316075189 GPO:09

MANUAL TECNICO

INTRODUCCION

La finalidad de este manual técnico es la de proporcionar al lector las pautas de configuración y la lógica con la que se ha desarrollado este proyecto, la cual se sabe que es propia de cada programador; por lo que se considera necesario ser documentada. Aclarando que este manual no pretende ser un curso de aprendizaje de cada una de las herramientas empleadas para el desarrollo. Para un mayor detalle acerca de cada una de las herramientas utilizadas, y su forma de operación y aplicación, se recomienda consultar los manuales respectivos de cada una de ellos.

OBJETIVO

El alumno deberá aplicar y demostrar los conocimientos adquiridos durante todo el curso.

Diagrama de Gantt

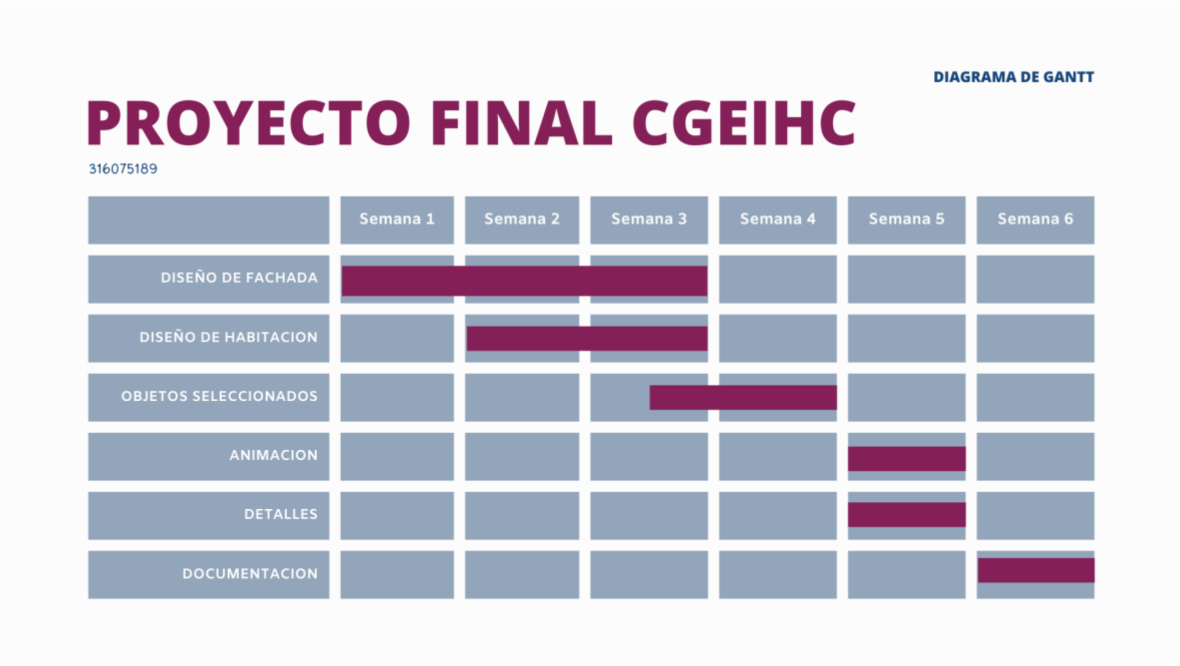

DIAGRAMA DE GANTT

PROYECTO FINAL CGEIHC

316075189

	Semana 1	Semana 2	Semana 3	Semana 4	Semana 5	Semana 6
DISEÑO DE FACHADA	■	■	■			
DISEÑO DE HABITACION		■	■			
OBJETOS SELECCIONADOS			■	■		
ANIMACION					■	
DETALLES					■	
DOCUMENTACION						■

Alcance del proyecto

Este busca el cáncer que se cubran todos los temas abarcados a lo largo de ir trimestre en el laboratorio, te gusta que puedo tener una aplicación más adelante con distintos conocimientos que adquirimos al terminar la carrera, esta experiencia también nos va a servir para poder tener material para posteriormente presentarnos ante una entrevista de trabajo.

Limitantes

Los limitaciones de este proyecto es que posiblemente las animaciones que hasta ahora conocemos son de maneras muy simples no podemos comparar las con películas o video juegos, los recursos que utilizamos también podemos decir que son limitados, pero si posteriormente nosotros buscamos trabajo en este tipo de áreas posiblemente los recursos ya no serán un limitante.

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"

// Function prototypes
void KeyCallback(GLFWwindow* window, int key, int scancode, int action, int mode);
void MouseCallback(GLFWwindow* window, double xPos, double yPos);
void DoMovement();

// Window dimensions
```

```
const GLuint WIDTH = 800, HEIGHT = 600;
int SCREEN_WIDTH, SCREEN_HEIGHT;

// Camera
Camera  camera(glm::vec3(0.0f, 5.0f, 20.0f));
GLfloat lastX = WIDTH / 2.0;
GLfloat lastY = HEIGHT / 2.0;
bool keys[1024];
bool firstMouse = true;
// Light attributes

float rot = 90.0f;
float rot2 = 0.0f;
bool mov1 = false;
bool mov2 = false;
glm::vec3 lightPos(0.0f, 0.0f, 0.0f);
bool active;
float tiempo;
glm::vec3 PosIni(-95.0f, 0.0f, -45.0f);

float posX;           //Variable para PosicionX
float posY;           //Variable para PosicionY
float posZ;           //Variable para PosicionZ
//Animación del coche
float movKitX = 0.0;
float movKitZ = 0.0;
float rotKit = 0.0;

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


// Positions of the point lights
```

```
glm::vec3 pointLightPositions[] = {
        glm::vec3(0.7f,  0.2f,  2.0f),
        glm::vec3(2.3f, -3.3f, -4.0f),
        glm::vec3(-4.0f,  2.0f, -12.0f),
        glm::vec3(0.0f,  0.0f, -3.0f)
};
glm::vec3 LightP1;

float vertices[] = {
        -0.5f, -0.5f, -0.5f,  0.0f,  0.0f, -1.0f,
               0.5f, -0.5f, -0.5f,  0.0f,  0.0f, -1.0f,
               0.5f,  0.5f, -0.5f,  0.0f,  0.0f, -1.0f,
               0.5f,  0.5f, -0.5f,  0.0f,  0.0f, -1.0f,
          -0.5f,  0.5f, -0.5f,  0.0f,  0.0f, -1.0f,
          -0.5f, -0.5f, -0.5f,  0.0f,  0.0f, -1.0f,

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

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

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

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



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



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

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

       // Create a GLFWwindow object that we can use for GLFW's functions
```

```
        GLFWwindow* window = glfwCreateWindow(WIDTH, HEIGHT, "316075189_PROYECTOFINAL_GPO09",
nullptr, nullptr);


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


                return EXIT_FAILURE;
        }


        glfwMakeContextCurrent(window);


        glfwGetFramebufferSize(window, &SCREEN_WIDTH, &SCREEN_HEIGHT);


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


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


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


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

```
Shader lightingShader("Shaders/lighting.vs", "Shaders/lighting.frag");
//Shader lampShader("Shaders/lamp.vs", "Shaders/lamp.frag");
Shader lampShader("Shaders/lamp2.vs", "Shaders/lamp2.frag");
Shader Anim("Shaders/anim.vs", "Shaders/anim.frag");
Shader Anim2("Shaders/anim2.vs", "Shaders/anim2.frag");
//Cuarto
Model Cuarto((char*)"Models/Cuarto/Cuarto.obj");
Model Puerta((char*)"Models/Cuarto/Puerta.obj");
Model Ducha((char*)"Models/Cuarto/Ducha.obj");
Model BarraL((char*)"Models/Cuarto/barraLateral.obj");
Model BarraT((char*)"Models/Cuarto/barraTrabajo.obj");
Model Carrito((char*)"Models/Cuarto/carrito.obj");
Model EstanSup((char*)"Models/Cuarto/estanteSuperior.obj");
Model MeCentral((char*)"Models/Cuarto/mesaCentral.obj");
Model Pizarron((char*)"Models/Cuarto/Pizarron.obj");
Model Vidrio((char*)"Models/Cuarto/Vidrio.obj");
Model Vidrio2((char*)"Models/Cuarto/Vidrio2.obj");
Model Pecera1((char*)"Models/Cuarto/Pecera1.obj");
Model Pecera2((char*)"Models/Cuarto/Pecera2.obj");
Model Ciencia((char*)"Models/Cuarto/Ciencia.obj");



//OBJETOS
Model Micro((char*)"Models/Micro/11642_Microwave_v1_L3.obj");
Model craneo((char*)"Models/skull/skull.obj");
Model Planta((char*)"Models/Planta/indoor plant_02.obj");
Model Material((char*)"Models/Cuarto/material.obj");
Model Libros((char*)"Models/Cuarto/Libros.obj");
Model Porta((char*)"Models/Cuarto/Porta.obj");
Model Tetera((char*)"Models/Cuarto/Tetera.obj");
Model Locker((char*)"Models/Cuarto/Locker.obj");

//FACHADA

Model Punta((char*)"Models/Castillo/Punta.obj");
Model Nubes((char*)"Models/Castillo/Nubes.obj");
Model Torres((char*)"Models/Castillo/Torres.obj");
```

```
        Model Arboles((char*)"Models/Castillo/Arbol_paleta.obj");
        Model Barda((char*)"Models/Castillo/Barda.obj");
        Model Entrada((char*)"Models/Castillo/Entrada.obj");
        Model Piso((char*)"Models/Castillo/Piso.obj");
        Model Rio((char*)"Models/Castillo/Rio.obj");
        Model Castillo((char*)"Models/Castillo/Castillo.obj");
        Model Vent_der((char*)"Models/Castillo/Ventanas_der.obj");
        Model Vent_izq((char*)"Models/Castillo/Ventanas_izq.obj");
        Model Porton_D((char*)"Models/Castillo/Porton_der.obj");
        Model Porton_I((char*)"Models/Castillo/Porton_izq.obj");
        Model BardaPa((char*)"Models/Castillo/BardaPa.obj");
        Model Arcoiris((char*)"Models/Castillo/Arcoiris.obj");
        Model Escaleras((char*)"Models/Castillo/Escaleras.obj");




        // First, set the container's VAO (and VBO)
        GLuint VBO, VAO;
        glGenVertexArrays(1, &VAO);
        glGenBuffers(1, &VBO);
        glBindVertexArray(VAO);
        glBindBuffer(GL_ARRAY_BUFFER, VBO);
        glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW);
        // Position attribute
        glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(GLfloat), (GLvoid*)0);
        glEnableVertexAttribArray(0);
        // normal attribute
        glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(float), (void*)(3 * sizeof(float)));
        glEnableVertexAttribArray(1);

        // Set texture units
        lightingShader.Use();
        glUniform1i(glGetUniformLocation(lightingShader.Program, "material.diffuse"), 0);
        glUniform1i(glGetUniformLocation(lightingShader.Program, "material.specular"), 1);
        glm::mat4  projection  =  glm::perspective(camera.GetZoom(),  (GLfloat)SCREEN_WIDTH  /
(GLfloat)SCREEN_HEIGHT, 0.1f, 100.0f);


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


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


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


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


                // OpenGL options
                glEnable(GL_DEPTH_TEST);




                //Load Model



                // 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);



                // 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"),
0.5f, 0.5f, 0.5f);

               glUniform3f(glGetUniformLocation(lightingShader.Program,
"dirLight.specular"), 0.00f, 0.00f, 0.0f);



               // Point light 1

               glm::vec3 lightColor;

               lightColor.x = abs(sin(glfwGetTime() * Light1.x));

               lightColor.y = abs(sin(glfwGetTime() * Light1.y));

               lightColor.z = sin(glfwGetTime() * Light1.z);



               glUniform3f(glGetUniformLocation(lightingShader.Program,
"pointLights[0].position"),       pointLightPositions[0].x,       pointLightPositions[0].y,
pointLightPositions[0].z);

               glUniform3f(glGetUniformLocation(lightingShader.Program,
"pointLights[0].ambient"), lightColor.x, lightColor.y, lightColor.z);

               glUniform3f(glGetUniformLocation(lightingShader.Program,
"pointLights[0].diffuse"), lightColor.x, lightColor.y, lightColor.z);

               glUniform3f(glGetUniformLocation(lightingShader.Program,
"pointLights[0].specular"), 0.0f, 0.0f, 0.0f);

               glUniform1f(glGetUniformLocation(lightingShader.Program,
"pointLights[0].constant"), 1.0f);

               glUniform1f(glGetUniformLocation(lightingShader.Program,
"pointLights[0].linear"), 0.8f);

               glUniform1f(glGetUniformLocation(lightingShader.Program,
"pointLights[0].quadratic"), 1.8f);



               // 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"), 0.0f, 0.0f, 0.0f);
```

```
               glUniform3f(glGetUniformLocation(lightingShader.Program,
"pointLights[1].specular"), 0.0f, 0.0f, 0.0f);

               glUniform1f(glGetUniformLocation(lightingShader.Program,
"pointLights[1].constant"), 1.0f);

               glUniform1f(glGetUniformLocation(lightingShader.Program,
"pointLights[1].linear"), 0.0f);

               glUniform1f(glGetUniformLocation(lightingShader.Program,
"pointLights[1].quadratic"), 0.0f);



               // 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"), 0.0f, 0.0f, 0.0f);

               glUniform3f(glGetUniformLocation(lightingShader.Program,
"pointLights[2].diffuse"), 0.0f, 0.0f, 0.0f);

               glUniform3f(glGetUniformLocation(lightingShader.Program,
"pointLights[2].specular"), 0.0f, 0.0f, 0.0f);

               glUniform1f(glGetUniformLocation(lightingShader.Program,
"pointLights[2].constant"), 1.0f);

               glUniform1f(glGetUniformLocation(lightingShader.Program,
"pointLights[2].linear"), 0.0f);

               glUniform1f(glGetUniformLocation(lightingShader.Program,
"pointLights[2].quadratic"), 0.0f);



               // 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"), 0.0f, 0.0f, 0.0f);

               glUniform3f(glGetUniformLocation(lightingShader.Program,
"pointLights[3].diffuse"), 0.0f, 0.0f, 0.0f);

               glUniform3f(glGetUniformLocation(lightingShader.Program,
"pointLights[3].specular"), 0.0f, 0.0f, 0.0f);

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

               glUniform1f(glGetUniformLocation(lightingShader.Program,
"pointLights[3].linear"), 0.0f);

               glUniform1f(glGetUniformLocation(lightingShader.Program,
"pointLights[3].quadratic"), 0.0f);



               // SpotLight

               glUniform3f(glGetUniformLocation(lightingShader.Program,
"spotLight.position"), 0.0f, 10.0f, 0.0f);
```

```
               glUniform3f(glGetUniformLocation(lightingShader.Program,
"spotLight.direction"), 0.0f, 10.0f, 0.0f);

               glUniform3f(glGetUniformLocation(lightingShader.Program,
"spotLight.ambient"), 1.0f, 1.0f, 1.0f);

               glUniform3f(glGetUniformLocation(lightingShader.Program,
"spotLight.diffuse"), 1.0f, 1.0f, 1.0f);

               glUniform3f(glGetUniformLocation(lightingShader.Program,
"spotLight.specular"), 0.5f, 0.5f, 0.5f);

               glUniform1f(glGetUniformLocation(lightingShader.Program,
"spotLight.constant"), 1.0f);

               glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight.linear"),
0.07f);

               glUniform1f(glGetUniformLocation(lightingShader.Program,
"spotLight.quadratic"), 0.17f);

               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"), 16.0f);


               // 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));



               glm::mat4 model(1);


               //Carga de modelo

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

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

               glUniform1i(glGetUniformLocation(lightingShader.Program,
"activaTransparencia"), 0);

               Piso.Draw(lightingShader);


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

               glUniform1i(glGetUniformLocation(lightingShader.Program,
"activaTransparencia"), 0);





               //CUARTO

               model = glm::mat4(1);

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

               Cuarto.Draw(lightingShader);





               //DUcha

               model = glm::mat4(1);

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

               Ducha.Draw(lightingShader);


               ////Vidrio

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

               glUniform1i(glGetUniformLocation(lightingShader.Program,
"activaTransparencia"), 0);

               glEnable(GL_BLEND);//Avtiva la funcionalidad para trabajar el canal alfa

               glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);

               model = glm::mat4(1);


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

               glUniform1i(glGetUniformLocation(lightingShader.Program,
"activaTransparencia"), 0);

               glUniform4f(glGetUniformLocation(lightingShader.Program, "colorAlpha"), 1.0f,
1.0f, 1.0f, 0.3f);

               glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
               glUniform1i(glGetUniformLocation(lightingShader.Program,
"activaTransparencia"), 0);

               Vidrio.Draw(lightingShader);

               glDisable(GL_BLEND);  //Desactiva el canal alfa

               glUniform4f(glGetUniformLocation(lightingShader.Program, "colorAlpha"), 1.0f,
1.0f, 1.0f, 1.0f);





               ////Vidrio 2

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

               glUniform1i(glGetUniformLocation(lightingShader.Program,
"activaTransparencia"), 0);

               glEnable(GL_BLEND);//Avtiva la funcionalidad para trabajar el canal alfa

               glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);

               model = glm::mat4(1);


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

               glUniform1i(glGetUniformLocation(lightingShader.Program,
"activaTransparencia"), 0);

               glUniform4f(glGetUniformLocation(lightingShader.Program, "colorAlpha"), 1.0f,
1.0f, 1.0f, 0.3f);

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

               glUniform1i(glGetUniformLocation(lightingShader.Program,
"activaTransparencia"), 0);

               model = glm::translate(model, glm::vec3(-0.60f, 1.82f, -0.43f));

               model = glm::rotate(model, glm::radians(rot2), glm::vec3(0.0f, 1.0f, 0.0));

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


               Vidrio2.Draw(lightingShader);

               glDisable(GL_BLEND);  //Desactiva el canal alfa

               glUniform4f(glGetUniformLocation(lightingShader.Program, "colorAlpha"), 1.0f,
1.0f, 1.0f, 1.0f);



               //Puerta

               model = glm::mat4(1);

               model = glm::translate(model, glm::vec3(-0.60f, 1.82f, -0.43f));

               model = glm::rotate(model, glm::radians(rot2), glm::vec3(0.0f, 1.0f, 0.0));
```

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

                Puerta.Draw(lightingShader);


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

                Puerta.Draw(lightingShader);

                //Barra  Trabajo

                model = glm::mat4(1);

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

                BarraT.Draw(lightingShader);


                //Barra Lareral

                model = glm::mat4(1);

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

                BarraL.Draw(lightingShader);


                //Mesa

                model = glm::mat4(1);

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

                MeCentral.Draw(lightingShader);


                //Pizarron

                model = glm::mat4(1);

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

                Pizarron.Draw(lightingShader);


                //Estante Sup

                model = glm::mat4(1);

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

                EstanSup.Draw(lightingShader);


                //CIENCIA
```

```
        view = camera.GetViewMatrix();

        model = glm::mat4(1);

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

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

        model = glm::translate(model, glm::vec3(-0.36f, 1.96f, 0.570f));

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

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

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


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


        Ciencia.Draw(lightingShader);

        ////////////////////////////////////////////////////////////TRASLADADOS

                //PORAT

        model = glm::mat4(1);

        model = glm::translate(model, glm::vec3(-0.850f, 1.94f, 0.300f));

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

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

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

        Porta.Draw(lightingShader);


        //Material

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

        glUniform1i(glGetUniformLocation(lightingShader.Program,
"activaTransparencia"), 0);

        glEnable(GL_BLEND);//Avtiva la funcionalidad para trabajar el canal alfa

        glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);

        model = glm::mat4(1);


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

        glUniform1i(glGetUniformLocation(lightingShader.Program,
"activaTransparencia"), 0);

        glUniform4f(glGetUniformLocation(lightingShader.Program, "colorAlpha"), 1.0f,
1.0f, 1.0f, 0.7f);

        model = glm::translate(model, glm::vec3(-0.850f, 1.94f, 0.300f));

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

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

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

                glUniform1i(glGetUniformLocation(lightingShader.Program,
"activaTransparencia"), 0);

                Material.Draw(lightingShader);

                glDisable(GL_BLEND);  //Desactiva el canal alfa

                glUniform4f(glGetUniformLocation(lightingShader.Program, "colorAlpha"), 1.0f,
1.0f, 1.0f, 1.0f);






                //Libros

                model = glm::mat4(1);

                model = glm::translate(model, glm::vec3(-0.60f, 1.94f, 0.600f));

                model = glm::scale(model, glm::vec3(0.08f, 0.08f, 0.08f));

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

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

                Libros.Draw(lightingShader);


                //PECERA2

                model = glm::mat4(1);

                model = glm::translate(model, glm::vec3(-0.413f, 1.94f, 0.600f));

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

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

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

                Pecera2.Draw(lightingShader);


                //PECERA1

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

                glUniform1i(glGetUniformLocation(lightingShader.Program,
"activaTransparencia"), 0);

                glEnable(GL_BLEND);//Avtiva la funcionalidad para trabajar el canal alfa

                glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);

                model = glm::mat4(1);


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

```
        glUniform1i(glGetUniformLocation(lightingShader.Program,
"activaTransparencia"), 0);

        glUniform4f(glGetUniformLocation(lightingShader.Program, "colorAlpha"), 1.0f,
1.0f, 1.0f, 0.7f);

        //model = glm::translate(model, glm::vec3(0.0f, 0.0f, 5.0f));

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

        glUniform1i(glGetUniformLocation(lightingShader.Program,
"activaTransparencia"), 0);

        Pecera1.Draw(lightingShader);

        glDisable(GL_BLEND);  //Desactiva el canal alfa

        glUniform4f(glGetUniformLocation(lightingShader.Program, "colorAlpha"), 1.0f,
1.0f, 1.0f, 1.0f);


        //Microandas

        model = glm::mat4(1);

        model = glm::translate(model, glm::vec3(-0.860f, 1.97f, -0.0800f));

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

        model = glm::scale(model, glm::vec3(0.020f, 0.02f, 0.02f));

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

        Micro.Draw(lightingShader);



         //Creaneo


        glUniform3f(glGetUniformLocation(lightingShader.Program, "material.ambient"),
0.25f, 0.20725f, 0.20715f);

        glUniform3f(glGetUniformLocation(lightingShader.Program, "material.diffuse"),
1.0f, 0.8290f, 0.8290f);

        glUniform3f(glGetUniformLocation(lightingShader.Program,
"material.specular"), 0.2966480f, 0.2966480f, 0.296648f);

        glUniform1f(glGetUniformLocation(lightingShader.Program,
"material.shininess"), 0.008f);

        model = glm::mat4(1);

        model = glm::mat4(1);

        model = glm::translate(model, glm::vec3(-0.860f, 1.938f, 0.050f));

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

        model = glm::scale(model, glm::vec3(0.010f, 0.01f, 0.01f));

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

        craneo.Draw(lightingShader);
```

```
               //TETERA

               model = glm::mat4(1);

               model = glm::translate(model, glm::vec3(-0.860f, 1.933f, 0.175f));

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

               model = glm::scale(model, glm::vec3(0.0150f, 0.015f, 0.015f));

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

               Tetera.Draw(lightingShader);




                //Planta

               model = glm::mat4(1);

               model = glm::translate(model, glm::vec3(-0.860f, 1.82f, -0.3f));

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

               model = glm::scale(model, glm::vec3(0.030f, 0.03f, 0.03f));

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

               Planta.Draw(lightingShader);


               //Locker

               model = glm::mat4(1);

               model = glm::translate(model, glm::vec3(-0.820f, 1.82f, -0.4f));

               model = glm::scale(model, glm::vec3(0.030f, 0.03f, 0.03f));

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

               Locker.Draw(lightingShader);



        ////////////////////////////////////////////////////////////////////////////////////7
//FACHADA




                //castillo

               model = glm::mat4(1);

               //model = glm::scale(model, glm::vec3(0.50f, 0.5f, 0.5f));

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

               Castillo.Draw(lightingShader);
```

```
                ////Ventanas der
                glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
                glUniform1i(glGetUniformLocation(lightingShader.Program,
"activaTransparencia"), 0);
                glEnable(GL_BLEND);//Avtiva la funcionalidad para trabajar el canal alfa
                glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
                model = glm::mat4(1);


                glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
                glUniform1i(glGetUniformLocation(lightingShader.Program,
"activaTransparencia"), 0);
                glUniform4f(glGetUniformLocation(lightingShader.Program, "colorAlpha"), 1.0f,
1.0f, 1.0f, 0.5f);
                //model = glm::translate(model, glm::vec3(0.0f, 0.0f, 5.0f));
                glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
                glUniform1i(glGetUniformLocation(lightingShader.Program,
"activaTransparencia"), 0);
                Vent_der.Draw(lightingShader);
                glDisable(GL_BLEND);  //Desactiva el canal alfa
                glUniform4f(glGetUniformLocation(lightingShader.Program, "colorAlpha"), 1.0f,
1.0f, 1.0f, 1.0f);



                ////Ventanas izq
                glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
                glUniform1i(glGetUniformLocation(lightingShader.Program,
"activaTransparencia"), 0);
                glEnable(GL_BLEND);//Avtiva la funcionalidad para trabajar el canal alfa
                glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
                model = glm::mat4(1);


                glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
                glUniform1i(glGetUniformLocation(lightingShader.Program,
"activaTransparencia"), 0);
                glUniform4f(glGetUniformLocation(lightingShader.Program, "colorAlpha"), 1.0f,
1.0f, 1.0f, 0.5f);
                //model = glm::translate(model, glm::vec3(0.0f, 0.0f, 5.0f));
                glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
                glUniform1i(glGetUniformLocation(lightingShader.Program,
"activaTransparencia"), 0);
```

```
		Vent_izq.Draw(lightingShader);
		glDisable(GL_BLEND);  //Desactiva el canal alfa
		glUniform4f(glGetUniformLocation(lightingShader.Program, "colorAlpha"), 1.0f, 1.0f, 1.0f, 1.0f);


		//PUNTA
		model = glm::mat4(1);
		// model = glm::scale(model, glm::vec3(0.5f, 0.5f, 0.5f));
		glUniformMatrix4fv(glGetUniformLocation(lightingShader.Program,  "model"),  1, GL_FALSE, glm::value_ptr(model));
		Punta.Draw(lightingShader);


		//NUBES
		model = glm::mat4(1);
		//model = glm::scale(model, glm::vec3(0.05f, 0.5f, 0.5f));
		glUniformMatrix4fv(glGetUniformLocation(lightingShader.Program,  "model"),  1, GL_FALSE, glm::value_ptr(model));
		Nubes.Draw(lightingShader);




		model = glm::mat4(1);
		//model = glm::scale(model, glm::vec3(0.05f, 0.5f, 0.5f));
		glUniformMatrix4fv(glGetUniformLocation(lightingShader.Program,  "model"),  1, GL_FALSE, glm::value_ptr(model));
		Arcoiris.Draw(lightingShader);



		//TORRES


		model = glm::mat4(1);
		//model = glm::scale(model, glm::vec3(0.05f, 0.5f, 0.5f));
		glUniformMatrix4fv(glGetUniformLocation(lightingShader.Program,  "model"),  1, GL_FALSE, glm::value_ptr(model));
		Torres.Draw(lightingShader);



		//Arbol
		model = glm::mat4(1);
```

```
                //model = glm::scale(model, glm::vec3(0.05f, 0.5f, 0.5f));

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

                Arboles.Draw(lightingShader);

                //Barda

                model = glm::mat4(1);

                //model = glm::scale(model, glm::vec3(0.05f, 0.5f, 0.5f));

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

                Barda.Draw(lightingShader);




                //Barda Paletas

                model = glm::mat4(1);

                //model = glm::scale(model, glm::vec3(0.05f, 0.5f, 0.5f));

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

                BardaPa.Draw(lightingShader);


                //PORTON DER

                model = glm::mat4(1);

                model = glm::translate(model, glm::vec3(0.4120f, 0.05f, 2.250f));

                model = glm::rotate(model, glm::radians(-rot2), glm::vec3(0.0f, 1.0f, 0.0));

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

                Porton_D.Draw(lightingShader);




                //PORTON IZQ

                model = glm::mat4(1);

                model = glm::translate(model, glm::vec3(-0.4120f, 0.05f, 2.250f));

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

                Porton_I.Draw(lightingShader);


                //ENTRADA

                model = glm::mat4(1);

                //model = glm::scale(model, glm::vec3(0.05f, 0.5f, 0.5f));
```

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


               //ESCALERA
               model = glm::mat4(1);
               //model = glm::scale(model, glm::vec3(0.05f, 0.5f, 0.5f));
               glUniformMatrix4fv(glGetUniformLocation(lightingShader.Program,  "model"),  1,
GL_FALSE, glm::value_ptr(model));
               Escaleras.Draw(lightingShader);


               //PISO
               model = glm::mat4(1);
               //model = glm::scale(model, glm::vec3(0.05f, 0.5f, 0.5f));
               glUniformMatrix4fv(glGetUniformLocation(lightingShader.Program,  "model"),  1,
GL_FALSE, glm::value_ptr(model));
               Piso.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)
```

```
Anim.Use();
tiempo = glfwGetTime();
modelLoc = glGetUniformLocation(Anim.Program, "model");
viewLoc = glGetUniformLocation(Anim.Program, "view");
projLoc = glGetUniformLocation(Anim.Program, "projection");

// Set matrices
glUniformMatrix4fv(viewLoc, 1, GL_FALSE, glm::value_ptr(view));
glUniformMatrix4fv(projLoc, 1, GL_FALSE, glm::value_ptr(projection));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1f(glGetUniformLocation(Anim.Program, "time"), tiempo);
model = glm::mat4(1);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Rio.Draw(Anim);



Anim2.Use();
tiempo = glfwGetTime();
modelLoc = glGetUniformLocation(Anim2.Program, "model");
viewLoc = glGetUniformLocation(Anim2.Program, "view");
projLoc = glGetUniformLocation(Anim2.Program, "projection");
glUniformMatrix4fv(viewLoc, 1, GL_FALSE, glm::value_ptr(view));
glUniformMatrix4fv(projLoc, 1, GL_FALSE, glm::value_ptr(projection));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
model = glm::mat4(1);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1f(glGetUniformLocation(Anim.Program, "time"), tiempo);
Carrito.Draw(Anim2);


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));
                        glBindVertexArray(VAO);
                        glDrawArrays(GL_TRIANGLES, 0, 36);
                }
                glBindVertexArray(0);



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


        // Terminate GLFW, clearing any resources allocated by GLFW.
        glfwTerminate();



        return 0;
}

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

        // Camera controls
        if (keys[GLFW_KEY_W] || keys[GLFW_KEY_UP])
        {
                camera.ProcessKeyboard(FORWARD, deltaTime);

        }

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

```
}

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


}

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


}

if (keys[GLFW_KEY_T])
{
        pointLightPositions[0].x += 0.01f;
}
if (keys[GLFW_KEY_G])
{
        pointLightPositions[0].x -= 0.01f;
}
if (keys[GLFW_KEY_O])
{
        if (rot2< 45.0f)
                rot2 += 0.5f;
        mov1 = true;
}
if (mov1)
{
        if (rot2 < 90.0f)
                rot2 += 0.05f;
        else {
                mov1 = false;
                mov2 = true;
        }
}
if (mov2)
{
        if (rot2 > 0.0f)
                rot2 -= 0.05f;
        else {
                mov2 = false;
                mov1 = true;
        }

}
if (keys[GLFW_KEY_1])
{

        rot += 1;

}
//Mov Personaje
if (keys[GLFW_KEY_2])
{
        posZ += 1;
}

if (keys[GLFW_KEY_3])
{
        posZ -= 1;
}

if (keys[GLFW_KEY_4])
{
        posX -= 1;
}

if (keys[GLFW_KEY_5])
{
        posX += 1;
        }
        if (keys[GLFW_KEY_Y])
        {
                pointLightPositions[0].y += 0.01f;
        }

        if (keys[GLFW_KEY_H])
        {
                pointLightPositions[0].y -= 0.01f;
        }
        if (keys[GLFW_KEY_U])
        {
                pointLightPositions[0].z -= 0.1f;
        }
        if (keys[GLFW_KEY_J])
        {
                pointLightPositions[0].z += 0.01f;
        }
        if (keys[GLFW_KEY_N])
        {
                pointLightPositions[3].x += 0.1f;
                pointLightPositions[3].y += 0.1f;
                pointLightPositions[3].z += 0.1f;
                circuito = true;
        }

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

void animacion()
{

        if (circuito)
        {
                if (recorrido1) //Recorrido inicial del carrito
                {
                        rotKit = 90;
                        movKitX += 0.1f;//Z
                        if (movKitX > 90.0f)//>90 sentido del recorrido Z
                        {
                                recorrido1 = false;
                                recorrido2 = true;
                        }
                }
                if (recorrido2)
                {
                        rotKit = -45;//90 sentido del carro
                        movKitZ += 0.1f;// X (+ adelante - atras)
                        movKitX -= 0.1f;
                        if (movKitZ > 90 && movKitX < 30)//>90 X sentido del recorrido (sentidos
iguales >)  && movKitZ < 30
                        {
                                recorrido2 = false;
                                recorrido3 = true;

                        }
                }

                if (recorrido3)
                {
                        rotKit = 270;//180 sentido del carro Z
                        movKitX -= 0.1f;
                        if (movKitX < 0)//<0 Z sentido del recorrido
                        {
                                recorrido3 = false;
                                recorrido4 = true;
                        }
                }

                if (recorrido4)
                {
```

```
			rotKit = 180;//270sentido del carro X
			movKitZ -= 0.1f;
			if (movKitZ < 0)//<0 X sentido del recorrido
			{
				recorrido4 = false;
				recorrido5 = true;
			}
		}

		if (recorrido5)
		{
			rotKit = 0; //sentido del carro Z
			movKitZ += 0.1f;
			if (movKitZ > 0)//>0 Z sentido del recorrido
			{
				recorrido5 = false;
				recorrido1 = 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)
		{
			Light1 = glm::vec3(1.0f, 1.0f, 0.0f);
		}
		else
		{
			Light1 = glm::vec3(0);//Cuado es solo un valor en los 3 vectores pueden dejar solo una componente
		}
	}
}
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
zlastX = xPos;
	lastY = yPos;

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

REQUERIMIENTOS DE FUNCIONAMIENTO

Para que la ejecución del proyecto pueda ser posible, es necesario que en una encuente con el siguiente material:

- Visual Studio 2019 o 2022 community para Windows
- OpenGL versión 3.3
- Archivos:
  - Models (carpeta)
  - Images (carpeta)
  - Shaders (carpeta)
  - SkyBox (carpeta)
  - assimp-vc140-mt.dll
  - glew32.dll
  - 316075189_PROYECTO_FINAL_GPO09
  - 316075189_PROYECTO_FINAL_GPO09.pbd

Todos los archivos deben estar contenidos en una misma carpeta.

CONCLUSION

Podemos concluir con este proyecto que se va a Kcaren todos los temas marcados por el temario, sin embargo podemos ver que aún surgen deficiencias al tratar de aplicarlo ya en un proyecto más grande que en una práctica, ya que estamos trabajando con múltiples objetos que podrán tener distintas alteraciones respecto a un cierto movimieto de algun objeto, posteriormente se espera mejorar este proyecto que sea presentado para que pueda tener una mejor presentación en un futro.