```cpp
/**
 * @file 419048901_Proyecto_GPO04.cpp
 * @brief Archivo principal CPP (main program) del proyecto
 * @author NumCuenta: 419048901
 * @date 11/05/2022
 */

// Operaciones E/S
#include <iostream>

// Oeraciones Matematicas
#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"
#include "modelAnim.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
```

```cpp
Camera  camera(glm::vec3(0.0f, 10.0f, 25.0f));
GLfloat lastX = WIDTH / 2.0;
GLfloat lastY = HEIGHT / 2.0;
bool keys[1024];
bool firstMouse = true;


// Light attributes
glm::vec3 lightPos(0.0f, 0.0f, 0.0f);
glm::vec3 PosIni(-16.0f, 1.0f, -70.0f);
glm::vec3 lightDirection(0.0f, -1.0f, -1.0f);
bool active;

bool encendido = false;
// Positions of the point lights
glm::vec3 pointLightPositions[] = {
    glm::vec3(0.0f, 19.0f, 0.0f)
};

// Position of the SpotLight
glm::vec3 spotLightPosition = glm::vec3(0.0f, 19.0f, 0.0f);

int dir = 0;
// Directions of the SpotLight
glm::vec3 spotLightDir[] = {
    glm::vec3(0.0f,-1.0f, 0.0f),  // Abajo
    glm::vec3(1.0f,0.0f, 0.0f),  // Derecha
    glm::vec3(0.0f,0.0f, -1.0f), // Atras
    glm::vec3(-1.0f,0.0f, 0.0f), // Izquierda
    glm::vec3(0.0f,0.0f, 1.0f),  // Frente
    glm::vec3(0.0f,1.0f, 0.0f),  // Arriba
    glm::vec3(0.0f,-1.0f, 0.0f)  // Abajo
};

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
};

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
};

glm::vec3 Light1 = glm::vec3(0);
glm::vec3 Light2 = glm::vec3(0);
glm::vec3 Light3 = glm::vec3(0);
glm::vec3 Light4 = glm::vec3(0);

/**
 * \var rotDor, actionDoor, openDoor
 * \brief Variables Animación de Puerta
 */
float rotDoor = 0.0f;
bool actionDoor = false, openDoor = false;

/**
 * \var rotCam, CamDerecha
 * \brief Variables Animación de Camara Seguridad
 */
float rotCam = 0.0;
bool CamDerecha = false;

/**
 * \var posIniCar, movKitXY, rotKit, circuito, recorridos1-8
 * \brief Variables Animación del coche
 */
glm::vec3 PosIniCar(80.0f, 0.0f, 14.0f);
```

```cpp
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;bool recorrido6 = false; bool recorrido7 = false; bool recorrido8 = false;

/**
 * \var posIniPerson
 * \brief Variable Animación del Personaje
 */
glm::vec3 PosIniPerson(-16.0f, 0.0f, -70.0f);

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


/**
 * \fn int main()
 * \brief Funcion del programa principal
 * \return Devuelve 0 de programa exitoso
 */
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 Gimnasio \"FORM\" :  419048901 - GPO04", 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);

    // Carga de Shaders
    Shader lightingShader("Shaders/lighting.vs", "Shaders/lighting.frag");
    Shader lampShader("Shaders/lamp.vs", "Shaders/lamp.frag");
    Shader SkyBoxshader("Shaders/SkyBox.vs", "Shaders/SkyBox.frag");
    Shader animShader("Shaders/anim.vs", "Shaders/anim.frag");

    // Carga de modelos de gimnasio
    Model Piso((char*)"Models/Gym/calles.obj");
    Model Habitacion((char*)"Models/Gym/habitacion.obj");
    Model Entrada((char*)"Models/Gym/entrada.obj");
    Model Puerta((char*)"Models/Gym/puertaPrincipal.obj");
    Model Estante((char*)"Models/Gym/estante.obj");
    Model Ventanas((char*)"Models/Gym/ventanas.obj");
    Model Marcos((char*)"Models/Gym/marcos.obj");
    Model Hidrante((char*)"Models/Ambiente/hidrante.obj");

    Model Banca_inclinada((char*)"Models/Gym/banca_inclinada.obj");
    Model Caminadora((char*)"Models/Gym/caminadora.obj");
    Model Barra((char*)"Models/Gym/barra.obj");
    Model Rack((char*)"Models/Gym/rack.obj");
    Model Rack2((char*)"Models/Gym/rack2.obj");
    Model Multi((char*)"Models/Gym/barraMulti.obj");

    Model Mancuerna((char*)"Models/Gym/mancuerna.obj");
    Model Mancuerna2((char*)"Models/Gym/mancuerna2.obj");
    Model Mancuerna3((char*)"Models/Gym/mancuerna3.obj");
    Model Rusa1((char*)"Models/Gym/rusa1.obj");
    Model Rusa2((char*)"Models/Gym/rusa2.obj");
    Model Rusa3((char*)"Models/Gym/rusa3.obj");
    Model Suiza((char*)"Models/Gym/suiza.obj");
    Model Gorra1((char*)"Models/Gym/gorra1.obj");
    Model Gorra2((char*)"Models/Gym/gorra2.obj");
    Model Gorra3((char*)"Models/Gym/gorra3.obj");

    // Carga de modelos de animación
    ModelAnim animacionPersonaje("Animaciones/abdominal.dae");
    Model Carro((char*)"Models/Ambiente/lamborginhi.obj");
    Model Soporte((char*)"Models/Ambiente/soporte.obj");
    Model Camara((char*)"Models/Ambiente/camara.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);

    // SkyBox attributes
    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);
```

```cpp
    glEnableVertexAttribArray(0);
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(GLfloat), (GLvoid*)0);

    // Load textures
    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 matrix Projection
    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();
        animacion();

        // 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);


        // Use cooresponding shader when setting uniforms/drawing objects
        /* -------- Lighting Shader --------*/
        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"), 0.45f,0.45f,0.45f);  // Luz ambiente + DiffuseModify
        glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.diffuse"), 0.1f, 0.1f, 0.1f);
        glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.specular"), 0.35f, 0.35f, 0.35f);

        // Point light
        glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].position"), pointLightPositions[0].x, pointLightPositions[0].y, pointLightPositions[0].z);
        glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].ambient"), 1.0f, 1.0f, 1.0f);
        glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].diffuse"), 1.0f, 1.0f, 1.0f);
        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"), 0.045f);
        glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[0].quadratic"), 0.0075f);

        // SpotLight GIANT
        glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight.position"), spotLightPosition.x, spotLightPosition.y, spotLightPosition.z);
        glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight.direction"), spotLightDir[dir].x, spotLightDir[dir].y, spotLightDir[dir].z);
        glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight.ambient"), 0.05f, 0.05f, 0.05f);
        glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight.diffuse"), 0.2f, 0.2f, 0.2f);
        glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight.specular"),0.05f, 0.05f, 0.05f);
        glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight.constant"), 1.0f);
        glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight.linear"), 0.045f);
        glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight.quadratic"), 0.0075f);
        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 = 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));

        // Obtener matriz de Vista
        view = camera.GetViewMatrix();

        // Operar y dibujar modelo de PISO
        glm::mat4 model(1);
        model = glm::mat4(1);
        model = glm::translate(model, glm::vec3(0.0f, 0.8f, 0.0f));
        glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
        glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
        Piso.Draw(lightingShader);

        // Operar y dibujar modelo de HIDRANTE
        model = glm::mat4(1);
        model = glm::translate(model, glm::vec3(0.0f, 0.0f, 5.0f));
        glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
        Hidrante.Draw(lightingShader);

        // Operar y dibujar modelo de HABITACION GIMNASIO
        model = glm::mat4(1);
        glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
        Habitacion.Draw(lightingShader);

        // Operar y dibujar modelos (ELEMENTO) Bancas inclinadas
```

```cpp
        model = glm::mat4(1);
        model = glm::translate(model, glm::vec3(13.0f, 0.0f, -41.0f));
        model = glm::rotate(model, glm::radians(90.0f), glm::vec3(0.0f, 
          -1.0f, 0.0f));
        glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
        Banca_inclinada.Draw(lampShader);

        model = glm::mat4(1);
        model = glm::translate(model, glm::vec3(13.0f, 0.0f, -33.0f));
        model = glm::rotate(model, glm::radians(90.0f), glm::vec3(0.0f, 
          -1.0f, 0.0f));
        glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
        Banca_inclinada.Draw(lampShader);

        // Operar y dibujar modelos (ELEMENTO) Caminadoras
        model = glm::mat4(1);
        model = glm::translate(model, glm::vec3(22.0f, 0.0f, -68.0f));
        glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
        Caminadora.Draw(lampShader);

        model = glm::mat4(1);
        model = glm::translate(model, glm::vec3(13.0f, 0.0f, -68.0f));
        glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
        Caminadora.Draw(lampShader);

        model = glm::mat4(1);
        model = glm::translate(model, glm::vec3(4.0f, 0.0f, -68.0f));
        glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
        Caminadora.Draw(lampShader);

        model = glm::mat4(1);
        model = glm::translate(model, glm::vec3(-5.0f, 0.0f, -68.0f));
        glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
        Caminadora.Draw(lampShader);

        // Operar y dibujar modelos (ELEMENTO) Barras
        model = glm::mat4(1);
        model = glm::translate(model, glm::vec3(-20.0f, 0.0f, -64.0f));
        glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
        Barra.Draw(lampShader);

        model = glm::mat4(1);
        model = glm::translate(model, glm::vec3(-28.0f, 0.0f, -53.0f));
        glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
        Barra.Draw(lampShader);

        // Operar y dibujar modelos (ELEMENTO) Racks mancuernas
        model = glm::mat4(1);
        glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
        Rack.Draw(lampShader);

        model = glm::mat4(1);
        glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
        Rack2.Draw(lampShader);

        // Operar y dibujar modelos (ELEMENTO) Mancuernas
        model = glm::mat4(1);
        model = glm::translate(model, glm::vec3(3.0f, 0.3f, 0.0f));
        glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
        Mancuerna.Draw(lampShader);
        model = glm::mat4(1);
        model = glm::translate(model, glm::vec3(-4.0f, 0.3f, 0.0f));
        glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
        Mancuerna.Draw(lampShader);
        model = glm::mat4(1);
        model = glm::translate(model, glm::vec3(-3.0f, 0.3f, 0.5f));
        model = glm::rotate(model, glm::radians(15.0f), glm::vec3(0.0f,
          -1.0f, 0.0f));
        glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
        Mancuerna.Draw(lampShader);

        // Operar y dibujar modelo Estante
        model = glm::mat4(1);
        glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
        Estante.Draw(lightingShader);

        // Operar y dibujar modelo (ELEMENTO) Multiejercicios - Fondos
        model = glm::mat4(1);
        model = glm::translate(model, glm::vec3(-12.0f, 9.0f, -88.5f));
        glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
        Multi.Draw(lightingShader);

        // Operar y dibujar modelos (ELEMENTO) Pesas Rusas
        model = glm::mat4(1);
        model = glm::translate(model, glm::vec3(-21.0f, 0.0f, -42.0f));
        glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
        Rusa1.Draw(lightingShader);
        model = glm::mat4(1);
        model = glm::translate(model, glm::vec3(-23.0f, 0.0f, -51.0f));
        model = glm::rotate(model, glm::radians(30.0f), glm::vec3(0.0f,
          1.0f, 0.0f));
        glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
        Rusa1.Draw(lightingShader);

        model = glm::mat4(1);
        model = glm::translate(model, glm::vec3(-21.0f, 0.0f, -52.0f));
        glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
        Rusa2.Draw(lightingShader);
        model = glm::mat4(1);
        model = glm::translate(model, glm::vec3(-20.2f, 0.0f, -55.0f));
        glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
        Rusa2.Draw(lightingShader);

        model = glm::mat4(1);
        model = glm::translate(model, glm::vec3(-21.0f, 0.0f, -62.0f));
        glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
        Rusa3.Draw(lightingShader);

        // Operar y dibujar modelos (ELEMENTO) Pelotas Suizas
        model = glm::mat4(1);
        model = glm::translate(model, glm::vec3(-15.0f, 0.0f, -26.5f));
        model = glm::scale(model, glm::vec3(1.5f, 1.5f, 1.5f));
        model = glm::rotate(model, glm::radians(180.0f), glm::vec3(0.0f, 
          1.0f, 0.0f));
        glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
        Suiza.Draw(lightingShader);

        model = glm::mat4(1);
        model = glm::translate(model, glm::vec3(-19.0f, 0.0f, -25.2f));
        model = glm::scale(model, glm::vec3(1.5f, 1.5f, 1.5f));
        model = glm::rotate(model, glm::radians(180.0f), glm::vec3(0.0f, 
          1.0f, 0.0f));
        glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
        Suiza.Draw(lightingShader);

        // Operar y dibujar modelos de Marcos Interiores
        model = glm::mat4(1);
        glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
        Marcos.Draw(lightingShader);

        // Operar y dibujar modelo de Entrada
        model = glm::mat4(1);
        glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
        Entrada.Draw(lightingShader);

        // Operar y dibujar modelo de Entrada - Puerta
        model = glm::mat4(1);
        model = glm::translate(model, glm::vec3(12.3f, 1.4f, -31.7f));
        model = glm::rotate(model, glm::radians( rotDoor ), glm::vec3
          (0.0f, 1.0f, 0.0f));
        glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
        Puerta.Draw(lightingShader);

        // Operar y dibujar modelos (ELEMENTO) Accesorios - Gorras
        model = glm::mat4(1);
        model = glm::translate(model, glm::vec3(0.0f, 4.12f, 3.4f));
        glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
        Gorra1.Draw(lightingShader);

        model = glm::mat4(1);
        model = glm::translate(model, glm::vec3(0.0f, 4.12f, 3.5f));
        glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
        Gorra2.Draw(lightingShader);

        model = glm::mat4(1);
        model = glm::translate(model, glm::vec3(0.0f, 4.12f, 3.7f));
        glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
        Gorra3.Draw(lightingShader);

        model = glm::mat4(1);
        model = glm::translate(model, glm::vec3(9.0f, 6.0f, 13.5f));
        glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
        Gorra3.Draw(lightingShader);

        // Operar y dibujar modelo de Carro
        model = glm::mat4(1);
        model = glm::translate(model, PosIniCar + glm::vec3(movKitX, 0,
          movKitZ));
        model = glm::rotate(model, glm::radians(rotKit), glm::vec3(0.0f,
          1.0f, 0.0));
        model = glm::scale(model, glm::vec3(1.5f, 1.5f, 1.5f));
        glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
        Carro.Draw(lightingShader);

        // Operar y dibujar modelo de Soportes de Camaras
        model = glm::mat4(1);
        model = glm::translate(model, glm::vec3(-12.0f, 15.0f, -82.6f));
        model = glm::scale(model, glm::vec3(2.0f, 2.0f, 2.0f));
        glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
        Soporte.Draw(lightingShader);

        model = glm::mat4(1);
        model = glm::translate(model, glm::vec3(-8.0f, 16.0f, -13.0f));
        model = glm::scale(model, glm::vec3(2.0f, 2.0f, 2.0f));
        glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
        Soporte.Draw(lightingShader);

        // Operar y dibujar modelo de Camaras
        model = glm::mat4(1);
        model = glm::translate(model, glm::vec3(-12.0f, 15.0f, -82.6f));
        model = glm::rotate(model, glm::radians(rotCam), glm::vec3(0.0f,
          1.0f, 0.0));
        model = glm::scale(model, glm::vec3(2.0f, 2.0f, 2.0f));
        glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
        Camara.Draw(lightingShader);

        model = glm::mat4(1);
        model = glm::translate(model, glm::vec3(-8.0f, 16.0f, -13.0f));
        model = glm::rotate(model, glm::radians(rotCam), glm::vec3(0.0f, 1.0f, 0.0));
        model = glm::scale(model, glm::vec3(2.0f, 2.0f, 2.0f));
        glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
        Camara.Draw(lightingShader);

        /* -------- Modelos con transparencia --------*/
        glEnable(GL_BLEND);// Activa la funcionalidad para trabajar el canal alfa
        glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);

        // Operar y dibujar modelo de Ventanas
        model = glm::mat4(1);
        glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
        glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 1);
        glUniform4f(glGetUniformLocation(lightingShader.Program, "colorAlpha"), 0.0f, 0.0f, 0.0f, 0.05f);
        Ventanas.Draw(lightingShader);

        glDisable(GL_BLEND);  //Desactiva el canal alfa
        glUniform4f(glGetUniformLocation(lightingShader.Program, "colorAlpha"), 1.0f, 1.0f, 1.0f, 1.0f);
        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)
        ////for (GLuint i = 0; i < 1; i++)
```

```cpp
        ////{
        ////    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);


        /* -------- Animacion Shader --------*/
        /*_________ Personaje Animado (Abdominales)  __________*/
        animacionPersonaje.initShaders(animShader.Program);
        animShader.Use();
        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"), 12.0f);
        glUniform3f(glGetUniformLocation(animShader.Program, "light.ambient"), 0.75f, 0.75f, 0.75f);
        glUniform3f(glGetUniformLocation(animShader.Program, "light.diffuse"), 0.75f, 0.75f, 0.75f);
        glUniform3f(glGetUniformLocation(animShader.Program, "light.specular"), 0.5f, 0.5f, 0.5f);
        glUniform3f(glGetUniformLocation(animShader.Program, "light.direction"), 0.0f, -1.0f, -1.0f);
        view = camera.GetViewMatrix();

        model = glm::mat4(1);
        model = glm::translate(model, glm::vec3(PosIniPerson.x, PosIniPerson.y, PosIniPerson.z));
        model = glm::scale(model, glm::vec3(0.06f));// ESCALAR ANIMACION al 6%
        glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
        animacionPersonaje.Draw(animShader);
        glBindVertexArray(0);



        /* -------- SKYBOX Shader --------*/
        // Atributos SKYBOX
        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));

        // Dibujar SKYBOX
        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);
    }


    // Terminate GLFW, clearing any resources allocated by GLFW.
    glfwTerminate();
    return 0;
}

/**
 * \fn void DoMovement()
 * \brief Modifica posiciones de Camara respecto a Entradas de Usuario
 */
void DoMovement()
{

    // Controles de Camara
    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);
    }


    // Control de Animacion Puerta
    if (keys[GLFW_KEY_F])
    {
        actionDoor = true;
    }

    // Control de Animacion Coche
    if (keys[GLFW_KEY_Z])
    {
        circuito = true;
    }

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


}


/**
 * \fn void animacion()
 * \brief Realiza animaciones de objetos, modificando las variables para operaciones basicas
 */
void animacion()
{

    //Movimiento de Camara Seguridad
    rotCam += (CamDerecha) ? 0.3f : -0.3f ;
    CamDerecha = (rotCam >= 90.0f) ? false : CamDerecha;
    CamDerecha = (rotCam <= -90.0f) ? true : CamDerecha;


    //Movimiento de Puerta
    if (actionDoor) {
        rotDoor += (openDoor) ? -0.8f : 0.8f ;
        if (rotDoor <= 0.0f) {
            openDoor = false;
            actionDoor = false;
        }
        if (rotDoor >= 90.0f) {
            openDoor = true;
            actionDoor = false;
        }
    }


    //Movimiento del coche
    if (circuito)
    {
        if (recorrido1)
        {
            rotKit = 0.0f;
            movKitX -= 0.2f;
            if ( movKitX < -35.0f )
            {
                recorrido1 = false;
                recorrido2 = true;
            }
        }

        if (recorrido2)
        {
            rotKit = -45.0f;
            movKitX -= 0.1f;
            movKitZ -= 0.1f;
            if ( movKitX < -50.0f && movKitZ < -15.0f )
            {
                recorrido2 = false;
                recorrido3 = true;
            }
        }

        if (recorrido3)
        {
            rotKit = 0.0f;
            movKitX -= 0.05f;
            if ( movKitX < -90.0f )
            {
                recorrido3 = false;
                recorrido4 = true;
            }
        }

        if (recorridoo4)
        {
            rotKit = 45.0f;
            movKitX -= 0.1f;
            movKitZ += 0.1f;
            if ( movKitX < -105.0f && movKitZ > 0.0f )
            {
                recorrido4 = false;
                recorrido5 = true;
            }
        }


        if (recorrido5)
        {
            rotKit = 0.0f;
            movKitX -= 0.2f;
            if (movKitX < -150.0f)
            {
                recorrido5 = false;
                recorrido6 = true;
            }
        }

        if (recorrido6)
        {
            rotKit = 90.0f;
            movKitZ += 0.2f;
            if ( movKitZ > 14.0f )
            {
                recorrido6 = false;
                recorrido7 = true;
            }
        }

        if (recorrido7)
        {
            rotKit = 180.0f;
            movKitX += 0.2f;
            if ( movKitX > 0.0f )
            {
                recorrido7 = false;
                recorrido8 = true;
            }
        }

        if (recorrido8)
        {
```

```cpp
            rotKit = -90.0f;
            movKitZ -= 0.2f;
            if (movKitZ < 0.0f)
            {
                recorrido8 = false;
                movKitX = 0.0f;
                movKitZ = 0.0f;
                recorrido1 = true;
            }
        }

    }


}


/**
 * \fn void KeyCallback()
 * \brief Opera cada que se presiona/libera una tecla a través de 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;
        }
    }

}


/**
 * \fn void MouseCallback()
 * \brief Procesa los movimientos del Mouse sobre la Camara en Ventana Principal
 */
```

```
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);
}
```