U.N.A.M

Facultad de Ingeniería

Proyecto

Fecha: 21/11/2021

Manual Técnico

Alumno: Camacho Morales Gerardo Iván

Profesor: Ing. Carlos Aldair Román Balbuena

Grupo Laboratorio: 09

Semestre: 2022-1

Objetivos

El alumno deberá aplicar y demostrar los conocimientos adquiridos durante

todo el curso.

Introducción

Para este manual veremos el código que se utilizo para modelar en OpenGL nuestro ambiente, los problemas que se tuvieron, el manejo de las matrices, la adaptación del Directional light, los Points light y SpotLight. Así como las animaciones realizadas en lo objetos exportados que se hicieron en el software Maya.

Desarrollo

```
#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"
```

Comenzamos agregando las bibliotecas que contienen nuestro programas para el manejo correcto de la cámara, textura y objetos

```
// 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
Camera  camera(glm::vec3(-94.0f, 5.0f, -35.0f));
GLfloat lastX = WIDTH / 2.0;
GLfloat lastY = HEIGHT / 2.0;
bool keys[1024];
bool firstMouse = true;
float range = 0.0f;
```

```
float rot = 0.0f;

//-----------------------------------------------------------
//Variables de animaciones
//-----------------------------------------------------------

//-----------------------------------------------------------
//Puerta
float rotP = 0.0f;
bool activaP = false;
float abierta = 0.0f;

//-----------------------------------------------------------
//Capsula
float rotC = 0.0f;
bool activaC = false;
float arriba = 0.0f;

//-----------------------------------------------------------
//Faro
float rotF = -10.0f;
bool activaF = false;
float vuelta = 0.0f;
float luzF = 0.5f;

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


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

Comenzamos agregando variables para nuestros objetos, que permitirán cambiar su posición en nuestro ambiente, así como su dirección. Es importante este punto ya que son los que nos permiten rotar, trasladar o escalar los objetos, así como nuestras luces y la cámara.

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

#define MAX_FRAMES 50
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
	float rotRodIzq;
	float rotRodDer;
	float rotBraIzq;
	float rotBraDer;
	float rotCuerpo;
	float rotInc;
	float rotInc2;
	float rotInc3;
	float rotInc4;
	float rotInc5;

}FRAME;
```

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

Continuamos con la asignación de nuestros frames, donde señalamos el máximo de frames que podemos utilizar, cuales son los que vamos a necesitar, por ejemplo para una extremidad de nuestro personaje del cual queremos guardar su posición inicial y la final para poder realiza nuestra animación. También podemos ver la variables que permitirán reproducir nuestra animación y el contador para guardar en que frame nos encontramos.

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

};

glm::vec3 LightP1;
glm::vec3 LightP2;
glm::vec3 LightP3;
glm::vec3 LightP4;
```

Contamos con los vectores para los Pont lights y las variables para poder asignarle el color deseado posteriormente.

```
//A lo frames les asignamos los valores de sus posiciones
void cargaFrame(void)
{
	FrameIndex = 0;
	KeyFrame[0].posX = -95.000000;
	KeyFrame[0].posY = 1.000000;
	KeyFrame[0].posZ = -45.000000;
	KeyFrame[0].rotRodIzq = 0.000000;
	KeyFrame[0].rotRodDer = 0.000000;
	KeyFrame[0].rotBraIzq = 0.000000;
	KeyFrame[0].rotBraDer = 0.000000;
	KeyFrame[0].rotCuerpo = 0.000000;

	FrameIndex = 1;
	KeyFrame[1].posX = -95.000000;
	KeyFrame[1].posY = 1.000000;
	KeyFrame[1].posZ = -44.400101;
	KeyFrame[1].rotRodIzq = 80.000000;
	KeyFrame[1].rotRodDer = -45.000000;
	KeyFrame[1].rotBraIzq = 80.000000;
	KeyFrame[1].rotBraDer = -45.000000;
	KeyFrame[1].rotCuerpo = -21.000000;

	FrameIndex = 2;
	KeyFrame[2].posX = -95.140030;
	KeyFrame[2].posY = 1.000000;
	KeyFrame[2].posZ = -43.670223;
	KeyFrame[2].rotRodIzq = -10.000000;
	KeyFrame[2].rotRodDer = -8.000000;
	KeyFrame[2].rotBraIzq = 80.000000;
	KeyFrame[2].rotBraDer = 80.000000;
	KeyFrame[2].rotCuerpo = -21.000000;

	FrameIndex = 3;
	KeyFrame[3].posX = -94.949989;
```

```
KeyFrame[3].posY = 1.360000;
KeyFrame[3].posZ = -43.410267;
KeyFrame[3].rotRodIzq = -45.000000;
KeyFrame[3].rotRodDer = -14.000000;
KeyFrame[3].rotBraIzq = 36.000000;
KeyFrame[3].rotBraDer = 40.000000;
KeyFrame[3].rotCuerpo = -89.000000;

FrameIndex = 4;
KeyFrame[4].posX = -95.250778;
KeyFrame[4].posY = 1.289995;
KeyFrame[4].posZ = -43.209839;
KeyFrame[4].rotRodIzq = -16.000153;
KeyFrame[4].rotRodDer = -5.000078;
KeyFrame[4].rotBraIzq = -8.000408;
KeyFrame[4].rotBraDer = 6.000175;
KeyFrame[4].rotCuerpo = -101.000038;

FrameIndex = 5;
KeyFrame[5].posX = -95.552292;
KeyFrame[5].posY = 1.559985;
KeyFrame[5].posZ = -43.279026;
KeyFrame[5].rotRodIzq = -45.000481;
KeyFrame[5].rotRodDer = -45.000149;
KeyFrame[5].rotBraIzq = 80.999191;
KeyFrame[5].rotBraDer = 80.000381;
KeyFrame[5].rotCuerpo = -200.999695;

FrameIndex = 6;
KeyFrame[6].posX = -96.012390;
KeyFrame[6].posY = 1.279985;
KeyFrame[6].posZ = -43.369011;
KeyFrame[6].rotRodIzq = 26.999519;
KeyFrame[6].rotRodDer = 24.999851;
KeyFrame[6].rotBraIzq = -26.000809;
KeyFrame[6].rotBraDer = -27.999619;
KeyFrame[6].rotCuerpo = -285.999695;

FrameIndex = 7;
KeyFrame[7].posX = -95.643181;
KeyFrame[7].posY = 1.609973;
KeyFrame[7].posZ = -43.257786;
KeyFrame[7].rotRodIzq = -45.000793;
KeyFrame[7].rotRodDer = -45.000225;
KeyFrame[7].rotBraIzq = 80.998962;
KeyFrame[7].rotBraDer = 80.000450;
KeyFrame[7].rotCuerpo = -206.000122;

FrameIndex = 8;
KeyFrame[8].posX = -95.223091;
KeyFrame[8].posY = 1.309974;
KeyFrame[8].posZ = -43.237789;
KeyFrame[8].rotRodIzq = -8.000793;
KeyFrame[8].rotRodDer = -2.000225;
KeyFrame[8].rotBraIzq = 11.998962;
KeyFrame[8].rotBraDer = 14.000450;
KeyFrame[8].rotCuerpo = -107.000122;

FrameIndex = 9;
KeyFrame[9].posX = -95.573311;
KeyFrame[9].posY = 1.159962;
KeyFrame[9].posZ = -43.506535;
KeyFrame[9].rotRodIzq = 65.998901;
KeyFrame[9].rotRodDer = 59.999718;
KeyFrame[9].rotBraIzq = 13.998520;
KeyFrame[9].rotBraDer = 4.000427;
```

```
KeyFrame[9].rotCuerpo = -184.999878;

FrameIndex = 10;
KeyFrame[10].posX = -95.574036;
KeyFrame[10].posY = 1.119953;
KeyFrame[10].posZ = -43.795429;
KeyFrame[10].rotRodIzq = -23.001297;
KeyFrame[10].rotRodDer = 23.999756;
KeyFrame[10].rotBraIzq = 13.998037;
KeyFrame[10].rotBraDer = 4.000429;
KeyFrame[10].rotCuerpo = -184.999161;

FrameIndex = 11;
KeyFrame[11].posX = -95.574036;
KeyFrame[11].posY = 0.969953;
KeyFrame[11].posZ = -43.945404;
KeyFrame[11].rotRodIzq = -45.001297;
KeyFrame[11].rotRodDer = 21.999756;
KeyFrame[11].rotBraIzq = -19.001963;
KeyFrame[11].rotBraDer = 47.000427;
KeyFrame[11].rotCuerpo = -158.999161;

FrameIndex = 12;
KeyFrame[12].posX = -96.154160;
KeyFrame[12].posY = 0.959953;
KeyFrame[12].posZ = -44.395329;
KeyFrame[12].rotRodIzq = 33.998703;
KeyFrame[12].rotRodDer = -45.000244;
KeyFrame[12].rotBraIzq = 28.998039;
KeyFrame[12].rotBraDer = -20.999573;
KeyFrame[12].rotCuerpo = -158.999161;

FrameIndex = 13;
KeyFrame[13].posX = -96.704277;
KeyFrame[13].posY = 0.959953;
KeyFrame[13].posZ = -44.795261;
KeyFrame[13].rotRodIzq = -45.001297;
KeyFrame[13].rotRodDer = 28.999756;
KeyFrame[13].rotBraIzq = -29.001961;
KeyFrame[13].rotBraDer = 50.000427;
KeyFrame[13].rotCuerpo = -121.999161;

FrameIndex = 14;
KeyFrame[14].posX = -97.094795;
KeyFrame[14].posY = 0.959946;
KeyFrame[14].posZ = -45.064838;
KeyFrame[14].rotRodIzq = -15.001671;
KeyFrame[14].rotRodDer = -6.000076;
KeyFrame[14].rotBraIzq = 80.997459;
KeyFrame[14].rotBraDer = 80.000320;
KeyFrame[14].rotCuerpo = -85.999962;

FrameIndex = 15;
KeyFrame[15].posX = -97.244682;
KeyFrame[15].posY = 1.479934;
KeyFrame[15].posZ = -45.064507;
KeyFrame[15].rotRodIzq = 23.998016;
KeyFrame[15].rotRodDer = 35.000065;
KeyFrame[15].rotBraIzq = 62.996880;
KeyFrame[15].rotBraDer = 66.000595;
KeyFrame[15].rotCuerpo = -86.000969;

FrameIndex = 16;
KeyFrame[16].posX = -97.484734;
KeyFrame[16].posY = 1.899934;
KeyFrame[16].posZ = -45.064507;
```

```
KeyFrame[16].rotRodIzq = 66.998016;
KeyFrame[16].rotRodDer = 68.000061;
KeyFrame[16].rotBraIzq = 80.996880;
KeyFrame[16].rotBraDer = 80.000595;
KeyFrame[16].rotCuerpo = -86.000969;

FrameIndex = 17;
KeyFrame[17].posX = -97.814659;
KeyFrame[17].posY = 1.659923;
KeyFrame[17].posZ = -45.064507;
KeyFrame[17].rotRodIzq = 80.997559;
KeyFrame[17].rotRodDer = 80.999962;
KeyFrame[17].rotBraIzq = 80.995514;
KeyFrame[17].rotBraDer = 80.000595;
KeyFrame[17].rotCuerpo = -68.002419;

FrameIndex = 18;
KeyFrame[18].posX = -98.155167;
KeyFrame[18].posY = 1.499903;
KeyFrame[18].posZ = -44.864540;
KeyFrame[18].rotRodIzq = 80.997650;
KeyFrame[18].rotRodDer = 80.999840;
KeyFrame[18].rotBraIzq = 80.994064;
KeyFrame[18].rotBraDer = 80.000595;
KeyFrame[18].rotCuerpo = -54.004372;

FrameIndex = 19;
KeyFrame[19].posX = -98.655853;
KeyFrame[19].posY = 1.259883;
KeyFrame[19].posZ = -44.404541;
KeyFrame[19].rotRodIzq = 80.997650;
KeyFrame[19].rotRodDer = 80.999840;
KeyFrame[19].rotBraIzq = 80.992615;
KeyFrame[19].rotBraDer = 80.000595;
KeyFrame[19].rotCuerpo = -45.006248;

FrameIndex = 20;
KeyFrame[20].posX = -99.396011;
KeyFrame[20].posY = 0.899883;
KeyFrame[20].posZ = -43.834637;
KeyFrame[20].rotRodIzq = 80.997650;
KeyFrame[20].rotRodDer = 80.999840;
KeyFrame[20].rotBraIzq = 80.992615;
KeyFrame[20].rotBraDer = 80.000595;
KeyFrame[20].rotCuerpo = -45.006248;

FrameIndex = 21;
KeyFrame[21].posX = -99.976715;
KeyFrame[21].posY = 0.809868;
KeyFrame[21].posZ = -43.194641;
KeyFrame[21].rotRodIzq = 80.997650;
KeyFrame[21].rotRodDer = 80.999840;
KeyFrame[21].rotBraIzq = 80.991165;
KeyFrame[21].rotBraDer = 80.000595;
KeyFrame[21].rotCuerpo = -14.007950;

FrameIndex = 22;
KeyFrame[22].posX = -99.976715;
KeyFrame[22].posY = 0.879868;
KeyFrame[22].posZ = -42.684727;
KeyFrame[22].rotRodIzq = -4.002350;
KeyFrame[22].rotRodDer = -15.000160;
KeyFrame[22].rotBraIzq = -0.008835;
KeyFrame[22].rotBraDer = 16.000595;
KeyFrame[22].rotCuerpo = -2.007950;
```

```
	FrameIndex++;
}
```

En esta parte se guardaron todas las posiciones de nuestras animaciones complejas, para cada objeto se guardo su posición en X, Y y Z y la rotación que tuvo respecto a su posición inicial. Con un total de 23 frames logramos realizar las dos animaciones, para obtener cada posición correctamente se imprimieron en consola y después se agregaron al código.

```
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;
	rotCuerpo = KeyFrame[0].rotCuerpo;


}

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].rotCuerpo -
KeyFrame[playIndex].rotCuerpo) / i_max_steps;
}
```

La primer función permite reiniciar la animación cargando el primer frame guardado y asignado los valores iniciales. Posteriormente vemos la función de interpolación que toma dos puntos para ver el cambio realizado entre ellos y así dar los cambios que hubo del primer punto al último.

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

	GLFWwindow* window = glfwCreateWindow(WIDTH, HEIGHT, "Practica 11", 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");
```

Comenzamos nuestro main llamando los shaders necesarios para poder correr nuestro programa.

```
	Model PiernaDer((char*)"Modelos/Mickey/piernader.obj");
	Model PiernaIzq((char*)"Modelos/Mickey/piernaizq.obj");
	Model Torso((char*)"Modelos/Mickey/torso.obj");
	Model BrazoDer((char*)"Modelos/Mickey/brazoder.obj");
	Model BrazoIzq((char*)"Modelos/Mickey/brazoizq.obj");
	Model Cabeza((char*)"Modelos/Mickey/cabeza.obj");
	Model Casa((char*)"Modelos/CasaMickey/CasaMickey.obj");
	Model Puerta((char*)"Modelos/CasaMickey/puertaMickey.obj");
	Model Capsula((char*)"Modelos/CasaMickey/capsulaMickey.obj");
	Model Faro((char*)"Modelos/CasaMickey/FaroMickey.obj");

	// Build and compile our shader program

	//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].rotCuerpo = 0;
		KeyFrame[i].rotInc = 0;
		KeyFrame[i].rotInc2 = 0;
		KeyFrame[i].rotInc3 = 0;
		KeyFrame[i].rotInc4 = 0;
```

```
		KeyFrame[i].rotInc5 = 0;
	}
```

Importamos nuestros objetos realizados en Maya y todos nuestros frames los inicializamos en 0, evitando que haya datos basura. Al importar los objetos nunca se tuvo problema ya que el archivo obj siempre se guardo en la misma carpeta junto con las texturas utilizadas. Algunas cuentan con un diseño de 64x64 pixeles o 512x512 para que nuestro programa corra correctamente.

```
	// 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
	};


	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)
    };
```

Continuando con la asignación de nuestros índices y de nuestro skybox que es el que permitirá darle el ambiente, recordando que próximamente importaremos los archivos .tga.

```
// 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), (GLvoid *)0);
// Note that we skip over the other data in our buffer object (we don't need the
normals/textures, only positions).
        glEnableVertexAttribArray(0);
        glBindVertexArray(0);


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

        // 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");
```

Se importaron un total de 6 archivos tga que son los que pertenecen a cada cara del cubo de nuestro skybox. Al momento de importarlos se veían extraños como si tuvieran una especie de espejo, provocando que no diera ese realismo. Para solucionarlo en el software GIMP 2 se aplicó de nuevo el espejo.

```
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);


                // 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);
                // == ===========================
                // == ===========================
                // Directional light
                glUniform3f(glGetUniformLocation(lightingShader.Program,
"dirLight.direction"), -0.2f, -1.0f, -0.3f);
                glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.ambient"),
0.25f, 0.25f, 0.25f);
                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 model = glm::translate(model, glm::vec3(-95.0f, 1.0f, -
45.0f));
                glUniform3f(glGetUniformLocation(lightingShader.Program,
"pointLights[0].position"), -95.029f, 3.426f, -45.0019f);
                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"), -96.502f, 2.352f, -45.992);
                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"), LightP2.x, LightP2.y, LightP2.z);
                glUniform1f(glGetUniformLocation(lightingShader.Program,
"pointLights[1].constant"), 1.0f);
                glUniform1f(glGetUniformLocation(lightingShader.Program,
"pointLights[1].linear"), 0.05f);
```

```
                glUniform1f(glGetUniformLocation(lightingShader.Program,
"pointLights[1].quadratic"), 0.5f);

                // Point light 3
                glUniform3f(glGetUniformLocation(lightingShader.Program,
"pointLights[2].position"), -94.183f, 2.352f, -43.127f);
                glUniform3f(glGetUniformLocation(lightingShader.Program,
"pointLights[2].ambient"), 0.005f, 0.005f, 0.005f);
                glUniform3f(glGetUniformLocation(lightingShader.Program,
"pointLights[2].diffuse"), LightP3.x, LightP3.y, LightP3.z);
                glUniform3f(glGetUniformLocation(lightingShader.Program,
"pointLights[2].specular"), LightP3.x, LightP3.y, LightP3.z);
                glUniform1f(glGetUniformLocation(lightingShader.Program,
"pointLights[2].constant"), 1.0f);
                glUniform1f(glGetUniformLocation(lightingShader.Program,
"pointLights[2].linear"), 0.05f);
                glUniform1f(glGetUniformLocation(lightingShader.Program,
"pointLights[2].quadratic"), 0.5f);

                // Point light 4
                glUniform3f(glGetUniformLocation(lightingShader.Program,
"pointLights[3].position"), -91.074f, 2.514f, -44.978f);
                glUniform3f(glGetUniformLocation(lightingShader.Program,
"pointLights[3].ambient"), 0.005f, 0.005f, 0.005f);
                glUniform3f(glGetUniformLocation(lightingShader.Program,
"pointLights[3].diffuse"), LightP4.x, LightP4.y, LightP4.z);
                glUniform3f(glGetUniformLocation(lightingShader.Program,
"pointLights[3].specular"), LightP4.x, LightP4.y, LightP4.z);
                glUniform1f(glGetUniformLocation(lightingShader.Program,
"pointLights[3].constant"), 1.0f);
                glUniform1f(glGetUniformLocation(lightingShader.Program,
"pointLights[3].linear"), 0.05f);
                glUniform1f(glGetUniformLocation(lightingShader.Program,
"pointLights[3].quadratic"), 0.5f);

                // SpotLight
                glUniform3f(glGetUniformLocation(lightingShader.Program,
"spotLight.position"), -97.35f, 7.497f, -45.03f);
                glUniform3f(glGetUniformLocation(lightingShader.Program,
"spotLight.direction"), 0.3f, 0.1f, luzF);
                glUniform3f(glGetUniformLocation(lightingShader.Program,
"spotLight.ambient"), 0.0f, 0.0f, 1.0f);
                glUniform3f(glGetUniformLocation(lightingShader.Program,
"spotLight.diffuse"), 0.0f, 0.0f, 1.0f);
                glUniform3f(glGetUniformLocation(lightingShader.Program,
"spotLight.specular"), 1.0f, 1.0f, 1.0f);
                glUniform1f(glGetUniformLocation(lightingShader.Program,
"spotLight.constant"), 0.1f);
                glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight.linear"),
0.006f);
                glUniform1f(glGetUniformLocation(lightingShader.Program,
"spotLight.quadratic"), 0.001f);
                glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight.cutOff"),
glm::cos(glm::radians(30.5f)));
                glUniform1f(glGetUniformLocation(lightingShader.Program,
"spotLight.outerCutOff"), glm::cos(glm::radians(40.0f)));
```

Continuamos seleccionando la vista de la cámara que vamos a tener, seleccionando la perspectiva. Posteriormente vemos nuestras luces; para la ambiental se decidió no dejarla demasiado alta ya que no se apreciaría tanto los Ponts light, los cuales son 4 y está puestos para ponerlos en las 2 lámparas de nuestro modelo, y en dos focos que se encuentran en el techo de nuestra casa.

Una lámpara tiene el color verde y la otra el color rojo, un foco tiene asignado el color amarillo y el otro el color cian. Podemos ver que en el diffuse y el especular se tienen variables estas son las que permiten que se apaguen y se prendan.

Finalmente en el SpotLight que se tiene asignado el color azul fuerte y este estará asignado a nuestro faro, se le tiene asignada una variable que permitirá su rotación junto con la del faro.

```
// 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


                //-------------------------------------------------------------------------
---------------
                //      CARGAMOS LOS MODELOS
                // -------------------------------------------------------------------------
---------------

                // -------------------------------------------------------------------------
---------------
                //PERSONAJE MICKEY
                // -------------------------------------------------------------------------
---------------
                view = camera.GetViewMatrix();
                glm::mat4 model(1);
                model = glm::translate(model,glm::vec3(posX,posY,posZ));
                model = glm::rotate(model, glm::radians(rotCuerpo), glm::vec3(0.0f, 1.0f,
0.0));
                model = glm::translate(model, glm::vec3(0.028f, 0.396f, -0.003f));
                glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
                Torso.Draw(lightingShader);

                // -------------------------------------------------------------------------
---------------
                //Pierna Der
                view = camera.GetViewMatrix();
                model = glm::mat4(1);
                model = glm::translate(model, glm::vec3(posX, posY, posZ));
                model = glm::rotate(model, glm::radians(rotCuerpo), glm::vec3(0.0f, 1.0f,
0.0));
                model = glm::translate(model, glm::vec3(-0.014f, 0.29f, -0.006f));
                model = glm::rotate(model, glm::radians(-rotRodIzq), glm::vec3(1.0f, 0.0f,
0.0f));
                glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
                PiernaDer.Draw(lightingShader);

                // -------------------------------------------------------------------------
---------------
                //Pierna izq
                view = camera.GetViewMatrix();
                model = glm::mat4(1);
                model = glm::translate(model, glm::vec3(posX, posY, posZ));
```

```
                model = glm::rotate(model, glm::radians(rotCuerpo), glm::vec3(0.0f, 1.0f,
0.0f));
                model = glm::translate(model, glm::vec3(0.06f, 0.29f, -0.007f));
                model = glm::rotate(model, glm::radians(-rotRodDer), glm::vec3(1.0f, 0.0f,
0.0f));
                glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
                PiernaIzq.Draw(lightingShader);

                // -------------------------------------------------------------------------
--------------
                //Brazo derecho
                view = camera.GetViewMatrix();
                model = glm::mat4(1);
                model = glm::translate(model, glm::vec3(posX, posY, posZ));
                model = glm::rotate(model, glm::radians(rotCuerpo), glm::vec3(0.0f, 1.0f,
0.0f));
                model = glm::translate(model, glm::vec3(-0.007f, 0.477f, -0.017));
                model = glm::rotate(model, glm::radians(-rotBraDer), glm::vec3(1.0f, 0.0f,
0.0f));
                glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
                BrazoDer.Draw(lightingShader);

                // -------------------------------------------------------------------------
--------------
                //Brazo Izquierdo
                view = camera.GetViewMatrix();
                model = glm::mat4(1);
                model = glm::translate(model, glm::vec3(posX, posY, posZ));
                model = glm::rotate(model, glm::radians(rotCuerpo), glm::vec3(0.0f, 1.0f,
0.0f));
                model = glm::translate(model, glm::vec3(0.064f, 0.473f, -0.013));
                model = glm::rotate(model, glm::radians(-rotBraIzq), glm::vec3(1.0f, 0.0f,
0.0f));
                glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
                BrazoIzq.Draw(lightingShader);

                // -------------------------------------------------------------------------
--------------
                //Cabeza
                view = camera.GetViewMatrix();
                model = glm::mat4(1);
                model = glm::translate(model, glm::vec3(posX, posY, posZ));
                model = glm::translate(model, glm::vec3(0.0f, 0.1f, 0.0f));
                model = glm::rotate(model, glm::radians(rotCuerpo), glm::vec3(0.0f, 1.0f,
0.0));
                glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
                Cabeza.Draw(lightingShader);


                // -------------------------------------------------------------------------
--------------
                //CASA DE MICKEY
                // -------------------------------------------------------------------------
--------------
                view = camera.GetViewMatrix();
                model = glm::mat4(1);
                model = glm::translate(model, glm::vec3(-95.0f, 1.0f, -45.0f));
                glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
                Casa.Draw(lightingShader);




                // -------------------------------------------------------------------------
--------------
                //PUERTA DE MICKEY
                // -------------------------------------------------------------------------
--------------
                view = camera.GetViewMatrix();
                model = glm::mat4(1);
                model = glm::translate(model, glm::vec3(-89.454f, 1.608f, -45.476f));
```

```
                model = glm::rotate(model, glm::radians(rotP), glm::vec3(0.0f, -1.0f, 0.0));
                glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
                Puerta.Draw(lightingShader);


                // -----------------------------------------------------------------------
--------------
                //CAPSULA DE MICKEY
                // -----------------------------------------------------------------------
--------------
                view = camera.GetViewMatrix();
                model = glm::mat4(1);
                model = glm::translate(model, glm::vec3(-96.977f, 7.105f, -46.337f));
                model = glm::rotate(model, glm::radians(rotC), glm::vec3(-1.0f, 0.0f, 0.0));
                glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
                Capsula.Draw(lightingShader);

                // -----------------------------------------------------------------------
--------------
                //FARO DE MICKEY
                // -----------------------------------------------------------------------
--------------
                view = camera.GetViewMatrix();
                model = glm::mat4(1);
                model = glm::translate(model, glm::vec3(-96.97f, 7.497f, -45.039f));
                model = glm::rotate(model, glm::radians(rotF), glm::vec3(0.0f, -1.0f, 0.0));
                glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
                Faro.Draw(lightingShader);


                glBindVertexArray(0);
```

Tenemos entonces cargados nuestros objetos y los estamos asignando a una posición inicial, en algunos de ellos como en la de nuestro personaje vemos que tiene incluso rotaciones. Este fue un problema al inicio ya que al trabajar con matrices recordamos que no son conmutativas, por lo tanto si queríamos guardar su rotación y su traslación correctamente teníamos que ponerlos en un orden especifico para que al momento de hacer las animaciones no se vieran extrañas.

En ellas se tienen asignadas variables que utilizamos en los frames para poder guardar su posición y así animarlos.

```
                // 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)
                glBindVertexArray(lightVAO);
                for (GLuint i = 0; i < 4; i++)
                {
                        model = glm::mat4(1);
                        model = glm::translate(model, pointLightPositions[i]);
                        model = glm::scale(model, glm::vec3(0.2f)); // Make it a smaller cube
                        glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
                        glDrawArrays(GL_TRIANGLES, 0, 36);
```

```
                }
                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;
}
```

Terminamos nuestro main con el seteo de las matrices y la colocación de nuestro skybox.

```
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;
                        rotCuerpo += KeyFrame[playIndex].rotInc5;

                        i_curr_steps++;
                }

        }
}
```

Esta función permite ejecutar la animación que hemos guardado en nuestros frames asignando los valores guardados a los valores actuales. Ya que de un punto a otro hay demasiados cambios por la interpolación es una función bastante pesada.

```
// Is called whenever a key is pressed/released via GLFW
void KeyCallback(GLFWwindow *window, int key, int scancode, int action, int mode)
{
        if (keys[GLFW_KEY_L])
        {
                cargaFrame();
                if (play == false && (FrameIndex > 1))
                {

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

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

        }



        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(0.0f, 1.0f, 1.0f);
				LightP2 = glm::vec3(0.0f, 1.0f, 0.0f);
				LightP3 = glm::vec3(1.0f, 0.0f, 0.0f);
				LightP4 = glm::vec3(1.0f, 1.0f, 0.0f);
			}

			else {
				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);
				LightP4 = glm::vec3(0.0f, 0.0f, 0.0f);
			}

		}

		//Activa la animacion de la puerta
		if (keys[GLFW_KEY_P])
		{
			if (activaP == false)
				activaP = true;
			else
				activaP = false;
		}

		//Activa la animacion de la capsual
		if (keys[GLFW_KEY_O])
		{
			if (activaC == false)
				activaC = true;
			else
				activaC = false;
		}

		//Activa la animacion del faro
		if (keys[GLFW_KEY_I])
		{
			if (activaF == false)
				activaF = true;
			else
				activaF = false;
		}


}
```

Esta función del código es la encargada de detectar y activar las animaciones cuando se presiona la tecla asignada; vemos en el caso de las luces funcionan con la barra espaciadora, para la puerta es la tecla P, faro I, capsula O y finalmente la animación compleja comienza con nuestra tecla L.

Observamos que en la mayor parte vemos valores true y false, ya que gracias a ellos es que decimos si se activan o no, permitiendo en el caso de las animaciones sencillas dar la ilusión de pausa o reanudar.

```
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);
}
```

En esta sección vemos los controles de la cámara que tiene que ver con el mouse.

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

        if (keys[GLFW_KEY_M])
        {

                rot += 1;
                rotCuerpo += 1;

        }

        if (keys[GLFW_KEY_N])
        {

                rot += 1;
                rotCuerpo -= 1;

        }

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

        }

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

        }

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

        }

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

        }

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

        }

        if (keys[GLFW_KEY_7])
        {
                if (rotBraIzq > -45)
```

```
                rotBraIzq -= 1.0f;

}

if (keys[GLFW_KEY_8])
{
        if (rotBraDer < 80.0f)
                rotBraDer += 1.0f;

}

if (keys[GLFW_KEY_9])
{
        if (rotBraDer > -45)
                rotBraDer -= 1.0f;


}

//Moviemiento de la puerta
if (activaP) {
        if (rotP >= -90 && abierta == 0.0f) {
                rotP -= 0.2f;
        }
        else
                abierta = 1.0f;

        if (rotP <= 0 && abierta == 1.0f) {
                rotP += 0.2f;
        }
        else
                abierta = 0.0f;
}

//Moviemiento de la capsula
if (activaC) {
        if (rotC <= 40 && arriba == 0.0f) {
                rotC += 0.05f;
        }
        else
                arriba = 1.0f;

        if (rotC >= 0 && arriba == 1.0f) {
                rotC -= 0.05f;
        }
        else
                arriba = 0.0f;
}

//Moviemiento del faro
if (activaF) {
        if (rotF >= -90 && vuelta == 0.0f) {
                rotF -= 0.2f;
                luzF -= 0.0016;
        }
        else
                vuelta = 1.0f;

        if (rotF <= -0.30 && vuelta == 1.0f) {
                rotF += 0.2f;
                luzF += 0.0016;
        }
        else
                vuelta = 0.0f;
}


//Mov Personaje
if (keys[GLFW_KEY_H])
{
        posZ += 0.01;
```

```
	}

	if (keys[GLFW_KEY_Y])
	{
		posZ -= 0.01;
	}

	if (keys[GLFW_KEY_G])
	{
		posX -= 0.01;
	}

	if (keys[GLFW_KEY_J])
	{
		posX += 0.01;
	}

	if (keys[GLFW_KEY_X])
	{
		posY -= 0.01;
	}

	if (keys[GLFW_KEY_Z])
	{
		posY += 0.01;
	}



	// 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);
	}


}
```

Concluimos nuestro programa con la función encargada de detectar y cambiar la posición de nuestros objetos; en el caso de nuestro personaje haciendo las rotaciones de sus extremidades o la posición de todo el personaje completo, para el caso de la puerta, capsula y faro veíamos como se hacia un aumento y decremento en los valores que pertenecían a la rotación.

Y en el caso de la cámara vemos que de igual forma cambiamos su posición en nuestro ambiente.

Conclusiones

El proyecto realizado es una prueba clara de cómo las matemáticas están en todos lados, en especial en lo que se refiere a animación y luces. Con el manejo de matrices y transformaciones procesamos información que permite la manipulación de objetos y con ello creando ilusiones que resultan en animaciones.

El software seleccionado para la creación de los modelos es demasiado amigable y muy intuitivo, permitiendo manejar las caras, vértices o líneas para lograr hacer el modelo lo más parecido posible a las imágenes mostradas.

El avance del curso y de las prácticas realizadas durante el mismo permitieron que realizara este proyecto de una forma rápida y no tan complicado. Dejando bases sólidas para importar objetos, escalarlos, trasladarlos o rotarlos a conveniencia, así como el manejo correcto de las luces y la iluminación de nuestro ambiente.

El resultado final es poder ver animaciones complejas de nuestros objetos, permitiendo guardar los datos de nuestros objetos y permitiendo la fluidez de su traslado, y rotación del punto A al B.