
		glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
		arbolm.Draw(lightingShader);

		model = glm::mat4(1);
		model = glm::translate(model, glm::vec3(9.0, 0.0, 20.0));
```

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

//*****************************Termina Arboles*************************\\

model = glm::mat4(1);
model = glm::translate(model, glm::vec3(1.5, 1.5, 3.0));
model = glm::rotate(model, glm::radians(90.0f), glm::vec3(0.0, 1.0, 0.0));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
moto.Draw(lightingShader);

//moto en moviento

model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-18.0f + movx, 0.0f, 0.0f + movz));
model = glm::rotate(model, glm::radians(rotvehiculo), glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::translate(model, glm::vec3(0.0f, 0.97f, 21.0f)); // Translate it down a bit so it's at the center of the scene
//model = glm::rotate(model, glm::radians(rot), glm::vec3(1.0f, 0.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
motoc.Draw(lightingShader);

model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-18.0f + movx, 0.0f, 0.0f + movz));
model = glm::rotate(model, glm::radians(rotvehiculo), glm::vec3(0.0f, 1.0f, 0.0f));
```

```
model = glm::translate(model, glm::vec3(1.27f, 1.15f, 21.0f)); // Translate it down a bit so it's at the center of the scene

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

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

motov.Draw(lightingShader);


model = glm::mat4(1);

model = glm::translate(model, glm::vec3(-18.0f + movx, 0.0f, 0.0f + movz)); // Translate it down a bit so it's at the center of the scene

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

model = glm::translate(model, glm::vec3(1.47f, 0.5f, 21.0f)); // Translate it down a bit so it's at the center of the scene

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

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

motolld.Draw(lightingShader);


model = glm::mat4(1);

model = glm::translate(model, glm::vec3(-18.0f + movx, 0.0f, 0.0f + movz));

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

model = glm::translate(model, glm::vec3(-0.98f , 0.5f, 21.0f)); // Translate it down a bit so it's at the center of the scene

//model = glm::rotate(model, glm::radians(180.0f), glm::vec3(0.0f, 0.0f, 1.0f)); // model = glm::translate(model, glm::vec3(0.0f, 1.0f, -1.0f)); // Translate it down a bit so it's at the center of the scene

model = glm::rotate(model, glm::radians(-rotM), glm::vec3(0.0f, 0.0f, 1.0f)); // model = glm::translate(model, glm::vec3(0.0f, 1.0f, -1.0f)); // Translate it down a bit so it's at the center of the scene

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

```
        motollt.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;
}


void animacion()
{

            //Movimiento del personaje

            if (play)
```

```
		{
			if (i_curr_steps >= i_max_steps) //end of animation between frames?
			{
				playIndex++;
				if (playIndex>FrameIndex - 2)	//end of total animation?
				{
					printf("termina anim\n");
					playIndex = 0;
					play = false;
				}
				else //Next frame interpolations
				{
					i_curr_steps = 0; //Reset counter
											//Interpolation
					interpolation();
				}
			}
			else
			{
				//Draw animation
				posX += KeyFrame[playIndex].incX;
				posY += KeyFrame[playIndex].incY;
				posZ += KeyFrame[playIndex].incZ;

				rotRodIzq += KeyFrame[playIndex].rotInc;
				rotRodDer += KeyFrame[playIndex].rotInc2;
				rotBraIzq += KeyFrame[playIndex].rotInc3;
```

```
                rotBraDer += KeyFrame[playIndex].rotInc4;

                i_curr_steps++;
            }

        }
    }


// Is called whenever a key is pressed/released via GLFW
void KeyCallback(GLFWwindow *window, int key, int scancode, int action, int mode)
{
    if (keys[GLFW_KEY_L])
    {
        if (play == false && (FrameIndex > 1))
        {

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

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

```
        }

}

if (keys[GLFW_KEY_K])
{
        if (FrameIndex<MAX_FRAMES)
        {
                saveFrame();
        }

}


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(0.0f, 1.0f, 0.0f);
        else
                LightP1 = glm::vec3(0.0f, 0.0f, 0.0f);

}

if (keys[GLFW_KEY_I])
{
        activanim = !activanim;

}

if (keys[GLFW_KEY_O])
{
        activanim2 = !activanim2;

}

if (keys[GLFW_KEY_P])
{
        anim = !anim;
```

```
        }

}

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_1])
{

	rot += 1;

}

if (keys[GLFW_KEY_2])
{
	if (rotRodIzq < 80.0f)
		rotRodIzq += 1.0f;

}

if (keys[GLFW_KEY_3])
{
	if (rotRodIzq > -45)
		rotRodIzq -= 1.0f;

}


if (keys[GLFW_KEY_4])
{
	if (rotRodDer < 80.0f)
		rotRodDer += 1.0f;

}
```

```
if (keys[GLFW_KEY_5])
{
	if (rotRodDer > -45)
		rotRodDer -= 1.0f;

}


if (keys[GLFW_KEY_6])
{
	if (rotBraDer < 80.0f)
		rotBraDer += 1.0f;

}

if (keys[GLFW_KEY_7])
{
	if (rotBraDer > -45)
		rotBraDer -= 1.0f;

}

if (keys[GLFW_KEY_8])
{
	if (rotBraIzq < 80.0f)
		rotBraIzq += 1.0f;
```

```
}

if (keys[GLFW_KEY_9])
{
	if (rotBraIzq > -45)
		rotBraIzq -= 1.0f;

}

//Mov Personaje
if (keys[GLFW_KEY_H])
{
	posZ += 1;
}

if (keys[GLFW_KEY_Y])
{
	posZ -= 1;
}

if (keys[GLFW_KEY_G])
{
	posX -= 1;
}

if (keys[GLFW_KEY_J])
{
	posX += 1;
```

```
}

//**********Animacion 1*****************\\

if (activanim)

        if (flag2 == 0 || flag2 > 0)//flag2==0

                if (rot >= -90)
                {
                        rot -= 0.1f;
                        flag = rot;

                }
if (flag <= -90)//flag <= 0 && flag <= -90
{
        if (rot <= 0)
                rot += 0.1f;
        flag2 = rot;
}

//**********TERMINA Animacion 1*****************\\

//**********Animacion 2*****************\\

if (activanim2)

        if (flag2 == 0 || flag2 > 0)//flag2==0
```

```
                if (rot2 >= -90)
                {
                        rot2 -= 0.1f;
                        flag = rot2;

                }
if (flag <= -90)//flag <= 0 && flag <= -90
{
        if (rot2 <= 0)
                rot2 += 0.1f;
        flag2 = rot2;
}

//********** TERMINA  Animacion 2 ******************\\


//*************Animacion 3 (la moto)*****************\\


if (anim)
{
        if (recorrido1)
        {
                if (movx > 36.0f)
                {
                        recorrido1 = false;
```

```
                }
                else
                {
                        rotM += 0.4f;
                        movx += 0.1f;
                }
        }

}


//******************TERMINA  Animacion 3(la moto)******************\\

// 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);
        }









}
```