Manual Técnico

Computación Gráfica e Interacción Humano-Computadora

López Reyes Andrés

Grupo: 8

INTRODUCCIÓN

En el presente manual se podrán ver temas que son considerados por el cómputo gráfico como fundamentales, entre ellos se tiene el uso de primitivas básicas para crear figuras en el entorno de desarrollo OpenGL, así como también la recreación de una fachada a través de un software de modelado especializado, en este caso, Blender.
El manual será útil para aquellas personas que inician en el área de la computación gráfica y quieran ver implementaciones de objetos realizados a través de OpenGL y de Blender, de igual forma, aprenderán el cómo exportarlos y trabajar con ellos de manera efectiva.
En ocasiones, suele creerse que es muy complicada la recreación tridimensional y más cuando se trata de objetos realizados en softwares especializados, pero es todo lo contrario, se explicará detalladamente cada etapa.
Cada paso en el manual cuenta con su propia descripción que facilita la comprensión al lector y de esta manera lograr atraer a más gente que quiera iniciar en este campo tan interesante.

ÍNDICE GENERAL

Contenido

OBJETIVOS 3

OBJETIVO GENERAL 3

OBJETIVOS ESPECÍFICOS 3

DIAGRAMA DE GANTT 4

DOCUMENTACIÓN DEL CÓDIGO 5

CABECERAS 5

TAMAÑO DE LA VENTANA 5

PROTOTIPOS DE FUNCIONES 6

CÁMARA 6

VARIABLES PARA ANIMACIONES 6

LUCES E ILUMINACIÓN 7

INICIO DE LA FUNCIÓN PRINCIPAL (MAIN) 7

Definición y carga de shaders 9

Carga de modelos y animaciones 9

Definición de vértices para crear el cubo de la estufa 9

Vértices para el SkyBox 10

Índices para dibujar el cubo 12

Definición del VAO, VBO y SkyBox 12

Carga de texturas y luces 14

Dibujar estufa con primitivas básicas 19

Carga y dibujo de modelos y animaciones 20

Dibuja SkyBox y fin del Main 22

FUNCIONES 23

Movimiento del barco, nube y avión 23

Función para la activación del teclado 25

Animaciones y control de mouse 25

OBJETIVOS

OBJETIVO GENERAL

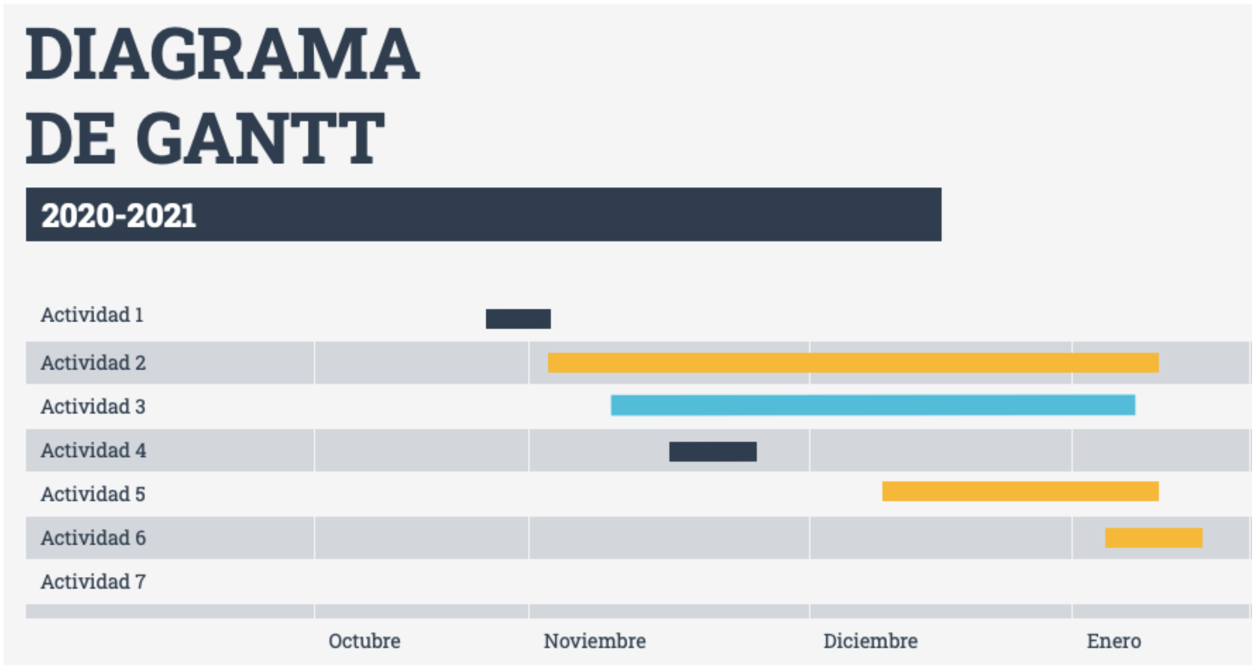
El alumno diseñará una fachada y recreará 7 objetos elegidos previamente, de igual forma, se podrá interactuar dentro del programa, esto para que pueda aplicar y demostrar todos los conocimientos que se adquirieron durante el curso.

OBJETIVOS ESPECÍFICOS

- Recrear 7 objetos recreados deben ser lo más parecidos posible a la imagen de referencia que se entregó previamente.
- Ayudar a las personas que inician con el cómputo gráfico para que logren comprenderlo de una manera más sencilla.
- Profundizar en los temas elementales que se ven en la asignatura.

Diagrama de GANTT

	Nombre de la actividad	Fecha de inicio	Duración (en días)	Fecha fin
Actividad 1	Selección de los 7 objetos a recrear	30-oct-2020	5	4-nov-2020
Actividad 2	Recreación de los objetos en Blender	10-nov-2020	68	17-ene-2021
Actividad 3	Pruebas de carga de modelos con extensión .obj a OpenGL	22-nov-2020	51	12-ene-2021
Actividad 4	Control de cámara	25-nov-2020	5	30-nov-2020
Actividad 5	Recreación de los objetos en OpenGL	15-dic-2020	28	12-ene-2021
Actividad 6	Elaboración de animaciones	10-ene-2021	7	17-ene-2021



Documentación del código

Cabeceras

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

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

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


// Other Libs
#include "stb_image.h"

// GL includes
#include "Shader.h"
#include "Camera.h"
#include "Model.h"
#include "Texture.h"
#include "modelAnim.h"

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

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

Tamaño de la ventana

```
const GLuint WIDTH = 900, HEIGHT = 900;
int SCREEN_WIDTH, SCREEN_HEIGHT;
```

Prototipos de funciones

```
void KeyCallback(GLFWwindow* window, int key, int scancode, int action,
int mode);
void MouseCallback(GLFWwindow* window, double xPos, double yPos);
void DoMovement();
int aux = 0;
void animacion();
```

Cámara

```
Camera camera(glm::vec3(-0.0f, 0.0f, 0.0f));
bool keys[1024];
GLfloat lastX = WIDTH / 2, lastY = HEIGHT / 2;
bool firstMouse = true;
GLfloat deltaTime = 0.0f;
GLfloat lastFrame = 0.0f;
float rot = 0.0f;
```

Variables para animaciones

```
float movKitX = 0.0;
float movKitZ = 0.0;
float rotKit = 0.0;

float movKitX2 = 0.0;
float movKitZ2 = 0.0;
float rotKit2 = 0.0;

float movKitX3 = 0.0;
float movKitZ3 = 0.0;
float rotKit3 = 0.0;

bool circuito = false;
bool circuito2 = false;
bool circuito3 = false;
bool recorrido1 = true;
bool recorrido2 = false;
bool recorrido3 = false;
bool recorrido4 = false;
bool recorrido5 = false;

bool recorrido6 = true;
bool recorrido7 = false;
bool recorrido8 = false;
bool recorrido9 = false;
bool recorrido10 = false;

bool recorrido11 = true;
bool recorrido12 = false;
bool recorrido13 = false;
bool recorrido14 = false;
bool recorrido15 = false;
```

Luces e iluminación

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

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

glm::vec3 LightP1;
```

Inicio de la función principal (Main)

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

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

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

        return EXIT_FAILURE;
    }

    glfwMakeContextCurrent(window);

    glfwGetFramebufferSize(window, &SCREEN_WIDTH, &SCREEN_HEIGHT);

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

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

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

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

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

Definición y carga de shaders

```
Shader shader("Shaders/modelLoading.vs", "Shaders/modelLoading.frag");
    Shader lightingShader("Shaders/lighting.vs", "Shaders/lighting.frag");
    Shader lampShader("Shaders/lamp.vs", "Shaders/lamp.frag");
    Shader SkyBoxshader("Shaders/SkyBox.vs", "Shaders/SkyBox.frag");
    Shader animShader("Shaders/anim.vs", "Shaders/anim.frag");
```

Carga de modelos y animaciones

```
Model ourModel((char*)"Models/Modelos/Casa Kame.obj");
    Model ourModel2((char*)"Models/Modelos/refr.obj");
    Model ourModel3((char*)"Models/Modelos/mesita lampara.obj");
    Model ourModel4((char*)"Models/Modelos/alfombra1.obj");
    Model ourModel5((char*)"Models/Modelos/sillones.obj");
    Model ourModel6((char*)"Models/Modelos/mesa negra.obj");
    Model ourModel7((char*)"Models/Modelos/lampara.obj");
    Model ourModel8((char*)"Models/Modelos/mesa microondas.obj");
    Model ourModel9((char*)"Models/Modelos/microondas.obj");
    Model ourModel10((char*)"Models/Modelos/mar.obj");
    Model nube((char*)"Models/Modelos/nube.obj");
    Model barco((char*)"Models/Modelos/barquito.obj");
    Model avioncito((char*)"Models/Modelos/avioncito.obj");

    //CARGA DE ANIMACIONES
    ModelAnim caida("Animaciones/Personaje/caida.dae");
    caida.initShaders(animShader.Program);

    ModelAnim roshi("Animaciones/Personaje2/roshi.dae");
    roshi.initShaders(animShader.Program);
```

Definición de vértices para crear el cubo de la estufa

```
GLfloat vertices[] =
    {
        //// Positions            // Normals              // Texture Coords


        //creación del cubo
        -0.5f, -0.5f, -0.5f,    0.0f, 0.0f,0.0f,            0.0f,0.0f,
        0.5f, -0.5f, -0.5f,         0.0f, 0.0f,0.0f,        0.1f,0.0f,
        0.5f,  0.5f, -0.5f,     0.0f, 0.0f,0.0f,            0.1f,0.1f,
        -0.5f,  0.5f, -0.5f,    0.0f, 0.0f,0.0f,            0.0f,0.1f, //cara
trasera
```

```
GLfloat vertices[] =
    {
        //// Positions            // Normals              // Texture Coords


        //creación del cubo
        -0.5f, -0.5f, -0.5f,    0.0f, 0.0f,0.0f,            0.0f,0.0f,
        0.5f, -0.5f, -0.5f,         0.0f, 0.0f,0.0f,        0.1f,0.0f,
        0.5f,  0.5f, -0.5f,     0.0f, 0.0f,0.0f,            0.1f,0.1f,
        -0.5f,  0.5f, -0.5f,    0.0f, 0.0f,0.0f,            0.0f,0.1f, //cara
trasera

        -0.5f, -0.5f, 0.5f,    0.0f, 0.0f,0.0f,             0.0f,0.0f,
        0.5f, -0.5f, 0.5f,    0.0f, 0.0f,0.0f,       0.1f,0.0f,
        0.5f,  0.5f, 0.5f,     0.0f, 0.0f,0.0f,          0.1f,0.10f,
        -0.5f,  0.5f, 0.5f,    0.0f, 0.0f,0.0f,             0.0f,0.1f, //cara
frontal

        -0.5f, -0.5f, -0.5f,    0.0f, 0.0f,0.0f,            0.0f,0.0f,
        -0.5f, -0.5f, 0.5f,         0.0f, 0.0f,0.0f,        0.1f,0.0f,
        -0.5f,  0.5f, 0.5f,     0.0f, 0.0f,0.0f,            0.1f,0.1f,
        -0.5f,  0.5f, -0.5f,    0.0f, 0.0f,0.0f,            0.0f,0.1f, //cara
lateral izq


        0.5f, -0.5f, -0.5f,    0.0f, 0.0f,0.0f,             0.0f,0.0f,
        0.5f, -0.5f, 0.5f,    0.0f, 0.0f,0.0f,       0.1f,0.0f,
        0.5f,  0.5f, 0.5f,     0.0f, 0.0f,0.0f,             0.1f,0.1f,
        0.5f,  0.5f, -0.5f,    0.0f, 0.0f,0.0f,             0.0f,0.1f, //cara
lateral der


        -0.5f, 0.5f, 0.5f,    0.0f, 0.0f,0.0f,              0.0f,0.0f,
        0.5f, 0.5f, 0.5f,     0.0f, 0.0f,0.0f,       1.0f,0.0f,
        0.5f,  0.5f, -0.5f,     0.0f, 0.0f,0.0f,            1.0f,1.0f,
        -0.5f,  0.5f, -0.5f,    0.0f, 0.0f,0.0f,            0.0f,1.0f, //cara de
arriba


        -0.5f, -0.5f, 0.5f,    0.0f, 0.0f,0.0f,             0.0f,0.0f,
        0.5f, -0.5f, 0.5f,    0.0f, 0.0f,0.0f,       0.1f,0.0f,
        0.5f,  -0.5f, -0.5f,     0.0f, 0.0f,0.0f,           0.1f,0.1f,
        -0.5f,  -0.5f, -0.5f,    0.0f, 0.0f,0.0f,           0.0f,0.1f //cara de
abajo
    };
```

Vértices para el SkyBox

```
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,
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


                      };
```

Índices para dibujar el cubo

```
GLuint
indices[]
=
                {  // Note that we start from 0!

                    0,1,3,
                    1,2,3,
                    4,5,7,
                    5,6,7,
                    8,9,11,
                    9,10,11,
                    12,13,15,
                    13,14,15,
                    16,17,19,
                    17,18,19,
                    20,21,23,
                    21,22,23

                };
```

Definición del VAO, VBO y SkyBox

```
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 y luces

```
    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);



    // Load textures
    GLuint texture1;
    glGenTextures(1, &texture1);

    int textureWidth, textureHeight, nrChannels;
    stbi_set_flip_vertically_on_load(true);
    unsigned char* image;
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
GL_LINEAR_MIPMAP_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,
GL_NEAREST_MIPMAP_NEAREST);
    // Diffuse map
    image = stbi_load("images/estufa.png", &textureWidth, &textureHeight,
&nrChannels, 0);
    glBindTexture(GL_TEXTURE_2D, texture1);
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, textureWidth, textureHeight, 0,
GL_RGBA, GL_UNSIGNED_BYTE, image);
    glGenerateMipmap(GL_TEXTURE_2D);
    if (image)
    {
        glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, textureWidth, textureHeight, 0,
GL_RGBA, GL_UNSIGNED_BYTE, image);
        glGenerateMipmap(GL_TEXTURE_2D);
    }
    else
    {
        std::cout << "Failed to load texture" << std::endl;
    }
    stbi_image_free(image);
    // Set texture units
    lightingShader.Use();
    glUniform1i(glGetUniformLocation(lightingShader.Program,
"material.diffuse"), 0);
    glm::mat4 projection = glm::perspective(camera.GetZoom(),
(float)SCREEN_WIDTH / (float)SCREEN_HEIGHT, 0.1f, 100.0f);



    // Game loop
    while (!glfwWindowShouldClose(window))
    {
        // Set frame time
        GLfloat currentFrame = glfwGetTime();
        deltaTime = currentFrame - lastFrame;
        lastFrame = currentFrame;

        // Check and call events
        glfwPollEvents();
        DoMovement();
        animacion();

        // Clear the colorbuffer
        glClearColor(1.0f, 1.0f, 1.0f, 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"), 0.0f);
        // == ==========================
        // 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.
        // == ==========================
    // Set texture units
    lightingShader.Use();
    glUniform1i(glGetUniformLocation(lightingShader.Program,
"material.diffuse"), 0);
    glm::mat4 projection = glm::perspective(camera.GetZoom(),
(float)SCREEN_WIDTH / (float)SCREEN_HEIGHT, 0.1f, 100.0f);



    // Game loop
    while (!glfwWindowShouldClose(window))
    {
        // Set frame time
        GLfloat currentFrame = glfwGetTime();
        deltaTime = currentFrame - lastFrame;
        lastFrame = currentFrame;

        // Check and call events
        glfwPollEvents();
        DoMovement();
        animacion();

        // Clear the colorbuffer
        glClearColor(1.0f, 1.0f, 1.0f, 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"), 0.0f);
        // == ==========================
        // 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"), 1.0f, 1.0f, 1.0f);
        glUniform3f(glGetUniformLocation(lightingShader.Program,
"dirLight.ambient"), 1.0f, 1.0f, 1.0f);
        glUniform3f(glGetUniformLocation(lightingShader.Program,
"dirLight.diffuse"), 1.0f, 1.0f, 1.0f);
        glUniform3f(glGetUniformLocation(lightingShader.Program,
"dirLight.specular"), 0.5f, 0.5f, 0.5f);


        // Point light 1
        glUniform3f(glGetUniformLocation(lightingShader.Program,
"pointLights[0].position"), pointLightPositions[0].x, pointLightPositions[0].y,
pointLightPositions[0].z);
        glUniform3f(glGetUniformLocation(lightingShader.Program,
"pointLights[0].ambient"), 0.05f, 0.05f, 0.05f);
        glUniform3f(glGetUniformLocation(lightingShader.Program,
"pointLights[0].diffuse"), LightP1.x, LightP1.y, LightP1.z);
        glUniform3f(glGetUniformLocation(lightingShader.Program,
"pointLights[0].specular"), LightP1.x, LightP1.y, LightP1.z);
        glUniform1f(glGetUniformLocation(lightingShader.Program,
"pointLights[0].constant"), 1.0f);
        glUniform1f(glGetUniformLocation(lightingShader.Program,
"pointLights[0].linear"), 0.09f);
        glUniform1f(glGetUniformLocation(lightingShader.Program,
"pointLights[0].quadratic"), 0.032f);



        // Point light 2
        glUniform3f(glGetUniformLocation(lightingShader.Program,
"pointLights[1].position"), pointLightPositions[1].x, pointLightPositions[1].y,
pointLightPositions[1].z);
        glUniform3f(glGetUniformLocation(lightingShader.Program,
"pointLights[1].ambient"), 0.05f, 0.05f, 0.05f);
        glUniform3f(glGetUniformLocation(lightingShader.Program,
"pointLights[1].diffuse"), 1.0f, 1.0f, 0.0f);
        glUniform3f(glGetUniformLocation(lightingShader.Program,
"pointLights[1].specular"), 1.0f, 1.0f, 0.0f);
        glUniform1f(glGetUniformLocation(lightingShader.Program,
"pointLights[1].constant"), 1.0f);
        glUniform1f(glGetUniformLocation(lightingShader.Program,
"pointLights[1].linear"), 0.09f);
        glUniform1f(glGetUniformLocation(lightingShader.Program,
"pointLights[1].quadratic"), 0.032f);

        // Point light 3
        // 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,
        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
        glActiveTexture(GL_TEXTURE0);
        glBindTexture(GL_TEXTURE_2D, texture1);
```

Dibujar estufa con primitivas básicas

```
        modelo2 = glm::translate(modelo2, glm::vec3(8.1f, -42.8, 30.0f));
        modelo2 = glm::scale(modelo2, glm::vec3(2.7f, 2.7f, 2.7f));
        modelo2 = glm::rotate(modelo2, glm::radians(180.0f),
glm::vec3(0.0f, 1.0f, 0.0f));
        glUniformMatrix4fv(modelLoc, 1, GL_FALSE,
glm::value_ptr(modelo2));

        glDrawElements(GL_TRIANGLES, 6, GL_UNSIGNED_INT, 0);
        glDrawElements(GL_TRIANGLES, 6, GL_UNSIGNED_INT, (GLvoid*)(6 * 1
* sizeof(GLfloat)));
        glDrawElements(GL_TRIANGLES, 6, GL_UNSIGNED_INT, (GLvoid*)(6 * 2
* sizeof(GLfloat)));
        glDrawElements(GL_TRIANGLES, 6, GL_UNSIGNED_INT, (GLvoid*)(6 * 3
* sizeof(GLfloat)));
        glDrawElements(GL_TRIANGLES, 6, GL_UNSIGNED_INT, (GLvoid*)(6 * 4
* sizeof(GLfloat)));
```

```
                glDrawElements(GL_TRIANGLES, 6, GL_UNSIGNED_INT, (GLvoid*)(6 * 5
        * sizeof(GLfloat)));
                glBindVertexArray(0);

                glBindVertexArray(VAO);
                glm::mat4 modelo(1);
                glm::mat4 modelo3(1);
```

Carga y dibujo de modelos y animaciones

```
                 //BARQUITO

                view = camera.GetViewMatrix();
                modelo = glm::mat4(1);
                modelo = glm::scale(modelo, glm::vec3(0.2f, 0.2f, 0.2f));
                modelo = glm::translate(modelo, PosIni + glm::vec3(movKitX2, 0,
        movKitZ2));
                modelo = glm::scale(modelo, glm::vec3(0.2f, 0.2f, 0.2f));
                modelo = glm::translate(modelo, glm::vec3(300.0f, -1240.0f, -
        400.0f));
                modelo = glm::rotate(modelo, glm::radians(rotKit2), glm::vec3(0.0f,
        1.0f, 0.0f));
                glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(modelo));
                barco.Draw(lightingShader);
                glBindVertexArray(0);

                //Nube de Goku
                modelo2 = glm::mat4(1);
                view = camera.GetViewMatrix();
                modelo2 = glm::mat4(1);
                modelo2 = glm::scale(modelo2, glm::vec3(0.2f, 0.2f, 0.2f));
                modelo2 = glm::translate(modelo2, PosIni + glm::vec3(movKitX, 0,
        movKitZ));
                modelo2 = glm::scale(modelo2, glm::vec3(0.2f, 0.2f, 0.2f));
                modelo2 = glm::translate(modelo2, glm::vec3(300.0f, -700.0f,
        200.0f));
                glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(modelo2));
                nube.Draw(lightingShader);



                //Avioncito
                modelo3 = glm::mat4(1);
                view = camera.GetViewMatrix();
               modelo3 = glm::mat4(1);
               modelo3 = glm::scale(modelo3, glm::vec3(0.2f, 0.2f, 0.2f));
               modelo3 = glm::translate(modelo3, PosIni + glm::vec3(movKitX3, 0,
       movKitZ3));
               modelo3 = glm::scale(modelo3, glm::vec3(0.18f, 0.18f, 0.18f));
               modelo3 = glm::translate(modelo3, glm::vec3(300.0f, -600.0f,
       500.0f));
               modelo3 = glm::rotate(modelo3, glm::radians(rotKit3),
       glm::vec3(0.0f, 1.0f, 0.0f));
               glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(modelo3));
               avioncito.Draw(lightingShader);
               glBindVertexArray(0);

               shader.Use();

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

               // Draw the loaded model
               glm::mat4 model(1);
               model = glm::scale(model, glm::vec3(0.1f, 0.1f, 0.1f));    // It's a
       bit too big for our scene, so scale it down
               model = glm::translate(model, glm::vec3(0.0f, -500.0, 200.0f)); //
       Translate it down a bit so it's at the center of the scene
               glUniformMatrix4fv(glGetUniformLocation(shader.Program, "model"), 1,
       GL_FALSE, glm::value_ptr(model));

               //DIBUJA MODELOS

               ourModel.Draw(shader); //Se dibuja la fachada, isla, mesa  y sillas
       del comedor
               ourModel2.Draw(shader); //Se dibuja el refrigerador
               ourModel3.Draw(shader); //Se dibuja la mea de la lampara
               ourModel4.Draw(shader); //Se dibuja la alfombra
               ourModel5.Draw(shader); //Se dibujan los sillones rojos
               ourModel6.Draw(shader); //Se dibuja la mesa negra
               ourModel7.Draw(shader); //Se dibuja la lámpara
               ourModel8.Draw(shader); //Se dibuja la mesa microondas
               ourModel9.Draw(shader); //Se dibuja el microondas
               ourModel10.Draw(shader); //Se dibuja el mar

               //Dibuja personaje animado
               animShader.Use();
               modelLoc = glGetUniformLocation(animShader.Program, "model");
              modelLoc = glGetUniformLocation(animShader.Program, "model");
              viewLoc = glGetUniformLocation(animShader.Program, "view");
              projLoc = glGetUniformLocation(animShader.Program, "projection");
              glUniformMatrix4fv(viewLoc, 1, GL_FALSE, glm::value_ptr(view));
              glUniformMatrix4fv(projLoc, 1, GL_FALSE,
      glm::value_ptr(projection));
              glUniform3f(glGetUniformLocation(animShader.Program,
      "material.specular"), 0.5f, 0.5f, 0.5f);
              glUniform1f(glGetUniformLocation(animShader.Program,
      "material.shininess"), 32.0f);
              glUniform3f(glGetUniformLocation(animShader.Program,
      "light.ambient"), 1.0f, 1.0f, 1.0f);
              glUniform3f(glGetUniformLocation(animShader.Program,
      "light.diffuse"), 1.0f, 1.0f, 1.0f);
              glUniform3f(glGetUniformLocation(animShader.Program,
      "light.specular"), 1.0f, 1.0f, 1.0f);
              glUniform3f(glGetUniformLocation(animShader.Program,
      "light.direction"), 0.0f, -1.0f, -1.0f);
              view = camera.GetViewMatrix();

              //Animación caída
              model = glm::mat4(1);
              model = glm::translate(model, glm::vec3(2.1f, -44.1, 30.0f));
              model = glm::scale(model, glm::vec3(0.02f)); // it's a bit too big
      for our scene, so scale it down
              glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
              caida.Draw(animShader);
              glBindVertexArray(0);

              //Animación Roshi
              modelo = glm::mat4(1);
              modelo = glm::translate(modelo, glm::vec3(-10.1f, -44.1, 25.0f));
              modelo = glm::scale(modelo, glm::vec3(0.02f));      // it's a bit
      too big for our scene, so scale it down
              glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(modelo));
              roshi.Draw(animShader);
              glBindVertexArray(0);
```

Dibuja SkyBox y fin del Main

```
        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);

            glfwTerminate();
            return 0;
        }
```

Funciones

Movimiento del barco, nube y avión

```
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_I])
{
    pointLightPositions[3].x += 0.1f;
    pointLightPositions[3].y += 0.1f;
    pointLightPositions[3].z += 0.1f;
    circuito = true;
}

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

if (keys[GLFW_KEY_K])
{
    pointLightPositions[3].x += 0.1f;
    pointLightPositions[3].y += 0.1f;
    pointLightPositions[3].z += 0.1f;
    circuito2 = true;
}

if (keys[GLFW_KEY_L])
{
    circuito2 = false;
}

if (keys[GLFW_KEY_N])
{
    pointLightPositions[3].x += 0.1f;
    pointLightPositions[3].y += 0.1f;
    pointLightPositions[3].z += 0.1f;
    circuito3 = true;

}
```

```
                    if (keys[GLFW_KEY_M])
                    {
                        circuito3 = false;

                    }
                }
```

Función para la activación del teclado

```
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;
        }
    }
```

Animaciones y control de mouse

```
void animacion()
{
    //animación de la nube

    if (circuito)
    {
        if (recorrido1)
        {
            rotKit = 90;
            movKitX += 0.1f;
            if (movKitX > 90)
            {
                recorrido1 = false;
                recorrido2 = true;
            }
        }
        if (recorrido2)
```

```
void animacion()
{
    //animación de la nube

    if (circuito)
    {
        if (recorrido1)
        {
            rotKit = 90;
            movKitX += 0.1f;
            if (movKitX > 90)
            {
                recorrido1 = false;
                recorrido2 = true;
            }
        }
        if (recorrido2)
        {
            rotKit = 315;
            movKitZ += 0.1f;
            movKitX -= 0.1f;
            if (movKitZ > 90)
            {
                recorrido2 = false;
                recorrido3 = true;

            }
        }

        if (recorrido3)
        {
            rotKit = 180;
            movKitZ -= 0.1f;
            if (movKitZ < 0)
            {
                recorrido3 = false;
                rotKit = 90;
                recorrido1 = true;
            }
        }





    }
    //Animación del barquito
    if (circuito2)
    {
        if (recorrido6)
        {
            movKitZ2 += 0.05f;
            if (movKitZ2 > 90)
            {
                recorrido6 = false;
                recorrido7 = true;
            }
        }
        if (recorrido7)
        {
            rotKit2 = 90;
            movKitX2 += 0.05f;
            if (movKitX2 > 90)
            {
                recorrido7 = false;
                recorrido8 = true;

            }
        }
        if (recorridoB)
        {
            rotKit2 = 180;
            movKitZ2 -= 0.05f;
            if (movKitZ2 < 0)
            {
                recorrido8 = false;
                recorrido9 = true;
            }
        }

        if (recorrido9)
        {
            rotKit2 = 270;
            movKitX2 -= 0.05f;
            if (movKitX2 < 0)
            {
                recorrido9 = false;
                recorrido10 = true;
            }
        }
        if (recorrido10)
        {
            rotKit2 = 0;
            movKitZ2 += 0.05f;
            if (movKitZ2 > 0)
            {
                recorrido10 = false;
                recorrido6 = true;
            }
        }
    }

    //Animación del avioncito
    if (circuito3)
    {
        if (recorrido11)
        {
            rotKit3 = 90;
            movKitX3 += 0.1f;
            if (movKitX3 > 90)
            {
                recorrido11 = false;
                recorrido12 = true;
            }
        }
        if (recorrido12)
        {
            rotKit3 = 0;
            movKitZ3 += 0.1f;
            if (movKitZ3 > 90)
            {
                recorrido12 = false;
                recorrido13 = true;

            }
        }

        if (recorrido13)
        {
            rotKit3 = 270;
            movKitX3 -= 0.1f;
            if (movKitX3 < 0)
            {
                recorrido13 = false;
                recorrido14 = true;
            }
        }
        if (recorrido14)
        {
            rotKit3 = 180;
            movKitZ3 -= 0.1f;
            if (movKitZ3 < 0)
            {
                recorrido14 = false;

                recorrido15 = true;
            }
        }
        if (recorrido15)
        {
            rotKit3 = 90;
            movKitX3 -= 0.1f;
            if (movKitX3 < 90)
            {
                recorrido15 = false;


                recorrido11 = true;
            }
        }
    }
}




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);
}
```