	**Documentación**
Secretaría/División: División de Ingeniería Eléctrica	Área/Departamento: Ingeniería en Computación

Laboratorio de Computación Gráfica

Proyecto Final

Nombre completo de los alumnos		Firma
Sánchez Bautista Alan Ulises		
N° de brigada:	**Fecha de entrega:** martes 19 de enero 2020	**Grupo:** 04
Calificación:	**Profesor:** Ing. Carlos Aldair Román Balbuena	

Sánchez Bautista Alan Ulises
Grupo: 04

Índice

1. Requisitos del proyecto

Objetivo
El alumno deberá aplicar y demostrar los conocimientos adquiridos durante todo el curso.

Descripción
El alumno deberá seleccionar una fachada y un espacio que pueden ser reales o ficticios y presentar imágenes de referencia de dichos espacios para su recreación 3D en OpenGL.

En la imagen de referencia se debe visualizar 7 objetos que el alumno va a recrear virtualmente y donde dichos objetos deben ser lo más parecido a su imagen de referencia.

Se debe recrear un objeto de los 7 usando las primitivas básicas en OpenGL.

Consideraciones
- El proyecto se debe entregar de forma individual, con un manual de usuario donde se explique cada interacción dentro del ambiente virtual recreado y un manual técnico que contenga la documentación del proyecto que incluya objetivos, diagrama de Gantt, alcance del proyecto y la documentación del código.

- Se debe compartir la liga de su proyecto en un repositorio en GitHub y se debe subir en la plataforma de classroom a más tardar a las 11:59 pm del martes para el grupo 04 y 09 y del día jueves para el grupo 08 para que el profesor pueda descargar el proyecto para su evaluación.

- Proyectos con evaluaciones menores a 5, se considerarán proyectos deficientes y por lo tanto serán sancionados con 1 o 2 puntos menos en su calificación final.

- Los objetos recreados repetidamente se contarán como un objeto dentro de la evaluación.

- Se debe ocupar el código base visto durante el curso y otorgado por el profesor.

Evaluación
Los puntos a evaluar son los siguientes:
- Presentación del proyecto→Se evaluará la presentación del manual de

usuario y técnico (10 puntos o -10 puntos).

- Proyecto en repositorio→20 puntos o -20.
- Realismo del espacio virtual contra la foto de referencia (40 puntos o -20 puntos).
- Archivo Ejecutable→ Existe, Abre y Funciona (5 puntos o -5 puntos).
- Modelado (mínimo 7 elementos ya especificados en el pdf donde se colocaron las imágenes de referencia) →Texturizado, geometría de los muebles y de los edificios (40 puntos).
- Animaciones→ Deben ser 5 animaciones donde sean 3 sencillas y 2 complejas (30 puntos o -30 puntos).
- Manejo de cámara (5 puntos).

Total del proyecto: 150 puntos-->10 Calif.

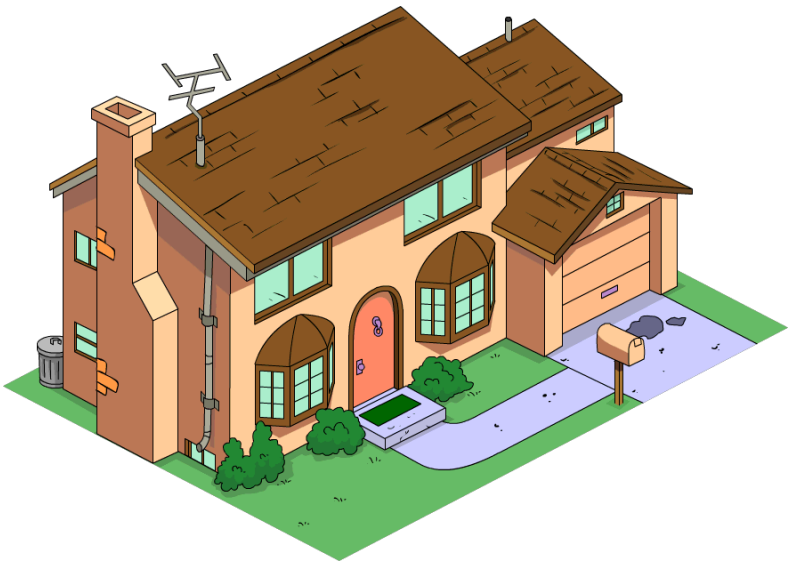
2. Propuesta

Alcance del proyecto:

Con este proyecto, espero aprobar el laboratorio de la materia de computación gráfica, así como tomar interés en el uso de gráficos 3D y 2D en el resto de mi carrera universitaria y posiblemente en el campo laboral, donde poder crear modelos, videojuegos, animaciones y usar los gráficos a mi favor.

Escenario propuesto:

Propongo utilizar la primera planta de la casa de Los Simpson como fachada.



Recrearé la sala de televisión dentro de la casa.

Sánchez Bautista Alan Ulises

Grupo: 04

Modelaré los siguientes objetos:

Televisión y videograbadora

Tapete

Puerta

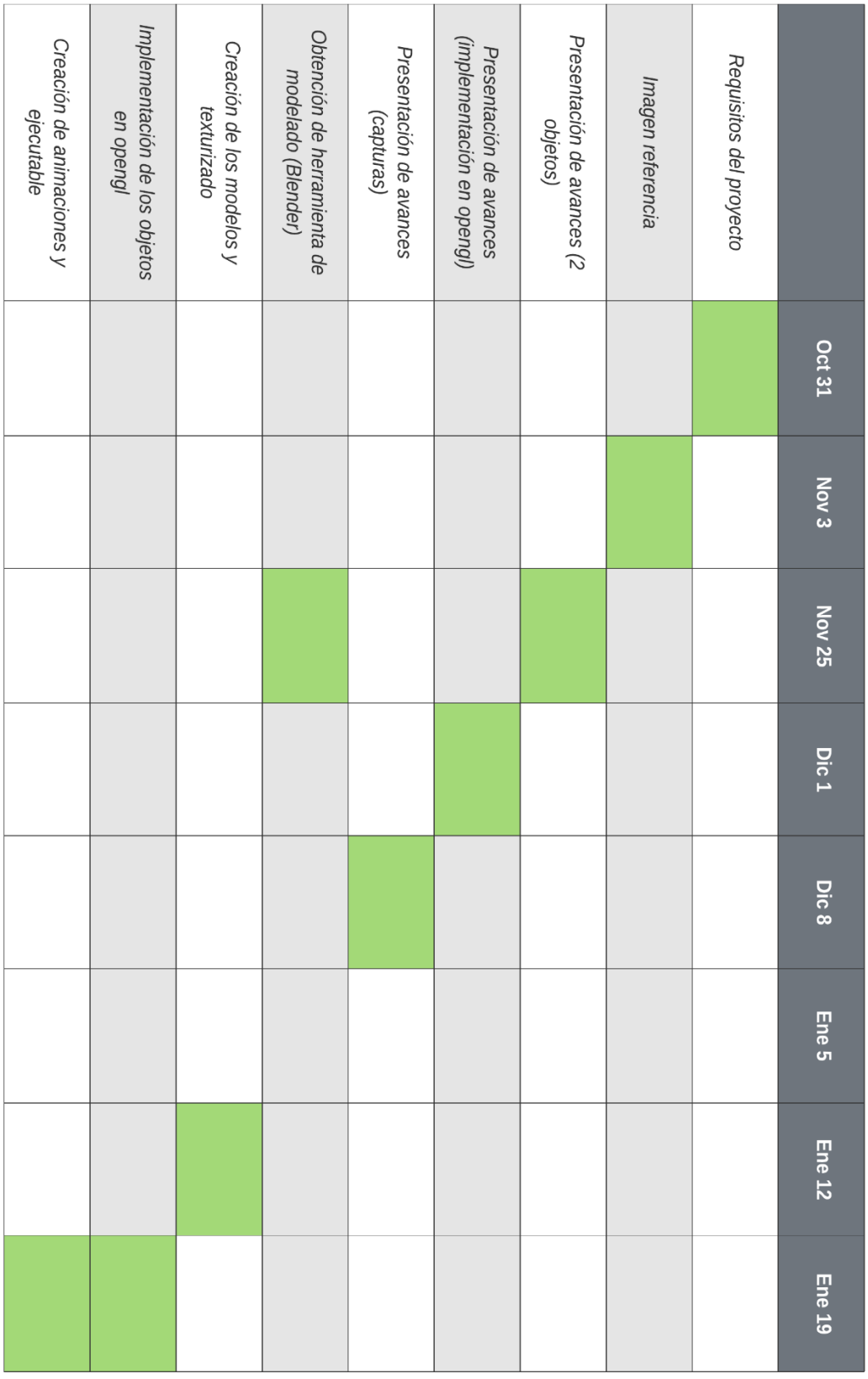
Sillón

Mueble

Mesa de teléfono

Pintura

3. Diagrama de Gantt



4. Documentación del código

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

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

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

// Other Libs
#include "stb_image.h"

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

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


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

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

// Window dimensions
const GLuint WIDTH = 800, HEIGHT = 600;
int SCREEN_WIDTH, SCREEN_HEIGHT;

// Camera
Camera  camera(glm::vec3(100.0f, 50.0f, 0.0f));
GLfloat lastX = WIDTH / 2.0;
GLfloat lastY = HEIGHT / 2.0;
bool keys[1024];
bool firstMouse = true;
float range = 0.0f;
float rot = 0.0f;
float PosInX = 30.0f, PosInY = 0.0f, PosInZ = -100.0f;
```

Para hacer las animaciones se crearon las siguientes variables, esto con el propósito de manipular los valores que tiene cada objeto en sus matrices de traslación y rotación, además de que algunas de estas variables como las de “recorrido” se utilizan para la lógica de las animaciones.

```
//Animación del libro
float movLibX = 0.0;
float movLibY = 0.0;
float movLibZ = 0.0;
float rotLib = 180.0;
float rotLibX = 90.0;

bool caidaLibro = false;
bool recorrido1 = true;
bool recorrido2 = false;
bool recorrido3 = false;
bool recorrido4 = false;
bool recorrido5 = false;

//Animación del cajón
float movCajZ = 0.0;

bool abrirCerrarCajon = false;
bool recorridoCaj1 = true;
bool recorridoCaj2 = false;
bool recorridoCaj3 = false;


//Animación de la puerta
float movPue = 90.0;
float posPueX = -61.5;
float posPueZ = -21.3;

bool abrirCerrarPue = false;
bool recorridoPue1 = true;
bool recorridoPue2 = false;

//Animación del barco
float movBar = 0.0;
float posBarX = -30.5f;

bool moverBarco = false;
bool recorridoBar1 = true;
bool recorridoBar2 = false;
bool recorridoBar3 = true;
bool recorridoBar4 = false;

//Animación de la antena izquierda
float rotAntIzq = -45.0;
float posAntIY = 23.9;

bool moverAntenaIzq = false;
bool recorridoIzq1 = true;
```

```
bool recorridoIzq2 = false;


//Animación de la antena derecha
float rotAntDerZ = 90.0;
float rotAntDerX = 0.0;

bool moverAntenaDer = false;
bool recorridoDer1 = true;
bool recorridoDer2 = false;
bool recorridoDer3 = true;
bool recorridoDer4 = false;



// Light attributes
glm::vec3 lightPos(0.0f, 0.0f, 0.0f);
glm::vec3 PosIni(30.0f, 0.0f, -100.0f);
bool active;


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

// Keyframes
float posX = PosIni.x, posY = PosIni.y, posZ = PosIni.z;

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

glm::vec3 LightP1;



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

      // Create a GLFWwindow object that we can use for GLFW's functions
      GLFWwindow* window = glfwCreateWindow(WIDTH, HEIGHT,
"SanchezBautistaAlanUlises_ProyectoFinal_GPO4", 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");

	Model Casa((char*)"Models/Proyecto/Casa.obj");
	Model Puerta((char*)"Models/Proyecto/Puerta.obj");
	Model Mueble((char*)"Models/Proyecto/Mueble.obj");
	Model Television((char*)"Models/Proyecto/Television.obj");
	Model Sillon((char*)"Models/Proyecto/Sillon.obj");
	Model Alfombra((char*)"Models/Proyecto/Alfombra.obj");
	Model Mesa((char*)"Models/Proyecto/Mesa.obj");
	Model Cuadro((char*)"Models/Proyecto/Cuadro.obj");
	Model Barco((char*)"Models/Proyecto/Barco.obj");
	Model Cajon((char*)"Models/Proyecto/Cajon.obj");
	Model Antena((char*)"Models/Proyecto/AntenaTele.obj");
```

```
Model Libro((char*)"Models/Proyecto/Libro.obj");
// Build and compile our shader program
// 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)
	};


	// 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");

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


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



            // Point light 2
            glUniform3f(glGetUniformLocation(lightingShader.Program,
"pointLights[1].position"), pointLightPositions[1].x, pointLightPositions[1].y,
pointLightPositions[1].z);
```

```
            glUniform3f(glGetUniformLocation(lightingShader.Program,
"pointLights[1].ambient"), 0.05f, 0.05f, 0.05f);
            glUniform3f(glGetUniformLocation(lightingShader.Program,
"pointLights[1].diffuse"), 1.0f, 1.0f, 0.0f);
            glUniform3f(glGetUniformLocation(lightingShader.Program,
"pointLights[1].specular"), 1.0f, 1.0f, 0.0f);
            glUniform1f(glGetUniformLocation(lightingShader.Program,
"pointLights[1].constant"), 1.0f);
            glUniform1f(glGetUniformLocation(lightingShader.Program,
"pointLights[1].linear"), 0.09f);
            glUniform1f(glGetUniformLocation(lightingShader.Program,
"pointLights[1].quadratic"), 0.032f);

            // Point light 3
            glUniform3f(glGetUniformLocation(lightingShader.Program,
"pointLights[2].position"), pointLightPositions[2].x, pointLightPositions[2].y,
pointLightPositions[2].z);
            glUniform3f(glGetUniformLocation(lightingShader.Program,
"pointLights[2].ambient"), 0.05f, 0.05f, 0.05f);
            glUniform3f(glGetUniformLocation(lightingShader.Program,
"pointLights[2].diffuse"), 0.0f, 1.0f, 1.0f);
            glUniform3f(glGetUniformLocation(lightingShader.Program,
"pointLights[2].specular"), 0.0f, 1.0f, 1.0f);
            glUniform1f(glGetUniformLocation(lightingShader.Program,
"pointLights[2].constant"), 1.0f);
            glUniform1f(glGetUniformLocation(lightingShader.Program,
"pointLights[2].linear"), 0.09f);
            glUniform1f(glGetUniformLocation(lightingShader.Program,
"pointLights[2].quadratic"), 0.032f);

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

            // SpotLight
            glUniform3f(glGetUniformLocation(lightingShader.Program,
"spotLight.position"), camera.GetPosition().x, camera.GetPosition().y,
camera.GetPosition().z);
            glUniform3f(glGetUniformLocation(lightingShader.Program,
"spotLight.direction"), camera.GetFront().x, camera.GetFront().y,
camera.GetFront().z);
```

```
            glUniform3f(glGetUniformLocation(lightingShader.Program,
"spotLight.ambient"), 0.0f, 0.0f, 0.0f);
            glUniform3f(glGetUniformLocation(lightingShader.Program,
"spotLight.diffuse"), 0.0f, 0.0f, 0.0f);
            glUniform3f(glGetUniformLocation(lightingShader.Program,
"spotLight.specular"), 0.0f, 0.0f, 0.0f);
            glUniform1f(glGetUniformLocation(lightingShader.Program,
"spotLight.constant"), 1.0f);
            glUniform1f(glGetUniformLocation(lightingShader.Program,
"spotLight.linear"), 0.09f);
            glUniform1f(glGetUniformLocation(lightingShader.Program,
"spotLight.quadratic"), 0.032f);
            glUniform1f(glGetUniformLocation(lightingShader.Program,
"spotLight.cutOff"), glm::cos(glm::radians(12.5f)));
            glUniform1f(glGetUniformLocation(lightingShader.Program,
"spotLight.outerCutOff"), glm::cos(glm::radians(15.0f)));



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

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


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

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

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

            //Carga de modelos
```

Se carga cada uno de los modelos por separado, tomé esta decisión para poder tener más flexibilidad al momento de elegir los elementos que serán animados.

```
            //Casa
            view = camera.GetViewMatrix();
            glm::mat4 model(1);
            tmp = model = glm::translate(model, glm::vec3(1, 1, 1));
```

```
            model = glm::translate(model, PosIni);
            model = glm::scale(model, glm::vec3(2.0f, 2.0f, 2.0f));
            glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
            Casa.Draw(lightingShader);

            //Puerta
            view = camera.GetViewMatrix();
            model = glm::translate(tmp, PosIni);
            model = glm::translate(model, glm::vec3(posPueX, 5.0f, posPueZ));
            model = glm::scale(model, glm::vec3(2.0f, 2.0f, 2.0f));
            model = glm::rotate(model, glm::radians(movPue), glm::vec3(0.0f,
1.0f, 0.0f));
            glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
            Puerta.Draw(lightingShader);

            //Mueble
            view = camera.GetViewMatrix();
            model = glm::translate(tmp, PosIni);
            model = glm::translate(model, glm::vec3(-59.0f, 5.2f, 1.0f));
            model = glm::scale(model, glm::vec3(2.0f, 4.0f, 4.0f));
            glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
            Mueble.Draw(lightingShader);

            //Television
            view = camera.GetViewMatrix();
            model = glm::translate(tmp, PosIni);
            model = glm::translate(model, glm::vec3(-33.0f, 5.2f, 15.5f));
            model = glm::scale(model, glm::vec3(3.5f, 4.0f, 4.0f));
            model = glm::rotate(model, glm::radians(-90.0f), glm::vec3(0.0f,
1.0f, 0.0f));
            glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
            Television.Draw(lightingShader);

            //AntenaIzq
            view = camera.GetViewMatrix();
            model = glm::translate(tmp, PosIni);
            model = glm::translate(model, glm::vec3(-31.0f, posAntIY, 15.5f));
            model = glm::scale(model, glm::vec3(3.5f, 4.0f, 4.0f));
            model = glm::rotate(model, glm::radians(rotAntIzq), glm::vec3(0.0f,
0.0f, 1.0f));
            glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
            Antena.Draw(lightingShader);

            //AntenaDer
            view = camera.GetViewMatrix();
            model = glm::translate(tmp, PosIni);
            model = glm::translate(model, glm::vec3(-34.3f, 22.7f, 15.5f));
            model = glm::scale(model, glm::vec3(3.5f, 4.0f, 4.0f));
            model = glm::rotate(model, glm::radians(rotAntDerZ), glm::vec3(0.0f,
0.0f, 1.0f));
            model = glm::rotate(model, glm::radians(rotAntDerX), glm::vec3(1.0f,
0.0f, 0.0f));
            glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
            Antena.Draw(lightingShader);
```

```
            //Sillon
            view = camera.GetViewMatrix();
            model = glm::translate(tmp, PosIni);
            model = glm::translate(model, glm::vec3(-30.0f, 5.2f, -22.5f));
            model = glm::scale(model, glm::vec3(5.0f, 5.0f, 5.0f));
            model = glm::rotate(model, glm::radians(90.0f), glm::vec3(0.0f,
1.0f, 0.0f));
            glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
            Sillon.Draw(lightingShader);

            //Alfombra
            view = camera.GetViewMatrix();
            model = glm::translate(tmp, PosIni);
            model = glm::translate(model, glm::vec3(-33.0f, 5.0f, -1.5f));
            model = glm::scale(model, glm::vec3(6.5f, 6.0f, 6.5f));
            model = glm::rotate(model, glm::radians(90.0f), glm::vec3(0.0f,
1.0f, 0.0f));
            glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
            Alfombra.Draw(lightingShader);

            //Mesa
            view = camera.GetViewMatrix();
            model = glm::translate(tmp, PosIni);
            model = glm::translate(model, glm::vec3(-44.5f, 5.2f, -24.0f));
            model = glm::scale(model, glm::vec3(3.0f, 3.0f, 3.0f));
            model = glm::rotate(model, glm::radians(-90.0f), glm::vec3(0.0f,
1.0f, 0.0f));
            glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
            Mesa.Draw(lightingShader);

            //Cajon
            view = camera.GetViewMatrix();
            model = glm::translate(tmp, PosIni + glm::vec3(0.0f, 0.0f,
movCajZ));
            model = glm::translate(model, glm::vec3(-44.5f, 13.2f, -24.0f));
            model = glm::scale(model, glm::vec3(2.7f, 3.0f, 3.0f));
            model = glm::rotate(model, glm::radians(0.0f), glm::vec3(0.0f, 1.0f,
0.0f));
            glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
            Cajon.Draw(lightingShader);

            //Cuadro
            view = camera.GetViewMatrix();
            model = glm::translate(tmp, PosIni);
            model = glm::translate(model, glm::vec3(-30.5f, 30.0f, -27.0f));
            model = glm::scale(model, glm::vec3(3.0f, 3.0f, 3.0f));
            model = glm::rotate(model, glm::radians(-90.0f), glm::vec3(0.0f,
1.0f, 0.0f));
            glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
            Cuadro.Draw(lightingShader);

            //Barco
            view = camera.GetViewMatrix();
```

```
            model = glm::translate(tmp, PosIni);
            model = glm::translate(model, glm::vec3(posBarX, 29.0f, -26.9f));
            model = glm::scale(model, glm::vec3(3.0f, 3.0f, 3.0f));
            model = glm::rotate(model, glm::radians(-90.0f), glm::vec3(0.0f,
1.0f, 0.0f));
            model = glm::rotate(model, glm::sin(movBar), glm::vec3(1.0f, 0.0f,
0.0f));
            glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
            Barco.Draw(lightingShader);

            //Libro1
            view = camera.GetViewMatrix();
            model = glm::translate(tmp, PosIni + glm::vec3(movLibX, movLibY,
movLibZ));
            model = glm::translate(model, glm::vec3(-59.0f, 7.9f, 1.0f));
            model = glm::scale(model, glm::vec3(2.0f, 1.5f, 2.0f));
            model = glm::rotate(model, glm::radians(rotLibX), glm::vec3(1.0f,
0.0f, 0.0f));
            model = glm::rotate(model, glm::radians(rotLib), glm::vec3(0.0f,
1.0f, 0.0f));
            glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
            Libro.Draw(lightingShader);

            //Libro2
            view = camera.GetViewMatrix();
            model = glm::translate(tmp, PosIni);
            model = glm::translate(model, glm::vec3(-59.0f, 7.9f, 5.5f));
            model = glm::scale(model, glm::vec3(2.0f, 1.5f, 2.0f));
            model = glm::rotate(model, glm::radians(90.0f), glm::vec3(1.0f,
0.0f, 0.0f));
            model = glm::rotate(model, glm::radians(180.0f), glm::vec3(0.0f,
1.0f, 0.0f));
            model = glm::rotate(model, glm::radians(20.0f), glm::vec3(1.0f,
0.0f, 0.0f));
            glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
            Libro.Draw(lightingShader);

            glBindVertexArray(0);


            // Also draw the lamp object, again binding the appropriate shader
            lampShader.Use();
            // Get location objects for the matrices on the lamp shader (these
could be different on a different shader)
            modelLoc = glGetUniformLocation(lampShader.Program, "model");
            viewLoc = glGetUniformLocation(lampShader.Program, "view");
            projLoc = glGetUniformLocation(lampShader.Program, "projection");

            // Set matrices
            glUniformMatrix4fv(viewLoc, 1, GL_FALSE, glm::value_ptr(view));
            glUniformMatrix4fv(projLoc, 1, GL_FALSE,
glm::value_ptr(projection));
            model = glm::mat4(1);
            model = glm::translate(model, lightPos);
```

```
            //model = glm::scale(model, glm::vec3(0.2f)); // Make it a smaller
cube
            glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
            // Draw the light object (using light's vertex attributes)
            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;
}
```

Esta función es la que se encarga de la lógica de las animaciones implementadas en este proyecto.

```
void animacion()
{
```

La animación del libro consiste en tirarlo del mueble en el que se encuentra; para lograr el efecto primero lo tuve que mover en el eje X para sacarlo del muble, después haca una rotación sobre el eje Z para inmediatamente moverse en el eje Y, y por último hace una última rotación sobre el eje X para terminar tirado en el suelo.

```
	//Movimiento del libro
	if (caidaLibro)
	{

		if (recorrido1)
		{
			movLibX += 0.1f;
			if (movLibX > 6.0f)
			{
				recorrido1 = false;
				recorrido2 = true;

			}
		}

		if (recorrido2)
		{
			rotLib -= 5.0f;
			if (rotLib < 95)
			{
				recorrido2 = false;
				recorrido3 = true;

			}
		}

		if (recorrido3)
		{
			movLibY -= 0.15f;
			if (movLibY< -0.9)
			{
				recorrido3 = false;
				recorrido4 = true;
			}
		}

		if (recorrido4)
		{
			rotLibX -= 5.0f;
			if (rotLibX < 5)
			{
				recorrido4 = false;
				recorrido5 = true;
```

```
			}
		}

		if (recorrido5)
		{
			movLibY -= 0.15f;
			movLibZ -= 0.15f;
			if (movLibY < -2.5 and movLibZ < -1.0)
			{
				recorrido5 = false;
				caidaLibro = false;
			}
		}
	}
```

La animación del cajón es bastante sencilla; solo se mueve sobre el eje Z y se delimitan las distancias para que pareciera que entra y sale de la mesa de teléfono.

```
		//Movimiento del cajon
		if (abrirCerrarCajon)
		{

			if (recorridoCaj1)
			{
				movCajZ -= 0.05f;
				if (movCajZ < -1.0f)
				{
					recorridoCaj1 = false;
					recorridoCaj2 = true;

				}
			}

			if (recorridoCaj2)
			{
				movCajZ += 0.05f;
				if (movCajZ > 3.0f)
				{
					recorridoCaj2 = false;
					recorridoCaj3 = true;

				}
			}

			if (recorridoCaj3)
			{
				movCajZ -= 0.05f;
				if (movCajZ < -1.5f)
				{
					recorridoCaj3 = false;
					recorridoCaj1 = true;

				}
			}
```

```
}
```

La puerta tiene que hacer una rotación sobre el eje Y para que parezca que se abre, sin embargo, el efecto no está presente; para lograr que aparezca el efecto opté por incrementar el valor de la posición del objeto en X y decrementar Z y viceversa mientras la puerta hace la rotación simultáneamente.

```
//Movimiento de la puerta
if (abrirCerrarPue)
{

	if (recorridoPue1)
	{
		movPue += 0.1f;
		posPueX += 0.004f;
		posPueZ -= 0.004f;
		if (movPue > 165.0f)
		{
			recorridoPue1 = false;
			recorridoPue2 = true;

		}
	}

	if (recorridoPue2)
	{
		movPue -= 0.1f;
		posPueX -= 0.004f;
		posPueZ += 0.004f;
		if (movPue < 90.0f)
		{
			recorridoPue2 = false;
			recorridoPue1 = true;

		}
	}
}
```

Para lograr el efecto de un barco navegando utilicé la función seno para hacer las rotaciones en un intervalo de -1 a 1, y para hacer que el barco se mueva dentro del marco sin que afecté la rotación se activa otros dos eventos simultáneos a los utilizados para la rotación.

```
//Movimiento del barco
if (moverBarco)
{

	if (recorridoBar1)
	{
		movBar += 0.001f;
		if (movBar > 0.1f)
		{
			recorridoBar1 = false;
			recorridoBar2 = true;
```

```
			}
		}
		if (recorridoBar3)
		{
			posBarX += 0.005f;
			if (posBarX > -29.0f)
			{
				recorridoBar3 = false;
				recorridoBar4 = true;

			}
		}


		if (recorridoBar2)
		{
			movBar -= 0.001f;
			if (movBar < -0.1f)
			{
				recorridoBar2 = false;
				recorridoBar1 = true;

			}
		}
		if (recorridoBar4)
		{
			posBarX -= 0.005f;
			if (posBarX < -32.0f)
			{
				recorridoBar4 = false;
				recorridoBar3 = true;

			}
		}
	}
```

La antena izquierda hace un movimiento muy simple, solo es una rotación en el eje Z.

```
	//Movimiento de la antena izquierda
	if (moverAntenaIzq)
	{

		if (recorridoIzq1)
		{
			rotAntIzq -= 0.1f;
			if (rotAntIzq < -70.0f)
			{
				recorridoIzq1 = false;
				recorridoIzq2 = true;

			}
		}
```

```
	if (recorridoIzq2)
	{
		rotAntIzq += 0.1f;
		if (rotAntIzq > -20.0f)
		{
			recorridoIzq2 = false;
			recorridoIzq1 = true;
		}
	}
}
```

La animación de la antena utiliza una lógica similar a la del barco para lograr 2 tipos de movimiento a la vez, en este caso se hacen 2 rotaciones simultaneas, una en el eje X y otra en el eje Z.

```
if (moverAntenaDer)
{

	if (recorridoDer1)
	{
		rotAntDerZ -= 0.1f;
		if (rotAntDerZ < 0.0f)
		{
			recorridoDer1 = false;
			recorridoDer2 = true;
		}
	}

	if (recorridoDer3)
	{
		rotAntDerX -= 0.1f;
		if (rotAntDerX < -15.0f)
		{
			recorridoDer3 = false;
			recorridoDer4 = true;
		}
	}

	if (recorridoDer2)
	{
		rotAntDerZ += 0.1f;
		if (rotAntDerZ > 90.0f)
		{
			recorridoDer2 = false;
			recorridoDer1 = true;
		}
	}

	if (recorridoDer4)
	{
		rotAntDerX += 0.1f;
		if (rotAntDerX > 15.0f)
		{
			recorridoDer4 = false;
			recorridoDer3 = true;
```

```
                        }
                }
        }
}


// Is called whenever a key is pressed/released via GLFW
void KeyCallback(GLFWwindow *window, int key, int scancode, int action, int mode)
{
	if (GLFW_KEY_ESCAPE == key && GLFW_PRESS == action)
	{
		glfwSetWindowShouldClose(window, GL_TRUE);
	}

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

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

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

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

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

	lastX = xPos;
	lastY = yPos;

	camera.ProcessMouseMovement(xOffset, yOffset);
```

```
}

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

	// Camera controls
	if (keys[GLFW_KEY_W] || keys[GLFW_KEY_UP])
	{
		camera.ProcessKeyboard(FORWARD, 0.05f);

	}

	if (keys[GLFW_KEY_S] || keys[GLFW_KEY_DOWN])
	{
		camera.ProcessKeyboard(BACKWARD, 0.05f);


	}

	if (keys[GLFW_KEY_A] || keys[GLFW_KEY_LEFT])
	{
		camera.ProcessKeyboard(LEFT, 0.05f);


	}

	if (keys[GLFW_KEY_D] || keys[GLFW_KEY_RIGHT])
	{
		camera.ProcessKeyboard(RIGHT, 0.05f);
	}
```

Se establece que para iniciar las animaciones se presiona la tecla I, para pausar la tecla O y para reiniciar las posiciones y pausar la tecla R.

```
	if (keys[GLFW_KEY_I])
	{
		caidaLibro = true;
		abrirCerrarCajon = true;
		abrirCerrarPue = true;
		moverBarco = true;
		moverAntenaIzq = true;
		moverAntenaDer = true;
	}

	if (keys[GLFW_KEY_O])
	{
		caidaLibro = false;
		abrirCerrarCajon = false;
		abrirCerrarPue = false;
		moverBarco = false;
		moverAntenaIzq = false;
		moverAntenaDer = false;
	}
```

```
if (keys[GLFW_KEY_R])
{
	caidaLibro = false;
	abrirCerrarCajon = false;
	abrirCerrarPue = false;
	moverBarco = false;
	moverAntenaIzq = false;
	moverAntenaDer = false;

	//Animación del coche
	movLibX = 0.0;
	movLibY = 0.0;
	movLibZ = 0.0;
	rotLib = 180.0;
	rotLibX = 90.0;

	caidaLibro = false;
	recorrido1 = true;
	recorrido2 = false;
	recorrido3 = false;
	recorrido4 = false;
	recorrido5 = false;

	//Animación del cajón
	movCajZ = 0.0;

	abrirCerrarCajon = false;
	recorridoCaj1 = true;
	recorridoCaj2 = false;
	recorridoCaj3 = false;

	//Animación de la puerta
	movPue = 90.0;
	posPueX = -61.5;
	posPueZ = -21.3;

	abrirCerrarPue = false;
	recorridoPue1 = true;
	recorridoPue2 = false;

	//Animación del barco
	movBar = 0.0;
	posBarX = -30.5f;

	moverBarco = false;
	recorridoBar1 = true;
	recorridoBar2 = false;
	recorridoBar3 = true;
	recorridoBar4 = false;

	//Animación de la antena izquierda
	rotAntIzq = -45.0;
	posAntIY = 23.9;
```

```
            moverAntenaIzq = false;
            recorridoIzq1 = true;
            recorridoIzq2 = false;


            //Animación de la antena derecha
            rotAntDerZ = 90.0;
            rotAntDerX = 0.0;

            moverAntenaDer = false;
            recorridoDer1 = true;
            recorridoDer2 = false;
            recorridoDer3 = true;
            recorridoDer4 = false;
      }

}
```