UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

SEMESTRE: 2021 - II

LAB. COMPUTACIÓN GRÁFICA E INTERACCIÓN HUMANO-COMPUTADORA
ING. CARLOS ALDAIR ROMAN BALBUENA

GRUPO: 08

ALUMNO: MARCELO ROMERO ADOLFO

PROYECTO FINAL
MANUAL TÉCNICO

FECHA DE ENTREGA:
29/07/2021

Índice

Manual técnico

Objetivos del proyecto:

1. El alumno deberá aplicar y demostrar los conocimientos adquiridos durante todo el curso.
2. Representar en un ambiente virtual la casa icónica de la serie Rick y Morty desarrollando en Visual Studio el código que permita visualizar cada uno de los elementos propuestos para recrear dicho escenario, Además de presentar modelos con animación sujeta a las características de la serie.
3. El proyecto debe de funcionar de manera correcta según lo implementado en el respectivo repositorio git de descarga

Descripción:

El alumno deberá seleccionar una fachada y un espacio que pueden ser reales o ficticios y presentar imágenes de referencia de dichos espacios para su recreación 3D en OpenGL.
En la imagen de referencia se debe visualizar 7 objetos que el alumno va a recrear virtualmente y donde dichos objetos deben ser lo más parecido a su imagen de referencia, así como su ambientación.
Mi selección fue recrear la casa de Rick y Morty de la serie animada con el mismo con el mismo nombre, la cual fue realizada con la mayoría de los detalles planteados en la propuesta:

Diagrama de Gant:

Actividades	JUNIO		JULIO			
	Semana 3	Semana 4	Semana 1	Semana 2	Semana 3	Semana 4
Planeación de desarrollo de Proyecto						
Recreación del repositorio en GIT						
Búsqueda de modelos con licencia.						
Búsqueda e implementación de texturas						
Implementación de modelos al proyecto.						
Implementación de animaciones						
Pruebas						
Documentación del proyecto.						
Integración de proyecto al repositorio						

Alcance del proyecto:

El proyecto cumple con todas las características solicitadas, se lograron encontrar todos los modelos que se requerían para implementar, al igual que las animaciones realizadas fueron desarrolladas de manera correcta según lo solicitado, específicamente l que realiza la nave descendiendo de manera parabólica se tomaron en cuenta los conceptos de este efecto. Finalmente, el proyecto está dentro del repositorio listo para poderse clonar y probar sin errores.

Limitantes:

Para este proyecto los mayores obstáculos que se presentaron fueron a la hora de implementar modelos ya que estos podían ser de tamaños no soportables para el programa Visual Studio, sin embargo la mayoría de lo que se agrego era necesario para que ambiente virtual tuviera todos los elementos importantes que se deben

contar, un ejemplo de lo que digo es el modelo de mees3.obj dentro de la carpeta de Models el cual estaba planeado de anexar pero los límites que soporta el programa no lo permitieron, otra gran problemática fue al momento de anexar al repositorio pues este tiene ciertas restricciones que no permitían meter todo el proyecto por lo que fue necesario comprimir los archivos que tenían esa dificultad al subirse, para todo lo demás era cuestión de practica para recrear los modelos que tuviera que agregar y el código viene referenciado de las practicas vistas en clase por lo que era realizar los cambios requeridos para lograr el efecto deseado.

Documentación del código:

```cpp
//Marcelo Romero Adolfo
//Proyecto Final Computacion Grafica Gpo 08

#include <iostream>
#include <cmath>

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

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

// Other Libs
#include "stb_image.h"

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

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


// Other includes
#include "Shader.h"
#include "Camera.h"
#include "Model.h"
#include "Texture.h"


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

// Window dimensions
const GLuint WIDTH = 800, HEIGHT = 600;
int SCREEN_WIDTH, SCREEN_HEIGHT, i=0, j=0, k=0;

// Camera
Camera  camera(glm::vec3(0.0f, 50.0f, 550.0f));
GLfloat lastX = WIDTH / 2.0;
GLfloat lastY = HEIGHT / 2.0;
bool keys[1024];
bool firstMouse = true;
float range = 0.0f;
float rot = 0.0f;
```

Declaracion de librerias

Declaracion de variables

```cpp
// Light attributes
glm::vec3 lightPos(0.0f, 0.0f, 0.0f);
glm::vec3 PosIni(0.0f, 0.0f, 0.0f);
bool active;

//Animación para nave y robot
float movKitX = 0.0;
float movKitZ = 0.0;
float movKitY = 0.0;
float movKitX2 = 0.0;
float movKitY2 = 0.0;
float movKitZ2 = 0.0;
float rotKit = 90.0;
float rotKitX2 = 0.0;
float rotKitY2 = 0.0;
float rotKitZ2 = 0.0;
bool circuito = false;
bool circuito2 = 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 = false;
bool recorrido12 = false;
bool recorrido13 = false;

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

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

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

Declaracion de variables para el recorrido de modelos

Declaración de variables de control

```
    float rotRodIzq;
    float rotRodDer;
    float rotBraIzq;
    float rotBraDer;
    float BrazM;
    float BrazRob;
    float rotInc;
    float rotInc2;
    float rotInc3;
    float rotInc4;
    float rotInc5;
    float rotInc6;

}FRAME;

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

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

glm::vec3 LightP1;



void saveFrame(void)
{

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

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

    KeyFrame[FrameIndex].rotRodIzq = rotRodIzq;
    KeyFrame[FrameIndex].rotRodDer = rotRodDer;
    KeyFrame[FrameIndex].rotBraIzq = rotBraIzq;
    KeyFrame[FrameIndex].rotBraDer = rotBraDer;
    KeyFrame[FrameIndex].BrazM = BrazM;
    KeyFrame[FrameIndex].BrazRob = BrazRob;


    FrameIndex++;
}

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

    rotRodIzq = KeyFrame[0].rotRodIzq;
    rotRodDer = KeyFrame[0].rotRodDer;
    rotBraIzq = KeyFrame[0].rotBraIzq;
    rotBraDer = KeyFrame[0].rotBraDer;
    BrazM = KeyFrame[0].BrazM;
    BrazRob = KeyFrame[0].BrazRob;

}

void interpolation(void)
{

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

    KeyFrame[playIndex].rotInc = (KeyFrame[playIndex + 1].rotRodIzq - KeyFrame
      [playIndex].rotRodIzq) / i_max_steps;
    KeyFrame[playIndex].rotInc2 = (KeyFrame[playIndex + 1].rotRodDer - KeyFrame
      [playIndex].rotRodDer) / i_max_steps;
    KeyFrame[playIndex].rotInc3 = (KeyFrame[playIndex + 1].rotBraIzq - KeyFrame
      [playIndex].rotBraIzq) / i_max_steps;
    KeyFrame[playIndex].rotInc4 = (KeyFrame[playIndex + 1].rotBraDer - KeyFrame
      [playIndex].rotBraDer) / i_max_steps;
    KeyFrame[playIndex].rotInc5 = (KeyFrame[playIndex + 1].BrazM - KeyFrame
      [playIndex].BrazM) / i_max_steps;
    KeyFrame[playIndex].rotInc6 = (KeyFrame[playIndex + 1].BrazRob - KeyFrame
      [playIndex].BrazRob) / i_max_steps;

}



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, "ProyectoFinalLab",
      nullptr, nullptr);

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

        return EXIT_FAILURE;
    }

    glfwMakeContextCurrent(window);

    glfwGetFramebufferSize(window, &SCREEN_WIDTH, &SCREEN_HEIGHT);

    // Set the required callback functions
    glfwSetKeyCallback(window, KeyCallback);
    glfwSetCursorPosCallback(window, MouseCallback);
    printf("%f", glfwGetTime());

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

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

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

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

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

    Model CasaRyM((char*)"Models/CasaRyM/CasaRyM.obj");
    Model Rick((char*)"Models/Rick/CuerpoRick.obj");
    Model BrazoDeR((char*)"Models/Rick/BrazoDeR.obj");

    Model BrazoIzR((char*)"Models/Rick/BrazoIzR.obj");
    Model Nada((char*)"Models/Nada/Nada.obj");
    Model Portal((char*)"Models/Portal/Portal.obj");
    Model Alien((char*)"Models/Alien/Alien.obj");
    Model models[3] = {Nada,Alien,Portal};
    Model Robot((char*)"Models/Robot/CuerpRob.obj");
    Model BRobot((char*)"Models/Robot/BrazRob.obj");
    Model Nave((char*)"Models/Nave/Nave.obj");
    Model Boxm((char*)"Models/boxm/boxm.obj");
    Model CuerpoM((char*)"Models/mees/CuerpoM2.obj");
    Model modelsM[2] = { Nada,CuerpoM };
    Model HombroDM((char*)"Models/mees/HombDM.obj");
    Model BrazDM((char*)"Models/mees/BrazDM.obj");
    Model HombroIM((char*)"Models/mees/HombIM.obj");
    Model BrazIM((char*)"Models/mees/BrazIM.obj");
    Model M1((char*)"Models/mees/mees.obj");
    Model M2((char*)"Models/mees/mees2.obj");
    Model M3((char*)"Models/mees/mees.obj");//Deje el modelo "mees.obj"
      aporposito ya que agregar "mees3.obj" satura el programa
    Model models2[4] = { Nada,M3,M2,M1};


    //Inicialización de KeyFrames

    for (int i = 0; i < MAX_FRAMES; i++)
    {
        KeyFrame[i].posX = 0;
        KeyFrame[i].incX = 0;
        KeyFrame[i].incY = 0;
        KeyFrame[i].incZ = 0;
        KeyFrame[i].rotRodIzq = 0;
        KeyFrame[i].rotRodDer = 0;
        KeyFrame[i].rotBraIzq = 0;
        KeyFrame[i].rotBraDer = 0;
        KeyFrame[i].BrazM = 0;
        KeyFrame[i].BrazRob = 0;
        KeyFrame[i].rotInc = 0;
        KeyFrame[i].rotInc2 = 0;
        KeyFrame[i].rotInc3 = 0;
        KeyFrame[i].rotInc4 = 0;
        KeyFrame[i].rotInc5 = 0;
        KeyFrame[i].rotInc6 = 0;
    }



    // Set up vertex data (and buffer(s)) and attribute pointers
    GLfloat vertices[] =
    {
        // Positions           // Normals              // Texture Coords
        -0.5f, -0.5f, -0.5f,    0.0f,  0.0f, -1.0f,     0.0f,  0.0f,
```

Declaración de variables de control de modelos

Definiendo KeyFrames a sus respectivos modelos

Definiendo KeyFrames a sus respectivos modelos

Intengrando funcion de KeyFrames a sus respectivos modelos

Anexando shaders

Anexando Modelos desde su carpeta correspondiente

Inicializando KeyFrames

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

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

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

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

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

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


    GLfloat skyboxVertices[] = {
        // Positions
        -1.0f,  1.0f, -1.0f,
        -1.0f, -1.0f, -1.0f,
        1.0f, -1.0f, -1.0f,
        1.0f, -1.0f, -1.0f,
        1.0f,  1.0f, -1.0f,
        -1.0f,  1.0f, -1.0f,

        -1.0f, -1.0f,  1.0f,
        -1.0f, -1.0f, -1.0f,
        -1.0f,  1.0f, -1.0f,
        -1.0f,  1.0f, -1.0f,
        -1.0f,  1.0f,  1.0f,
        -1.0f, -1.0f,  1.0f,

        1.0f, -1.0f, -1.0f,
        1.0f, -1.0f,  1.0f,
        1.0f,  1.0f,  1.0f,
        1.0f,  1.0f,  1.0f,
        1.0f,  1.0f, -1.0f,
        1.0f, -1.0f, -1.0f,

        -1.0f, -1.0f,  1.0f,
        -1.0f,  1.0f,  1.0f,
        1.0f,  1.0f,  1.0f,
        1.0f,  1.0f,  1.0f,
        1.0f, -1.0f,  1.0f,
        -1.0f, -1.0f,  1.0f,

        -1.0f,  1.0f, -1.0f,
        1.0f,  1.0f, -1.0f,
        1.0f,  1.0f,  1.0f,
        1.0f,  1.0f,  1.0f,
        -1.0f,  1.0f,  1.0f,
        -1.0f,  1.0f, -1.0f,

        -1.0f, -1.0f, -1.0f,
        -1.0f, -1.0f,  1.0f,
        1.0f, -1.0f, -1.0f,
        1.0f, -1.0f, -1.0f,
        -1.0f, -1.0f,  1.0f,
        1.0f, -1.0f,  1.0f
    };


    GLuint indices[] =
    {  // Note that we start from 0!
        0,1,2,3,
        4,5,6,7,
        8,9,10,11,
        12,13,14,15,
        16,17,18,19,
        20,21,22,23,
        24,25,26,27,
        28,29,30,31,
        32,33,34,35
    };

    // Positions all containers
    glm::vec3 cubePositions[] = {
        glm::vec3(0.0f,  0.0f,  0.0f),
        glm::vec3(2.0f,  5.0f, -15.0f),
        glm::vec3(-1.5f, -2.2f, -2.5f),
        glm::vec3(-3.8f, -2.0f, -12.3f),
        glm::vec3(2.4f, -0.4f, -3.5f),
        glm::vec3(-1.7f,  3.0f, -7.5f),
        glm::vec3(1.3f, -2.0f, -2.5f),
        glm::vec3(1.5f,  2.0f, -2.5f),
        glm::vec3(1.5f,  0.2f, -1.5f),
        glm::vec3(-1.3f,  1.0f, -1.5f)
    };


    // First, set the container's VAO (and VBO)
    GLuint VBO, VAO, EBO;
    glGenVertexArrays(1, &VAO);
    glGenBuffers(1, &VBO);
    glGenBuffers(1, &EBO);

    glBindVertexArray(VAO);
    glBindBuffer(GL_ARRAY_BUFFER, VBO);
    glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW);

    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EBO);
    glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(indices), indices,
      GL_STATIC_DRAW);

    // Position attribute
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat),
      (GLvoid*)0);
    glEnableVertexAttribArray(0);
    // Normals attribute
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat),
      (GLvoid*)(3 * sizeof(GLfloat)));
    glEnableVertexAttribArray(1);
    // Texture Coordinate attribute
    glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat),
      (GLvoid*)(6 * sizeof(GLfloat)));
    glEnableVertexAttribArray(2);
    glBindVertexArray(0);

    // Then, we set the light's VAO (VBO stays the same. After all, the vertices
       are the same for the light object (also a 3D cube))
    GLuint lightVAO;
    glGenVertexArrays(1, &lightVAO);
    glBindVertexArray(lightVAO);
    // We only need to bind to the VBO (to link it with glVertexAttribPointer),
      no need to fill it; the VBO's data already contains all we need.
    glBindBuffer(GL_ARRAY_BUFFER, VBO);
    // Set the vertex attributes (only position data for the lamp))
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat),
    glEnableVertexAttribArray(0);
    glBindVertexArray(0);


    //SkyBox
    GLuint skyboxVBO, skyboxVAO;
    glGenVertexArrays(1, &skyboxVAO);
    glGenBuffers(1, &skyboxVBO);
    glBindVertexArray(skyboxVAO);
    glBindBuffer(GL_ARRAY_BUFFER, skyboxVBO);
    glBufferData(GL_ARRAY_BUFFER, sizeof(skyboxVertices), &skyboxVertices,
      GL_STATIC_DRAW);
    glEnableVertexAttribArray(0);
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(GLfloat),
      (GLvoid*)0);

    // Load textures
    vector<const GLchar*> faces;
    faces.push_back("SkyBox/gatosS.tga");
    faces.push_back("SkyBox/gatosS.tga");
    faces.push_back("SkyBox/gatosS.tga");
    faces.push_back("SkyBox/gatosS.tga");
    faces.push_back("SkyBox/gatosS.tga");
    faces.push_back("SkyBox/gatosS.tga");

    GLuint cubemapTexture = TextureLoading::LoadCubemap(faces);

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

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

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

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


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

Se mantuvo el modelado del cubo de las practicas por si se llegaba a ocupar

Anexando textura para el Skybox desde su carpeta

```
        // Use cooresponding shader when setting uniforms/drawing objects
        lightingShader.Use();
        GLint viewPosLoc = glGetUniformLocation(lightingShader.Program,
          "viewPos");
        glUniform3f(viewPosLoc, camera.GetPosition().x, camera.GetPosition().y,
          camera.GetPosition().z);
        // Set material properties
        glUniform1f(glGetUniformLocation(lightingShader.Program,
          "material.shininess"), 32.0f);
        // == ==========================
        // Here we set all the uniforms for the 5/6 types of lights we have. We
          have to set them manually and index
        // the proper PointLight struct in the array to set each uniform
          variable. This can be done more code-friendly
        // by defining light types as classes and set their values in there, or
          by using a more efficient uniform approach
        // by using 'Uniform buffer objects', but that is something we discuss
          in the 'Advanced GLSL' tutorial.
        // == ==========================
        // Directional light
        glUniform3f(glGetUniformLocation(lightingShader.Program,
          "dirLight.direction"), -0.2f, -1.0f, -0.3f);
        glUniform3f(glGetUniformLocation(lightingShader.Program,
          "dirLight.ambient"), 1.0f, 1.0f, 1.0f);
        glUniform3f(glGetUniformLocation(lightingShader.Program,
          "dirLight.diffuse"), 0.4f, 0.4f, 0.4f);
        glUniform3f(glGetUniformLocation(lightingShader.Program,
          "dirLight.specular"), 0.5f, 0.5f, 0.5f);


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


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

        // Point light 4
        glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights
          [3].position"), pointLightPositions[3].x, pointLightPositions[3].y,
          pointLightPositions[3].z);
        glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights
          [3].ambient"), 0.05f, 0.05f, 0.05f);
        glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights
          [3].diffuse"), 1.0f, 0.0f, 1.0f);
        glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights
          [3].specular"), 1.0f, 0.0f, 1.0f);
        glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights
          [3].constant"), 1.0f);
        glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights
          [3].linear"), 0.09f);
        glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights
          [3].quadratic"), 0.032f);

        // SpotLight
        glUniform3f(glGetUniformLocation(lightingShader.Program,
          "spotLight.position"), camera.GetPosition().x, camera.GetPosition().y,
          camera.GetPosition().z);
        glUniform3f(glGetUniformLocation(lightingShader.Program,
          "spotLight.direction"), camera.GetFront().x, camera.GetFront().y,
          camera.GetFront().z);
        glUniform3f(glGetUniformLocation(lightingShader.Program,
          "spotLight.ambient"), 0.0f, 0.0f, 0.0f);
        glUniform3f(glGetUniformLocation(lightingShader.Program,
          "spotLight.diffuse"), 0.0f, 0.0f, 0.0f);
        glUniform3f(glGetUniformLocation(lightingShader.Program,
          "spotLight.specular"), 0.0f, 0.0f, 0.0f);
        glUniform1f(glGetUniformLocation(lightingShader.Program,
          "spotLight.constant"), 1.0f);
        glUniform1f(glGetUniformLocation(lightingShader.Program,
          "spotLight.linear"), 0.09f);
        glUniform1f(glGetUniformLocation(lightingShader.Program,
          "spotLight.quadratic"), 0.032f);
        glUniform1f(glGetUniformLocation(lightingShader.Program,
          "spotLight.cutOff"), glm::cos(glm::radians(12.5f)));
        glUniform1f(glGetUniformLocation(lightingShader.Program,
          "spotLight.outerCutOff"), glm::cos(glm::radians(15.0f)));

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

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


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

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


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


        //Carga de modelo
        //Casa de Rick y Morty
        view = camera.GetViewMatrix();
        glm::mat4 model(1);
        glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
        CasaRyM.Draw(lightingShader);

        //Rick
        view = camera.GetViewMatrix();
        tmp = model = glm::translate(model, glm::vec3(-150, 40, 380));
        glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
        Rick.Draw(lightingShader);

        //BrazoIzq
        view = camera.GetViewMatrix();
        tmp = model = glm::translate(model, glm::vec3(3, 11.5, 1.5));
        //model = glm::translate(model, glm::vec3(posX, posY, posZ));
        //model = glm::rotate(model, glm::radians(rot), glm::vec3(0.0f, 1.0f,
          0.0));
        model = glm::rotate(model, glm::radians(-rotBraDer), glm::vec3(1.0f,
          0.0f, 0.0f));
        glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
        BrazoIzR.Draw(lightingShader);


        //BrazoDer
        view = camera.GetViewMatrix();
        model = glm::translate(tmp, glm::vec3(-7, 0, 0));
        model = glm::rotate(model, glm::radians(-45.0f), glm::vec3(1.0f, 0.0f,
          0.0f));
        model = glm::rotate(model, glm::radians(-rotBraIzq), glm::vec3(1.0f,
          0.0f, 0.0f));
        glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
        BrazoDeR.Draw(lightingShader);

        // 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
        shader.Use();
        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
        //Portal y Alien
        model = glm::rotate(model, glm::radians(45.0f), glm::vec3(1.0f, 0.0f,
          0.0f));
        model = glm::translate(model, glm::vec3(-10, 5, 100)); // Translate it
          down a bit so it's at the center of the scene
        model = glm::scale(model, glm::vec3(0.5f, 0.5f, 0.5f)); // It's a bit
          too big for our scene, so scale it down
```

Se mantuvo la configuración de iluminación de las practicas

Carga del modelo de la casa

Carga del modelo de Rick

Carga de modelos de los brazos Rick

Carga de modelos: portal y alíen

```
        model = glm::rotate(model, glm::radians(180.0f), glm::vec3(0.0f, 1.0f,
          0.0f));
        glUniformMatrix4fv(glGetUniformLocation(shader.Program, "model"), 1,
          GL_FALSE, glm::value_ptr(model));
        models[1].Draw(shader);

        //Robot de la mantequilla
        view = camera.GetViewMatrix();
        model = glm::mat4(1);
        model = glm::translate(model, glm::vec3(-22, 43.4, 150));
        model = glm::rotate(model, glm::radians(90.0f), glm::vec3(0.0f, 1.0f,
          0.0f));
        model = glm::translate(model, PosIni + glm::vec3(movKitX, 0, movKitZ));
        model = glm::rotate(model, glm::radians(rotKit), glm::vec3(0.0f, 1.0f,
          0.0));
        glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
        Robot.Draw(lightingShader);

        //Brazos de robot
        view = camera.GetViewMatrix();
        model = glm::mat4(1);
        model = glm::translate(model, glm::vec3(-22, 47.2, 150.5));
        model = glm::rotate(model, glm::radians(90.0f), glm::vec3(0.0f, 1.0f,
          0.0f));
        model = glm::translate(model, PosIni + glm::vec3(movKitX, 0, movKitZ));
        model = glm::rotate(model, glm::radians(rotKit), glm::vec3(0.0f, 1.0f,
          0.0));
        model = glm::rotate(model, glm::radians(-BrazRob), glm::vec3(1.0f, 0.0f,
          0.0f));
        glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
        BRobot.Draw(lightingShader);

        //Nave
        view = camera.GetViewMatrix();
        model = glm::mat4(1);
        model = glm::translate(model, glm::vec3(-250, 60, 450));
        model = glm::rotate(model, glm::radians(90.0f), glm::vec3(0.0f, 1.0f,
          0.0f));
        model = glm::translate(model, PosIni + glm::vec3(movKitX2, movKitY2,
          movKitZ2));
        model = glm::rotate(model, glm::radians(rotKitX2), glm::vec3(0.0f, 1.0f,
          0.0));
        model = glm::rotate(model, glm::radians(rotKitY2), glm::vec3(1.0f, 0.0f,
          0.0));
        model = glm::rotate(model, glm::radians(rotKitZ2), glm::vec3(0.0f, 0.0f,
          1.0));
        glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
        Nave.Draw(lightingShader);

        // Caja de Meeseeks
        view = camera.GetViewMatrix();
        model = glm::mat4(1);
        model = glm::translate(model, glm::vec3(-248, 56.5, 50));
        glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
        Boxm.Draw(lightingShader);

        //Meeseeks
        view = camera.GetViewMatrix();
        model = glm::mat4(1);
        model = glm::translate(model, glm::vec3(-250, 48.1, 70));
        model = glm::rotate(model, glm::radians(180.0f), glm::vec3(0.0f, 1.0f,
          0.0f));
        glUniformMatrix4fv(glGetUniformLocation(shader.Program, "model"), 1,
          GL_FALSE, glm::value_ptr(model));
        modelsM[k].Draw(shader);

        view = camera.GetViewMatrix();
        tmp = model = glm::translate(model, glm::vec3(-11.2, 12, 12.2));
        model = glm::rotate(model, glm::radians(-BrazM), glm::vec3(0.0f, 0.0f,
          1.0f));
        glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
        BrazDM.Draw(lightingShader);

        view = camera.GetViewMatrix();
        model = glm::mat4(1);
        model = glm::translate(model, glm::vec3(-250, 48.1, 30)); // 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));
        models2[j].Draw(shader);

        glBindVertexArray(0);

        // Draw skybox as last
        glDepthFunc(GL_LEQUAL);  // Change depth function so depth test passes
          when values are equal to depth buffer's content
        SkyBoxshader.Use();
        view = glm::mat4(glm::mat3(camera.GetViewMatrix()));    // Remove any
          translation component of the view matrix
        glUniformMatrix4fv(glGetUniformLocation(SkyBoxshader.Program, "view"),
          1, GL_FALSE, glm::value_ptr(view));
        glUniformMatrix4fv(glGetUniformLocation(SkyBoxshader.Program,
          "projection"), 1, GL_FALSE, glm::value_ptr(projection));

        // skybox cube
        glBindVertexArray(skyboxVAO);
        glActiveTexture(GL_TEXTURE1);
        glBindTexture(GL_TEXTURE_CUBE_MAP, cubemapTexture);
        glDrawArrays(GL_TRIANGLES, 0, 36);
        glBindVertexArray(0);
        glDepthFunc(GL_LESS); // Set depth function back to default


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


    glDeleteVertexArrays(1, &VAO);
    glDeleteVertexArrays(1, &lightVAO);
    glDeleteBuffers(1, &VBO);
    glDeleteBuffers(1, &EBO);
    glDeleteVertexArrays(1, &skyboxVAO);
    glDeleteBuffers(1, &skyboxVBO);
    // Terminate GLFW, clearing any resources allocated by GLFW.
    glfwTerminate();



    return 0;
}


void animacion()
{

    //Movimiento del personaje

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

            rotRodIzq += KeyFrame[playIndex].rotInc;
            rotRodDer += KeyFrame[playIndex].rotInc2;
            rotBraIzq += KeyFrame[playIndex].rotInc3;
            rotBraDer += KeyFrame[playIndex].rotInc4;
            BrazM += KeyFrame[playIndex].rotInc5;
            BrazRob += KeyFrame[playIndex].rotInc6;

            i_curr_steps++;
        }

    }
    //recorridos
    if (circuito)
    {
        if (recorrido1)
        {
            movKitX += 0.1f;
            if (movKitX > 30)
            {
                recorrido1 = false;
                recorrido2 = true;
            }
        }
        if (recorrido2)
        {
            rotKit = 320;
            movKitX -= 0.1f;
            movKitZ += 0.15f;
            if (movKitX < 0 && movKitZ > 47)
            {
                recorrido2 = false;
                recorrido3 = true;

            }
        }

        if (recorrido3)
        {
            rotKit = 90;
            movKitX += 0.1f;
            if (movKitX > 30)
            {
                recorrido3 = false;
                recorrido4 = true;
            }
        }
        if (recorrido4)
        {
            rotKit = 220;
            movKitX -= 0.1f;
            movKitZ -= 0.15f;
            if (movKitX < 0 && movKitZ > 0)
```

Carga de modelos: robot y sus brazos

Carga de modelos: Nave

Carga de modelos: Meeseeks y su caja

Dibujando KeyFrames de los respectivos modelos

Planteamiento de los recorridos del robot

```
            {
                recorrido4 = false;
                recorrido5 = true;
            }
        }
        if (recorrido5)
        {
            rotKit = 90;
            movKitX += 0.1f;
            if (movKitX > 0)
            {
                recorrido5 = false;
                recorrido1 = true;
            }
        }

    }
    if (circuito2)
    {
        if (recorrido6)
        {
            movKitY2 += 1;
            if (movKitY2 > 50)
            {
                recorrido6 = false;
                recorrido7 = true;
            }
        }
        if (recorrido7)
        {
            rotKitY2 = 330;
            movKitY2 += 1;
            movKitZ2 += 1;
            if (movKitZ2 > 200 && movKitY2 > 200)
            {
                recorrido7 = false;
                recorrido8 = true;

            }
        }

        if (recorrido8)
        {
            rotKitY2 = 360;
            rotKitX2 = 90;
            movKitX2 += 1;
            if (movKitX2 > 200)
            {
                recorrido8 = false;
                recorrido9 = true;
            }
        }

        if (recorrido9)
        {
            rotKitX2 = 45;
            movKitX2 += 1;
            movKitZ2 += 1;
            if (movKitX2 > 500  && movKitZ2 > 500)
            {
                recorrido9 = false;
                recorrido10 = true;
            }
        }
        if (recorrido10)
        {
            rotKitX2 = 180;
            movKitZ2 -= 1;
            if (movKitZ2 < 0)
            {
                recorrido10 = false;
                recorrido11 = true;
            }
        }
        if (recorrido11)
        {
            rotKitX2 = 270;
            movKitX2 -= 8;
            movKitY2 -= 1;
            if (movKitX2 < 200 && movKitY2 < 200)
            {
                recorrido11 = false;
                recorrido12 = true;
            }
        }
        if (recorrido12)
        {
            movKitX2 -= 2;
            movKitY2 -= 4;
            if (movKitX2 < 0 && movKitY2 < 0)
            {
                recorrido12 = false;
                recorrido13 = true;
            }
        }
        if (recorrido13)
        {
            rotKitX2 = 0;
            if (movKitX2 < 0 && movKitY2 < 0)
            {
                recorrido13 = false;
                recorrido6 = true;
            }
        }

```

Planteamiento de los recorridos de la nave

```


    }
}


// 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_T == key && GLFW_PRESS == action) {
        if (i == 0) {
            i = 2;
        }
        else {
            i -= 1;
        }
    }
    if (GLFW_KEY_M == key && GLFW_PRESS == action) {
        if (j == 0) {
            j = 3;
        }
        else {
            j -= 1;
        }
    }
    if (GLFW_KEY_N == key && GLFW_PRESS == action) {
        if (k == 0) {
            k = 1;
        }
        else {
            k -= 1;
        }
    }

    if (keys[GLFW_KEY_L])
    {
        if (play == false && (FrameIndex > 1))
        {

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

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

    }

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

    }


    if (GLFW_KEY_ESCAPE == key && GLFW_PRESS == action)
    {
        glfwSetWindowShouldClose(window, GL_TRUE);
    }

    if (key >= 0 && key < 1024)
    {
        if (action == GLFW_PRESS)
        {
            keys[key] = true;
        }
        else if (action == GLFW_RELEASE)
        {
            keys[key] = false;
        }
    }

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

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

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

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

Asignando teclas para acciona las animaciones de transición

Asignando teclas para acciona las animaciones por KeyFrame

```
        bottom to left

    lastX = xPos;
    lastY = yPos;

    camera.ProcessMouseMovement(xOffset, yOffset);
}

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

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

    }

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

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

    }

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

    }

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

    }

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

    }

    if (keys[GLFW_KEY_8])
    {
        if (BrazRob < 60.0f)
            BrazRob += 1.0f;

    }

    if (keys[GLFW_KEY_9])
    {
        if (BrazRob > -15)
            BrazRob -= 1.0f;

    }

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

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

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

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

    //CIRCUITOS
    if (keys[GLFW_KEY_U])
    {
        circuito = true;
    }

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

    if (keys[GLFW_KEY_O])
    {
        circuito2 = true;
    }

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


    // Camera controls
    if (keys[GLFW_KEY_W] || keys[GLFW_KEY_UP])
    {
        camera.ProcessKeyboard(FORWARD, deltaTime+=0.5);

    }

    if (keys[GLFW_KEY_S] || keys[GLFW_KEY_DOWN])
    {
        camera.ProcessKeyboard(BACKWARD, deltaTime+=0.5);


    }

    if (keys[GLFW_KEY_A] || keys[GLFW_KEY_LEFT])
    {
        camera.ProcessKeyboard(LEFT, deltaTime+=0.5);


    }

    if (keys[GLFW_KEY_D] || keys[GLFW_KEY_RIGHT])
    {
        camera.ProcessKeyboard(RIGHT, deltaTime+=0.5);
    }


}
```

Asignando teclas de movimiento en animaciones por KeyFrame

Asignando teclas para accionar las animaciones por recorridos

Teclas de control para el manejo de la cámara