MANUAL TÉCNICO - PROYECTO FINAL

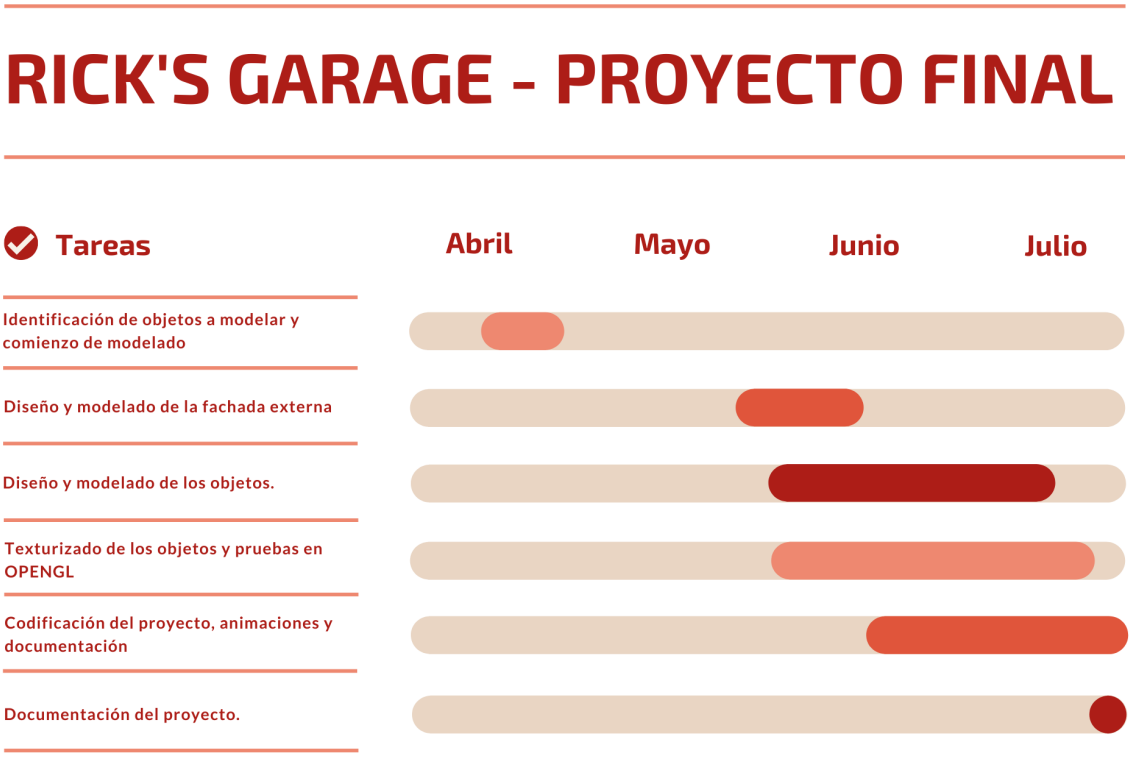
Rick's Garage

LABORATORIO DE COMPUTACIÓN GRÁFICA E INTERACCIÓN HUMANO-COMPUTADORA.
Profesor: Ing. Carlos Aldair Román Balbuena
Alumno: Silva Núñez Alejandro Bryan
Semestre 2021-2 - Grupo: 09
Entrega: 28/07/2021
MANUAL TÉCNICO - PROYECTO FINAL

Objetivos.

Recrear la fachada de la casa de Morty Smith y Rick Sanchez, de la serie animada "Rick & Morty". Modelar el interior del garaje/laboratorio que aparece en la serie y cumplir con los objetivos del proyecto propuesto por el profesor, el Ing. Carlos Aldair Román Balbuena.

Diagrama de Gantt.



Para este proyecto se identificaron 6 tareas relevantes y principales para su finalización. Estas fueron: Identificación de objetos a modelar y comienzo de modelado, diseño y

modelado de la fachada externa, diseño y modelado de los objetos, texturizado de los objetos y pruebas en OPENGL, codificación del proyecto, animaciones y documentación, y, finalmente, documentación del proyecto.
Como se puede observar, la documentación fue la tarea que menos tiempo tomó, en comparación con las demás. En contraste, el diseño y modelado de los objetos, así como su texturizado y la codificación del proyecto fueron las tareas más tardadas (aproximadamente 1 mes o 4 semanas para completarlas con éxito).
El diseño de la fachada no fue una de las tareas más rápidas, pero tampoco considero que se haya requerido demasiado (aproximadamente 1 semana).

Alcance del proyecto.

El alcance de este proyecto consiste en:

- **Diseño y modelado:** Diseñar y modelar el interior y exterior del garage de “Rick & Morty” (objetos definidos al inicio de este proyecto, fachada y extras), así como sólo el exterior correspondiente al resto de la casa y ambientación (puertas, ventanas, automóvil, pasto, jardineras, vegetación, etc).
- **Luz:** Identificar las fuentes de luz más importantes dentro de la escena y replicarlas utilizando lo aprendido en clase, utilizando sus componentes difusas, ambientales y especulares.
- **Textura:** Crear los modelos de los objetos y texturizarlos, con base en los objetos de la serie ya mencionada.
- **Animación:** Crear dos animaciones utilizando keyframes, y 3 animaciones básicas (sin perder el contexto de la escena)
- **Cielo:** Hacer uso de un skybox para replicar la sensación de ambiente amplio.
- **Extras:** Agregar una animación por hueso/esqueleto.

Limitantes.

Considero que la única limitante para la realización de este proyecto fue el aprendizaje del uso de herramientas de modelado 3D, asi como la texturización de los mismos y la mejora al momento de las implementaciones.
Sin embargo, no se generaron limitantes ni de software, de hardware, de tiempo, o de conocimientos, ya que los temas tratados en las prácticas del laboratorio fueron los utilizados en este proyecto.

Documentación del código.

El código principal del proyecto se desarrolla en el archivo RicksGarage/RicksGarage/main.cpp

Bibliotecas utilizadas: En este proyecto se están utilizando las bibliotecas iostream, cmath, glew, glfw3, stb_image, glm, matrix_transform, type_ptr, SOIL2, shader.h, camera.h, model.h, texture.h, modelAnim.h, como se puede observar en la imagen 1.

```cpp
#include <iostream>
#include <cmath>

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

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

// Other Libs
#include "stb_image.h"
```

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

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

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

Funciones utilizadas: En este proyecto se utilizaron las siguientes funciones:

```cpp
// 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();
void animacionCarro();
void animacionGarage();
void animacionSilla();
```

- **KeyCallBack** (encargada de verificar si es que se ha presionado alguna tecla durante la ejecución del programa, mediante GLFW). Verifica a las teclas L y K (encargadas de la animación por keyframes), así como las teclas espacio y F (dedicadas al manejo de la luz) y escape (para terminar la ejecución del programa)

```
// Is called whenever a key is pressed/released via GLFW
void KeyCallback(GLFWwindow *window, int key, int scancode, int action, int mode)
{
    // Saves all the keyframes
    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;
        }
    }
    // Plays keyframe animations
    if (keys[GLFW_KEY_K])
    {
        if (FrameIndex<MAX_FRAMES)
        {
            saveFrame();
        }
    }
```

```
    // Closes principal window
    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;
        }
    }
    // Alternates between lights
    if (keys[GLFW_KEY_SPACE])
    {
        active = !active;
        if (active) {
            LightP1 = glm::vec3(0.5f, 0.0f, 0.0f);
            LightP2 = glm::vec3(0.0f, 0.0f, 0.0f);
            LightP3 = glm::vec3(0.0f, 0.0f, 0.0f);
        }
        else {
            LightP1 = glm::vec3(0.0f, 0.0f, 0.0f);
            LightP2 = glm::vec3(0.7f, 0.7f, 0.7f);
            LightP3 = glm::vec3(0.7f, 0.7f, 0.7f);
        }
    }
    //Turn all the lights off
    if (keys[GLFW_KEY_F]) {
        LightP1 = glm::vec3(0.0f, 0.0f, 0.0f);
        LightP2 = glm::vec3(0.0f, 0.0f, 0.0f);
        LightP3 = glm::vec3(0.0f, 0.0f, 0.0f);
    }
}
```

- **MouseCallBack** (implementación del movimiento dirigido por el mouse/ratón).

```
// Is called whenever the mouse is moved
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);
}
```

- **DoMovement** (cuya finalidad es comenzar los movimientos de las puertas/animaciones básicas de los objetos).

```
// Moves/alters the camera positions based on user input
void DoMovement()
{
    // Opens garage
    if (keys[GLFW_KEY_1]){
        if (rotGarage > -90.0f) {
            rotGarage -= 1.0f;
        }
    }
    // Closes garage
    if (keys[GLFW_KEY_2]){
        if (rotGarage < 0.0f) {
            rotGarage += 1.0f;
        }
    }
    // Opens garage door
    if (keys[GLFW_KEY_3]){
        if (rotGarageDoor < 90.0f) {
            rotGarageDoor += 1.0f;
        }
    }
    // Closes garage door
    if (keys[GLFW_KEY_4]){
        if (rotGarageDoor > 0.0f) {
            rotGarageDoor -= 1.0f;
        }
    }
    // Opens house door
    if (keys[GLFW_KEY_5]){
        if (rotHouseDoor > -100.0f) {
            rotHouseDoor -= 1.0f;
        }
    }
    // Closes house door
    if (keys[GLFW_KEY_6]){
        if (rotHouseDoor < -5.0f) {
            rotHouseDoor += 1.0f;
        }
    }
```

```
// Starts car animation
    if (keys[GLFW_KEY_7]){
        circuitoCar = true;
    }
    // Stops car animation
    if (keys[GLFW_KEY_8]){
        circuitoCar = false;
    }
    // Starts garage animation
    if (keys[GLFW_KEY_9]){
        garageAnim = true;
    }
    // Pauses garage animation
    if (keys[GLFW_KEY_0]){
        garageAnim = false;
    }
    // Starts chair animation
    if (keys[GLFW_KEY_O]) {
        sillaAnim = true;
    }
    // Pauses chair animations
    if (keys[GLFW_KEY_P]) {
        sillaAnim = false;
    }
    // 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);
    }
}
```

- **Animacion** (comienza una vez que KeyCallBack reconoce que se ha presionado la tecla “K”. Se encarga de realizar el movimiento por keyframes verificando la estructura de los keyframes hasta que se hayan recorrido todos, por medio de una interpolación).

```
void animacion()
{
//Movimiento por keyframes
    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
            movChairX += KeyFrame[playIndex].incChairX;
            movChairZ += KeyFrame[playIndex].incChairZ;
            rotCh += KeyFrame[playIndex].rotIncCh;
            rotPortal += KeyFrame[playIndex].incPortal;
            i_curr_steps++;
        }
    }
}
```

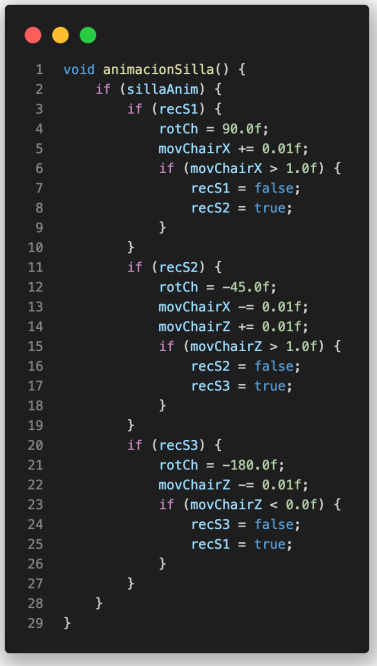
- **AnimacionCarro** (Es una función que define los límites de movimiento entre cada recorrido de la animación básica del automóvil, así como su rotación).

```
void animacionCarro() {
    if (circuitoCar) {
        if (rec1) {
            rotCar = 0.0f;
            movCarZ -= 0.1f;
            if (movCarZ < -40.0f) {
                rec1 = false;
                rec2 = true;
            }
        }
        if (rec2) {
            rotCar = -90.0f;
            movCarX += 0.1;
            if (movCarX > 7.0f) {
                rec2 = false;
                rec3 = true;
            }
        }
        if (rec3) {
            rotCar = -180.0f;
            movCarZ += 0.1f;
            if (movCarZ > 0.0f) {
                rec3 = false;
                rec4 = true;
            }
        }
        if (rec4) {
            rotCar = 90.0f;
            movCarX -= 0.1f;
            if (movCarX < 0.0f) {
                rec4 = false;
                rec1 = true;
            }
        }
    }
}
```

- **AnimacionGarage** (Es una función que define los límites de movimiento entre cada recorrido de la animación básica del portón del garaje, así como su rotación).

```
void animacionGarage() {
    if (garageAnim) {
        if (recG1) {
            rotGarage -= 0.5f;
            if (rotGarage < -90.0f) {
                recG1 = false;
                recG2 = true;
            }
        }
        if (recG2) {
            rotGarage += 0.5f;
            if (rotGarage > 0.0f) {
                recG2 = false;
                recG1 = true;
            }
        }
    }
}
```

- **AnimacionSilla** (Es una función que define los límites de movimiento entre cada recorrido de la animación básica de la silla, así como su rotación).



```
void animacionSilla() {
    if (sillaAnim) {
        if (recS1) {
            rotCh = 90.0f;
            movChairX += 0.01f;
            if (movChairX > 1.0f) {
                recS1 = false;
                recS2 = true;
            }
        }
        if (recS2) {
            rotCh = -45.0f;
            movChairX -= 0.01f;
            movChairZ += 0.01f;
            if (movChairZ > 1.0f) {
                recS2 = false;
                recS3 = true;
            }
        }
        if (recS3) {
            rotCh = -180.0f;
            movChairZ -= 0.01f;
            if (movChairZ < 0.0f) {
                recS3 = false;
                recS1 = true;
            }
        }
    }
}
```

Variables globales:

- camara (es un vector de tres componentes que se encarga de definir la posición inicial de la cámara en la escena).
- lastX y lastY (Variables GLFloat, encargadas de definir la última posición del mouse).
- keys[1024] (arreglo de booleanos, utilizada para cerrar la ventana con la tecla scape).
- firstMouse (booleano necesario para iniciar el movimiento del mouse).
- rotGarage, rotGarageDoor, rotHouseDoor (floats cuya finalidad es controlar la rotación de las puertas del garage y de la casa).
- active (booleano que verifica si la luz de la lampara roja está activa, e intercala con la luz blana en la función KeyCallBack)
- PosIniCar y PosIniChair (vectores de 3 componentes encargados de ubicar la posición inicial del automóvil y la silla).
- circuitoCar, garageAnim, sillaAnim (booleanos con la finalidad de controlar la animación de cada uno de los tres objetos).
- rec1, rec2, rec3, rec4, recG1, recG2, recS1, recS2, recS3 (booleanos que controlan el paso de los recorridos para las animaciones básicas).

- deltaTime y lastTime (GLFloats, cuyo valor define el tiempo entre frames de la ventana abierta).
- movChairX, movChairZ, rotCh, rotIncCh, rotPortal, incPortal (floats que definen el movimiento de la silla para su animación básica y por keyframes, así como la animación del portal port keyframes.

```
// Window dimensions
const GLuint WIDTH = 800, HEIGHT = 600;
int SCREEN_WIDTH, SCREEN_HEIGHT;
// Camera
Camera camera(glm::vec3(3.0f, 3.0f, 6.0f));
GLfloat lastX = WIDTH / 2.0;
GLfloat lastY = HEIGHT / 2.0;
bool keys[1024];
bool firstMouse = true;
float rotGarage = 0.0f, rotGarageDoor = 0.0f, rotHouseDoor = -100.0;
// Light attributes
bool active=true;
//Animation Variables
glm::vec3 PosIniCar(16.0f, 1.5f, 3.7f);
glm::vec3 PosIniChair(-0.5f, 1.1f, 1.0f);
bool circuitoCar = false, garageAnim = false, sillaAnim = false;
bool rec1 = true, rec2 = false, rec3 = false, rec4 = false;
bool recG1 = true, recG2 = false;
bool recS1 = true, recS2 = false, recS3 = false;
float movCarX = 0.0f, movCarZ = 0.0f, rotCar = 0.0f;
// Deltatime
GLfloat deltaTime = 0.0f;   // Time between current frame and last frame
GLfloat lastFrame = 0.0f;   // Time of last frame
// Keyframes
float movChairX = PosIniChair.x, movChairZ = PosIniChair.z, rotCh = 0.0f, rotIncCh = 0.0f, rotPortal = 0.0f, incPortal = 0.0;
```

Inicialización de la estructura de KeyFrames y variables:

- Elementos necesarios para realizar la animación por keyframes.

```
#define MAX_FRAMES 9
int i_max_steps = 190;
int i_curr_steps = 0;
typedef struct _frame
{
    //Variables para GUARDAR Key Frames
    float movChairX;        //Variable para PosicionX
    float movChairZ;        //Variable para PosicionZ
    float incChairX;        //Variable para IncrementoX
    float incChairZ;        //Variable para IncrementoZ
    float rotCh;            //Rotaci?n silla
    float rotIncCh;         //Incremento rotaci?n silla
    float rotPortal;        //Rotaci?n portal
    float incPortal;        //Incremento rotaci?n portal

}FRAME;

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

- Guardado de los keyframes de forma estática.

```
void saveFrame(void)
{
    printf("KeyFrame 0 guardado\n");
    //Keyframes Posici@n
    KeyFrame[0].movChairX = PosIniChair.x;
    KeyFrame[0].movChairZ = PosIniChair.z;
    //KeyFrames Rotaci@n
    KeyFrame[0].rotCh = rotCh;
    KeyFrame[0].rotPortal = -90;
    FrameIndex++;
    printf("KeyFrame 1 guardado\n");
    //Keyframes Posici@n
    KeyFrame[1].movChairX = 1.0f;
    KeyFrame[1].movChairZ = 0.0f;
    //KeyFrames Rotaci@n
    KeyFrame[1].rotCh = 180;
    KeyFrame[1].rotPortal = 180;
    FrameIndex++;
    printf("KeyFrame 2 guardado\n");
    //Keyframes Posici@n
    KeyFrame[2].movChairX = -1.0f;
    KeyFrame[2].movChairZ = 1.5f;
    //KeyFrames Rotaci@n
    KeyFrame[2].rotCh = -180;
    KeyFrame[2].rotPortal = -180;
    FrameIndex++;
```

```
    printf("KeyFrame 3 guardado\n");
    //Keyframes Posici@n
    KeyFrame[3].movChairX = PosIniChair.x;
    KeyFrame[3].movChairZ = PosIniChair.z;
    //KeyFrames Rotaci@n
    KeyFrame[3].rotCh = rotCh;
    KeyFrame[3].rotPortal = rotPortal;
    FrameIndex++;
    printf("KeyFrame 4 guardado\n");
    //Keyframes Posici@n
    KeyFrame[4].movChairX = 1.0f;
    KeyFrame[4].movChairZ = 0.0f;
    //KeyFrames Rotaci@n
    KeyFrame[4].rotCh = 180;
    KeyFrame[4].rotPortal = 180;
    FrameIndex++;
    printf("KeyFrame 5 guardado\n");
    //Keyframes Posici@n
    KeyFrame[5].movChairX = -1.0f;
    KeyFrame[5].movChairZ = 1.5f;
    //KeyFrames Rotaci@n
    KeyFrame[5].rotCh = -180;
    KeyFrame[5].rotPortal = -180;
    //FrameIndex++;
}
```

- Reseteo de los keyframes al inicio para no perder la posición inicial.

```
void resetElements(void)
{
    movChairX= KeyFrame[0].movChairX;
    movChairZ = KeyFrame[0].movChairZ;

    rotCh = KeyFrame[0].rotCh;
    rotPortal = KeyFrame[0].rotPortal;

}
```

- Interpolación de los keyframes, haciendo uso de la resta de sus incrementos del keyframe siguiente menos el actual, entre el número de pasos máximos.

```
void interpolation(void)
{

    KeyFrame[playIndex].incChairX = (KeyFrame[playIndex + 1].movChairX - KeyFrame[playIndex].movChairX) / i_max_steps;
    KeyFrame[playIndex].incChairZ = (KeyFrame[playIndex + 1].movChairZ - KeyFrame[playIndex].movChairZ) / i_max_steps;

    KeyFrame[playIndex].rotIncCh = (KeyFrame[playIndex + 1].rotCh - KeyFrame[playIndex].rotCh) / i_max_steps;
    KeyFrame[playIndex].incPortal = (KeyFrame[playIndex + 1].rotPortal - KeyFrame[playIndex].rotPortal) / i_max_steps;

}
```

Función Main:

- Se establecen las configuraciones principales de la ventana.

```cpp
int main()
{
    // Init GLFW
    glfwInit();

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

- Se importan los shaders y todos los objetos a utilizar.

```cpp
//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");

//Cargando los modelos
//FACHADA
Model Fachada((char*)"Models/Fachada/fachada.obj");
Model Zaguan((char*)"Models/Fachada/garagedoor.obj");
Model PuertaGarage((char*)"Models/Fachada/doortogarage.obj");
Model PuertaCasa((char*)"Models/Fachada/housedoor.obj");
//INTERIOR
Model Escritorio((char*)"Models/Interior/desktop.obj");
Model Lavadora((char*)"Models/Interior/washer.obj");
Model Pizarron((char*)"Models/Interior/board.obj");
Model Pizarron2((char*)"Models/Interior/board2.obj");
Model Reloj((char*)"Models/Interior/clock.obj");
Model Extinguidor((char*)"Models/Interior/extinguisher.obj");
Model Repisa((char*)"Models/Interior/shelf.obj");
Model Tuboensayo((char*)"Models/Interior/testtube.obj");
Model Extras((char*)"Models/Interior/extras.obj");
//PERSONAJES
Model Morty((char*)"Models/Personaje/morty.obj");
//EXTRAS
Model Carro((char*)"Models/Extra/car.obj");
Model Label((char*)"Models/Extra/label.obj");
Model Silla((char*)"Models/Extra/chair.obj");
Model Portal((char*)"Models/Extra/portal.obj");

//MODELO ANIMADO
ModelAnim animacionPersonaje("Animaciones/RickAnimado/macarena.dae");
animacionPersonaje.initShaders(animShader.Program);

```

- Se inicializan los keyframes con ceros.

```
//Inicializaci?n de KeyFrames
for(int i=0; i<MAX_FRAMES; i++)
{
    KeyFrame[i].movChairX = 0;
    KeyFrame[i].movChairZ = 0;
    KeyFrame[i].incChairX = 0;
    KeyFrame[i].incChairZ = 0;
    KeyFrame[i].rotCh = 0;
    KeyFrame[i].rotIncCh = 0;
    KeyFrame[i].rotPortal= 0;
    KeyFrame[i].incPortal = 0;

}
```

- Se definen los vertices de un cubo (utilizado sólo para darle ubicación a las luces)

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

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

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

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

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

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

- Se definen los vertices del skybox y se definen los indices.

```
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
    };
```

- Se posicionan los contenedores

```
// 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)
};
```

- Se definen el VBO, el VAO y el EBO generales.

```
// 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);
```

- Se establece el lightVAO para utilizar luces.

```
// 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);

```

- Se crea el skybox, se importan las caras y se cargan como su textura.

```
//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/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);
```

- Se define la proyección en perspectiva.

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

- Se comienza a usar el lightingShader para crear las luces del programa, se inicializan y se les asigna sus componentes ambiental, difusa, especular, así como sus constantes necesarias para crearlas, y sus posiciones.

```
// 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);
// Setting lights
// Directional light
glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.direction"), -0.2f, -1.0f, -0.3f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.ambient"), 0.3f, 0.3f, 0.3f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.diffuse"), 0.3f, 0.3f, 0.3f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.specular"), 0.1f, 0.1f, 0.1f);

// 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"), LightP2.x, LightP2.y, LightP2.z);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[1].specular"), 0.1f, 0.1f, 0.1f);
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"), LightP3.x, LightP3.y, LightP3.z);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[2].specular"), 0.1f, 0.1f, 0.1f);
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);
```

- Se dibujan los modelos 3D, y se les aplica lo necesario para dejarlos en la posición deseada (así como las variables necesarias para realizar las animaciones).

```
//Carga de modelos
        //FACHADA
        view = camera.GetViewMatrix();
        glm::mat4 model(1);
        tmp = model = glm::translate(model, glm::vec3(0, 1, 0));
        glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
        Fachada.Draw(lightingShader);

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

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

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

        //Pizarron2
        view = camera.GetViewMatrix();
        glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
        Pizarron2.Draw(lightingShader);

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

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

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

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

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

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

        //EXTRAS

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

        //ANIMADOS
        //Zaguan
        view = camera.GetViewMatrix();
        model = glm::mat4(1);
        model = glm::translate(model, glm::vec3(4.7f, 4.2f, 3.3781f));
        model = glm::rotate(model, glm::radians(rotGarage), glm::vec3(0.0f, 0.0f, 1.0f));
        glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
        Zaguan.Draw(lightingShader);

        //Puerta Garage
        view = camera.GetViewMatrix();
        model = glm::mat4(1);
        model = glm::translate(model, glm::vec3(-2.8f, 2.23f, -0.6996f));
        model = glm::rotate(model, glm::radians(rotGarageDoor), glm::vec3(0.0f, 1.0f, 0.0f));
        glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
        PuertaGarage.Draw(lightingShader);

        //Puerta Casa
        view = camera.GetViewMatrix();
        model = glm::mat4(1);
        model = glm::translate(model, glm::vec3(0.96f, 2.23f, -10.47f));
        model = glm::rotate(model, glm::radians(rotHouseDoor), glm::vec3(0.0f, 1.0f, 0.0f));
        glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
        PuertaCasa.Draw(lightingShader);

        //Carro
        view = camera.GetViewMatrix();
        model = glm::mat4(1);
        model = glm::translate(model, PosIniCar + glm::vec3(movCarX, 0.0f, movCarZ));
        model = glm::rotate(model, glm::radians(rotCar), glm::vec3(0.0f, 1.0f, 0.0));
        glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
        Carro.Draw(lightingShader);

        //SIlla
        view = camera.GetViewMatrix();
        model = glm::mat4(1);
        model = glm::translate(model, PosIniChair + glm::vec3(movChairX, 0.0f, movChairZ));
        model = glm::rotate(model, glm::radians(rotCh), glm::vec3(0.0f, 1.0f, 0.0));
        glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
        Silla.Draw(lightingShader);

        //Portal
        view = camera.GetViewMatrix();
        model = glm::mat4(1);
        model = glm::translate(model, glm::vec3(12.0f, 3.0f, 3.15));
        model = glm::rotate(model, glm::radians(rotPortal), glm::vec3(1.0f, 0.0f, 0.0));
        glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
        Portal.Draw(lightingShader);
        glBindVertexArray(0);
```

- Se crea la animación por hueso/esqueleto con el animShader.

```
//Rick Animado
        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"), 0.2f, 0.2f, 0.2f);
        glUniform1f(glGetUniformLocation(animShader.Program, "material.shininess"), 32.0f);
        glUniform3f(glGetUniformLocation(animShader.Program, "light.ambient"), 0.5f, 0.5f, 0.5f);
        glUniform3f(glGetUniformLocation(animShader.Program, "light.diffuse"), 0.5f, 0.5f, 0.5f);
        glUniform3f(glGetUniformLocation(animShader.Program, "light.specular"), 0.2f, 0.2f, 0.2f);
        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(8.0f,1.0f,3.0));
        model = glm::rotate(model, glm::radians(90.0f), glm::vec3(0.0f, 1.0f, 0.0f));
        model = glm::scale(model, glm::vec3(0.01f));
        glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
        animacionPersonaje.Draw(animShader);
        glBindVertexArray(0);
```

- Se dibuja el skybox y se cambian los buffers de la pantalla

```
// 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);
```

- Finalmente se eliminan los arreglos de VAO, lightVAO, VBO, EBO, skyboxVAO y skyboxVBO.

```
    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;
}
```