Facultad de Ingeniería

Proyecto final de Computación Gráfica e Interacción Humano Computadora

Casa de Marceline, de la serie animada Hora de aventura

Manual técnico

Número de cuenta: 316170040. Grupo: 12

Índice.

Objetivo.

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

Alcance del proyecto.

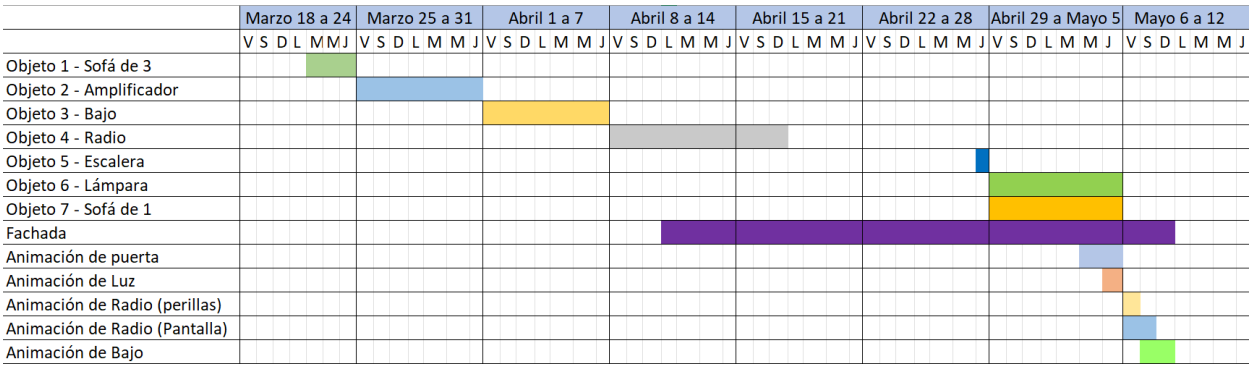
Hacer una recreación en 3D de la fachada elegida con 7 objetos pertenecientes al espacio mencionado y llevarlos a OpenGL para realizar 5 animaciones (3 sencillas y 2 complejas), aplicando lo aprendido en el curso en un tiempo aproximado de 3 meses.

Limitantes.

No existirán colisiones.

No pueden utilizarse métodos de modelado ni animación distintos a los vistos en el curso.

Diagrama de Gantt.



Documentación del código fuente.

Para facilitar la compresión del código fuente, tendremos una sección donde aparezca el código fuente completo (con los comentarios del código) y una parte dedicada a documentar las partes que fueron añadidas al código base.

Código fuente:

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

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

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

// Other Libs
#include "stb_image.h"

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

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

```
#include <glm/gtc/type_ptr.hpp>

//Carga modelos
#include "SOIL2/SOIL2.h"

// Declaración de funciones
void KeyCallback(GLFWwindow* window, int key, int scancode, int action, int mode);
void MouseCallback(GLFWwindow* window, double xPos, double yPos);
void DoMovement();
void animacion_bajo();

// Dimensiones de la ventana
const GLuint WIDTH = 800, HEIGHT = 600;
int SCREEN_WIDTH, SCREEN_HEIGHT;

// Cámara
Camera  camera(glm::vec3(5.0f, 5.0f, 20.0f));
GLfloat lastX = WIDTH / 2.0;
GLfloat lastY = HEIGHT / 2.0;
bool keys[1024];
bool firstMouse = true;

// Atributos de la luz
glm::vec3 lightPos(.0f, .0f, 0.0f);

//Variables de animaciones

//Lámpara//
bool activeL = false; //Booleano que funciona como switch para la animación de la lámpara.
glm::vec3 lampOn;              //Es un vec3 con los colores que produce la luz. Es la consecuencia del bool anterior.
glm::vec3 pointLightPositions(0.0f, 8.2f, 0.0f); // Posición de la pointlight (en la lámpara)
glm::vec3 Light1 = glm::vec3(0); //Luz de la pointlight

//Puerta del frente
bool activePF = false; //Booleano que funciona como switch para la animación de la puerta.
float rotPF = 0.f;     //Variable usada en el rotate de la puerta para producir la animación.
bool dirPF = false;    //Booleano que decide el sentido en que se moverá la puerta.

//Radio
bool activeR = false;          //Booleano que funciona como switch para la animación de la radio.
bool dirRI = false;            //Booleano que decide el sentido en que se moverá la perilla izquierda del radio.
bool dirRD = false;            //Booleano que decide el sentido en que se moverá la perilla derecha del radio.
float rotRI = 0.f;             //Ángulo de rotación de la perilla izquierda del radio.
float rotRD = 0.f;             //Ángulo de rotación de la perilla derecha del radio.
float tiempo;                       //Variable independiente en las función seno del archivo anim.vs

//Bajo
glm::vec3 PosIni(-3.95f, .55f, 0.0f); //Posición inicial del objeto para iniciar la animación. Punto de origen para el objeto
float movKitX = 0.0;           //Variable que se sumará a su posición en el eje X.
float movKitY = 0.0;           //Variable que se sumará a su posición en el eje Y.
float movKitZ = 0.0;           //Variable que se sumará a su posición en el eje Z.
float rotKitX = 0.0;           //Variable que hará rotar al objeto en el eje X.
float rotKitY = 0.0;//Variable que hará rotar al objeto en el eje Y.
float rotKitZ = 0.0;           //Variable que hará rotar al objeto en el eje Z.

bool circuito = false;         //Booleano que funciona como switch para la animación del bajo. Permite que el recorrido se lleve a
cabo.
bool recorrido1 = true;             //Paso 1 en el recorrido.
bool recorrido2 = false;       //Paso 2 en el recorrido.
bool recorrido3 = false;       //Paso 3 en el recorrido.
bool recorrido4 = false;       //Paso 4 en el recorrido.
bool recorrido5 = false;       //Paso 5 en el recorrido.
bool recorrido6 = false;       //Paso 6 en el recorrido.
bool recorrido7 = false;       //Paso 7 en el recorrido.
bool recorrido8 = false;       //Paso 8 en el recorrido.
bool recorrido9 = false;       //Paso 9 en el recorrido.
```

```
bool recorrido10 = false;     //Paso 10 en el recorrido.
////////////////////////////////////////////////////////////////////////////////////////

float vertices[] = {
      -0.5f, -0.5f, -0.5f,  0.0f,  0.0f, -1.0f,
             0.5f, -0.5f, -0.5f,  0.0f,  0.0f, -1.0f,
             0.5f,  0.5f, -0.5f,  0.0f,  0.0f, -1.0f,
             0.5f,  0.5f, -0.5f,  0.0f,  0.0f, -1.0f,
       -0.5f,  0.5f, -0.5f,  0.0f,  0.0f, -1.0f,
       -0.5f, -0.5f, -0.5f,  0.0f,  0.0f, -1.0f,

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

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

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

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

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




// Deltatime
GLfloat deltaTime = 0.0f;     // Tiempo entre el último frame y el actual
GLfloat lastFrame = 0.0f;     // Tiempo del último frame


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

      //Crea un objeto GLFWwindow que se pueda usar para las funciones de GLFW
      GLFWwindow* window = glfwCreateWindow(WIDTH, HEIGHT, "Proyecto Final", 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);

//Establece las funciones de callback necesarias
glfwSetKeyCallback(window, KeyCallback);
glfwSetCursorPosCallback(window, MouseCallback);

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

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

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


//Declaración de funciones de tipo Shader
Shader lightingShader("Shaders/lighting.vs", "Shaders/lighting.frag");
Shader lampShader("Shaders/lamp.vs", "Shaders/lamp.frag");
Shader Anim("Shaders/anim.vs", "Shaders/anim.frag");

//Definición de los modelos

//Fachada
Model Casa((char*)"Models/casaMarceline/casa.obj");
Model VentanaPF((char*)"Models/casaMarceline/vpf.obj");
Model VentanaPL((char*)"Models/casaMarceline/vpl.obj");
Model PuertaF((char*)"Models/casaMarceline/puertaFrente.obj");
Model PuertaL((char*)"Models/casaMarceline/puertaLado.obj");
Model PuertaP((char*)"Models/casaMarceline/puertaPerro.obj");
Model VentanaF((char*)"Models/casaMarceline/ventanaFrente.obj");
Model VentanaL((char*)"Models/casaMarceline/ventanaLado.obj");

//Objetos
//Declarados de tipo Model para poder traer los OBJ a OpenGL
Model Bajo((char*)"Models/bajoMarceline/bajoMarceline.obj");
Model Amp((char*)"Models/ampMarceline/ampMarceline.obj");
Model Escalera((char*)"Models/escaleraMarceline/escaleraMarceline.obj");
Model LamparaMetal((char*)"Models/lamparaMarceline/lamparaMarcelineMetal.obj");
Model Lampara((char*)"Models/lamparaMarceline/lamparaMarceline.obj");
Model Radio((char*)"Models/radioMarceline/radioMarceline.obj");
Model RadioPantalla((char*)"Models/radioMarceline/pantalla.obj");
Model RadioPI((char*)"Models/radioMarceline/perillaI.obj");
Model RadioPD((char*)"Models/radioMarceline/perillaD.obj");
Model Sofa3((char*)"Models/sofaMarceline/sofaMarceline.obj");
Model Sofa1((char*)"Models/sofaMarceline1/sofaMarceline1.obj");


//Configuración del VAO y VBO
GLuint VBO, VAO;
glGenVertexArrays(1, &VAO);
glGenBuffers(1, &VBO);
glBindVertexArray(VAO);
glBindBuffer(GL_ARRAY_BUFFER, VBO);
```

```
    glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW);

    //Atributos de posición
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(GLfloat), (GLvoid*)0);
    glEnableVertexAttribArray(0);
    //Atributos de la normal
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(float), (void*)(3 * sizeof(float)));
    glEnableVertexAttribArray(1);

    //Configuración de las unidades de textura
    lightingShader.Use();
    glUniform1i(glGetUniformLocation(lightingShader.Program, "material.diffuse"), 0);
    glUniform1i(glGetUniformLocation(lightingShader.Program, "material.specular"), 1);

    //Proyección en perspectiva
    glm::mat4 projection = glm::perspective(camera.GetZoom(), (GLfloat)SCREEN_WIDTH / (GLfloat)SCREEN_HEIGHT, 0.1f, 100.0f);

    //Configuración cíclica donde se establecen los atributos de luz y modelos. Se detiene cuando el usuario lo decida.
    while (!glfwWindowShouldClose(window)) {

        //DeltaTime del frame actual
        GLfloat currentFrame = glfwGetTime();
        deltaTime = currentFrame - lastFrame;
        lastFrame = currentFrame;

        //Revisa si ha ocurrido algún evento (como presionar alguna tecla o movimiento del ratón)
        //Mantiene en loop a las animaciones (animacíon())
        glfwPollEvents();
        DoMovement();
        animacion_bajo();

        //Limpia el colorbuffer
        glClearColor(0.1f, 0.1f, 0.1f, 1.0f);
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

        //Habilita opciones de openGL
        glEnable(GL_DEPTH_TEST);

        //Usa el shader correspondiente al configurar objetos uniforms/drawing
        lightingShader.Use();
        GLint viewPosLoc = glGetUniformLocation(lightingShader.Program, "viewPos");
        glUniform3f(viewPosLoc, camera.GetPosition().x, camera.GetPosition().y, camera.GetPosition().z);

        //Iluminación
        //Directional light
        glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.direction"), -0.2f, -1.0f, -0.3f);
        glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.ambient"), 0.4f, 0.4f, 0.4f);
        glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.diffuse"), 0.1f, 0.1f, 0.1f);
        glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.specular"), 1.f, 1.f, 1.f);

        //Point light
        glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].position"), pointLightPositions.x,
pointLightPositions.y, pointLightPositions.z);
        glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].ambient"), lampOn.x, lampOn.y, lampOn.z);
        glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].diffuse"), lampOn.x, lampOn.y, lampOn.z);
        glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].specular"), 1.0f, 1.0f, 0.0f);
        glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[0].constant"), 1.0f);
        glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[0].linear"), 0.35f);
        glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[0].quadratic"), .44f);

        // 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"), 1.0f, 1.0f, 1.0f);
```

```
glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight.diffuse"), 1.0f, 1.0f, 1.0f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight.specular"), 1.0f, 1.0f, 1.0f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight.constant"), 1.0f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight.linear"), 0.35f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight.quadratic"), .44f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight.cutOff"), glm::cos(glm::radians(12.5f)));
glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight.outerCutOff"), glm::cos(glm::radians(15.0f)));

//Configura las propiedades del material
glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 16.0f);

//Crea la transformación de la cámara
glm::mat4 view;
view = camera.GetViewMatrix();

//Obtiene la ubicación de las uniforms
GLint modelLoc = glGetUniformLocation(lightingShader.Program, "model");
GLint viewLoc = glGetUniformLocation(lightingShader.Program, "view");
GLint projLoc = glGetUniformLocation(lightingShader.Program, "projection");

//Pasa de matrices a shaders
glUniformMatrix4fv(viewLoc, 1, GL_FALSE, glm::value_ptr(view));
glUniformMatrix4fv(projLoc, 1, GL_FALSE, glm::value_ptr(projection));

//Carga de modelo
glm::mat4 model(1);
view = camera.GetViewMatrix();

//Fachada
model = glm::mat4(1);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
Casa.Draw(lightingShader);

//Puerta del frente
model = glm::mat4(1);
model = translate(model, glm::vec3(1.05f, 3.5f, 9.f));
model = glm::rotate(model, glm::radians(rotPF), glm::vec3(.0f, 1.f, 0.f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
PuertaF.Draw(lightingShader);

//Puerta lateral
model = glm::mat4(1);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
PuertaL.Draw(lightingShader);

//Puerta para el perro
model = glm::mat4(1);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
PuertaP.Draw(lightingShader);

//Objetos
                        //Amplificador
model = glm::mat4(1);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
Amp.Draw(lightingShader);

//Bajo
model = glm::mat4(1);
model = glm::translate(model, PosIni + glm::vec3(movKitX, movKitY, movKitZ));
model = glm::rotate(model, glm::radians(rotKitZ), glm::vec3(0.0f, 0.0f, 1.0));
model = glm::rotate(model, glm::radians(rotKitX), glm::vec3(1.0f, 0.0f, 0.0));
```

```
model = glm::rotate(model, glm::radians(rotKitY), glm::vec3(0.0f, 1.0f, 0.0));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
Bajo.Draw(lightingShader);

//Escalera
model = glm::mat4(1);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
Escalera.Draw(lightingShader);

//Parte metálica (no transparente) de la lámpara
model = glm::mat4(1);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
LamparaMetal.Draw(lightingShader);

//RADIO
model = glm::mat4(1);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
Radio.Draw(lightingShader);

//Perilla izquierda
model = glm::mat4(1);
model = translate(model, glm::vec3(-4.42f, 1.1f, 5.6f));
model = glm::rotate(model, glm::radians(44.f), glm::vec3(0.0f, 1.f, 0.f));
model = glm::rotate(model, glm::radians(rotRI), glm::vec3(1.0f, 0.f, 0.f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
RadioPI.Draw(lightingShader);

//Perilla derecha
model = glm::mat4(1);
model = translate(model, glm::vec3(-4.9f, 1.1f, 5.1f));
model = glm::rotate(model, glm::radians(44.f), glm::vec3(0.0f, 1.f, 0.f));
model = glm::rotate(model, glm::radians(rotRD), glm::vec3(1.0f, 0.f, 0.f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
RadioPD.Draw(lightingShader);

//Sofá para 1
model = glm::mat4(1);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
Sofa1.Draw(lightingShader);

//Sofá para 3
model = glm::mat4(1);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
Sofa3.Draw(lightingShader);

//OBJETOS TRANSPARENTES

glEnable(GL_BLEND);//Avtiva la funcionalidad para trabajar el canal alfa
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);

//Ventanas de la puerta del frente
model = glm::mat4(1);
model = translate(model, glm::vec3(1.05f, 3.5f, 9.f));
model = glm::rotate(model, glm::radians(rotPF), glm::vec3(.0f, 1.f, 0.f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 1);
glUniform4f(glGetUniformLocation(lightingShader.Program, "colorAlfa"), 0.0, 0.215, 0.29, 0.99);
VentanaPF.Draw(lightingShader);
```

```
//Ventanas de la puerta lateral
model = glm::mat4(1);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 1);
glUniform4f(glGetUniformLocation(lightingShader.Program, "colorAlfa"), 0.0, 0.215, 0.29, 0.99);
VentanaPL.Draw(lightingShader);

//Ventana frontal
model = glm::mat4(1);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 1);
glUniform4f(glGetUniformLocation(lightingShader.Program, "colorAlfa"), 0.0, 0.215, 0.29, 0.99);
VentanaF.Draw(lightingShader);

//Ventana lateral
model = glm::mat4(1);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 1);
glUniform4f(glGetUniformLocation(lightingShader.Program, "colorAlfa"), 0.0, 0.215, 0.29, 0.99);
VentanaL.Draw(lightingShader);

//Parte lumínica de la lámpara (cristal)
model = glm::mat4(1);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 1);
glUniform4f(glGetUniformLocation(lightingShader.Program, "colorAlfa"), 1.0, 1.0, 1.0, 0.99);
Lampara.Draw(lightingShader);

glDisable(GL_BLEND);  //Desactiva el canal alfa
glUniform4f(glGetUniformLocation(lightingShader.Program, "colorAlfa"), 1.0, 1.0, 1.0, 1.0); //Reset para el resto de cosas

//Pantalla de la radio
Anim.Use();
modelLoc = glGetUniformLocation(Anim.Program, "model");
viewLoc = glGetUniformLocation(Anim.Program, "view");
projLoc = glGetUniformLocation(Anim.Program, "projection");
glUniformMatrix4fv(viewLoc, 1, GL_FALSE, glm::value_ptr(view));
glUniformMatrix4fv(projLoc, 1, GL_FALSE, glm::value_ptr(projection));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
model = glm::mat4(1);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1f(glGetUniformLocation(Anim.Program, "time"), tiempo);
RadioPantalla.Draw(Anim);

glBindVertexArray(0);

//Fin de la transparencia

                        //Se vuelve a dibujar el objeto lamp para el shader apropiado
lampShader.Use();
//Obtiene objetos d eubicación para las matrices en el lampshader
modelLoc = glGetUniformLocation(lampShader.Program, "model");
viewLoc = glGetUniformLocation(lampShader.Program, "view");
projLoc = glGetUniformLocation(lampShader.Program, "projection");

//Configura las 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));
//Dibuja el objeto de luz
model = glm::mat4(1);
model = glm::translate(model, pointLightPositions);
```

```
		model = glm::scale(model, glm::vec3(0.2f)); // Make it a smaller cube
		glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
		glBindVertexArray(VAO);
		glDrawArrays(GL_TRIANGLES, 0, 36);
		glBindVertexArray(0);

		//Intercambia los búfferes de pantalla
		glfwSwapBuffers(window);
	}

	//Termina GLFW. Limpia todos los recursos utilizados por GLFW
	glfwTerminate();

	return 0;
}

// Moves/alters the camera positions based on user input
void DoMovement() {
	//Controles de la cámara
	if (keys[GLFW_KEY_W]) {
		camera.ProcessKeyboard(FORWARD, deltaTime * 2);
	}

	if (keys[GLFW_KEY_S] || keys[GLFW_KEY_DOWN]) {
		camera.ProcessKeyboard(BACKWARD, deltaTime * 2);
	}

	if (keys[GLFW_KEY_A] || keys[GLFW_KEY_LEFT]) {
		camera.ProcessKeyboard(LEFT, deltaTime * 2);
	}

	if (keys[GLFW_KEY_D] || keys[GLFW_KEY_RIGHT]) {
		camera.ProcessKeyboard(RIGHT, deltaTime * 2);
	}

	//Abre y cierra la puerta
	if (rotPF >= 85.f) {
		dirPF = false;
	}
	else if (rotPF <= 0.f) {
		dirPF = true;
	}

	if (activePF) {
		if (dirPF)
			rotPF += 1.f;
	}
	else {
		if (!dirPF)
			rotPF -= 1.f;
	}

	//Enciende // apaga la radio
	if (activeR) {
		tiempo = glfwGetTime();

		if (rotRI >= 180.f)
			dirRI = false;
		else if (rotRI <= 0.f)
			dirRI = true;

		if (dirRI)
			rotRI += 0.5f;
		else
			rotRI -= 0.8f;
```

```
		if (rotRD >= 85.f)
			dirRD = false;
		else if (rotRD <= 0.f)
			dirRD = true;

		if (dirRD)
			rotRD += 1.5f;
		else
			rotRD -= 1.f;
	}
	else {
		tiempo = 0.f;
	}
}

void animacion_bajo() {
	//Movimiento del bajo
	if (circuito) {
		if (recorrido1) {
			rotKitZ += 0.08 + (rotKitZ / 10);
			if (rotKitZ > 13.6) {
				recorrido1 = false;
				recorrido2 = true;
			}
		}
		if (recorrido2) {
			rotKitY -= 2.88 - (rotKitY / 20);
			if (rotKitY < -15.1) {
				recorrido2 = false;
				recorrido3 = true;
			}
		}

		if (recorrido3) {
			if (movKitX < 0.911)
				movKitX += (0.05061111111112 - movKitX / 20);
			rotKitZ += (0.9969444444444444445 - rotKitZ / 50);
			if (rotKitZ > 32.5) {
				recorrido3 = false;
				recorrido4 = true;
			}
		}

		if (recorrido4) {
			movKitX += 0.035;
			rotKitZ -= (0.25 + rotKitZ / 130);
			if (rotKitZ < 31.0) {
				recorrido4 = false;
				recorrido5 = true;
			}
		}

		if (recorrido5) {
			movKitX += 0.0083;
			rotKitZ += (0.05 + (rotKitZ / 120));
			if (rotKitZ > 38.8) {
				recorrido5 = false;
				recorrido6 = true;
			}
		}

		if (recorrido6) {
			rotKitZ += (0.05 + (rotKitZ / 120));
			if (rotKitZ > 40.) {
				recorrido6 = false;
				recorrido7 = true;
```

```
			}
		}

		if (recorrido7) {
			if (movKitX <= 1.9)
				movKitX += 0.005 + (movKitX / 40);
			rotKitZ += (0.05 + (rotKitZ / 40));
			if (rotKitZ > 72.) {
				recorrido7 = false;
				recorrido8 = true;
			}
		}

		if (recorrido8) {
			rotKitZ = 72.8;
			recorrido8 = false;
			recorrido9 = true;
			circuito = !circuito;
		}

		if (recorrido9) {
			rotKitY = -13.8;
			recorrido9 = false;
			recorrido10 = true;
		}

		if (recorrido10 && circuito) {
			movKitX = 0.0;
			rotKitY = 0.0;
			rotKitZ = 0.0;
			recorrido10 = false;
			recorrido1 = true;
		}
	}
}

//Se llama cuando alguna tecla es presionada vía 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_L]) {
		activeL = !activeL;
		if (activeL) {
			lampOn.x = 1.0f;
			lampOn.y = 1.0f;
			lampOn.z = 0.717647f;
			Light1 = glm::vec3(1.0f, 1.0f, .0f);
		}
		else {
			lampOn = glm::vec3(0.0f);
			Light1 = glm::vec3(0);//Cuado es solo un valor en los 3 vectores pueden dejar solo una componente
		}
	}

	if (keys[GLFW_KEY_P]) {
```

```
            activePF = !activePF;
        }

        if (keys[GLFW_KEY_R]) {
            activeR = !activeR;
        }

        if (keys[GLFW_KEY_B]) {
            circuito = !circuito;
        }
    }

    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);
    }
```

Código añadido al código base:

Las siguientes líneas del código, representan el código añadido al código base para la correcta visualización de la recreación en 3D de la fachada seleccionada. Este código es comentado a detalle para entender el correcto funcionamiento de las animaciones y la carga de los modelos.

Código de la carga de modelos (función main()).

```
//Fachada
        model = glm::mat4(1);
        glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
        glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
        Casa.Draw(lightingShader);
```

En este modelo, que compone la mayor parte de la fachada (muros, marcos de puerta y ventanas, objetos estáticos que forman parte de la fachada) se carga el modelo sin cambiar su posición ni rotarlo y con la transparencia desactivada.

```
        //Puerta del frente
        model = glm::mat4(1);
        model = translate(model,glm::vec3(1.05f,3.5f,9.f));
        model = glm::rotate(model, glm::radians(rotPF), glm::vec3(.0f, 1.f, 0.f));
        glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
        glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
        PuertaF.Draw(lightingShader);
```

Se carga el modelo de la puerta frontal, sin sus ventanas, con una translación hacia su posición y con la rotación activada, la cual será según lo marque la variable “rotPF”.

```
        //Puerta lateral
        model = glm::mat4(1);
        glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
        glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
        PuertaL.Draw(lightingShader);
```

Se carga el modelo de la puerta lateral, sin sus ventanas, sin transformaciones y con la transparencia desactivada.

```
        //Puerta para el perro
        model = glm::mat4(1);
        glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
        glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
        PuertaP.Draw(lightingShader);
```

Se carga el modelo de la puerta para el perro (que está en la puerta lateral), sin transformaciones y con la transparencia desactivada.

```
//Objetos
        //Amplificador
        model = glm::mat4(1);
        glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
        glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
        Amp.Draw(lightingShader);
```

Se carga el modelo del amplificador sin transformaciones y con la transparencia desactivada.

```
//Bajo
model = glm::mat4(1);
model = glm::translate(model, PosIni + glm::vec3(movKitX, movKitY, movKitZ));
model = glm::rotate(model, glm::radians(rotKitZ), glm::vec3(0.0f, 0.0f, 1.0));
model = glm::rotate(model, glm::radians(rotKitX), glm::vec3(1.0f, 0.0f, 0.0));
model = glm::rotate(model, glm::radians(rotKitY), glm::vec3(0.0f, 1.0f, 0.0));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
Bajo.Draw(lightingShader);
```

Se carga el modelo del bajo, sin transparencia activada. El bajo se traslada a su posición origen con la variable PosIni (vec3), y a esta posición se le suma un vec3 que altera la posición en todas las direcciones. Las variables “movKitX” (float), “movKitY” (float) y “movKitZ” (float) son alteradas en la función “animacion_bajo()” para alterar la posición del bajo. Además, tiene 3 transformaciones de rotación (cada una en direcciones distintas), donde la variable “rotKitZ” (float) afecta la rotación del bajo en el eje Z, la variable “rotKitX” (float) afecta la rotación del bajo en el eje X y la variable “rotKitY” (float) afecta la rotación del bajo en el eje Y, todas modificadas desde la función “animacion_bajo()”, la cual se encuentra detallada en la sección <<Código de la función “animacion_bajo”>>.

```
//Escalera
model = glm::mat4(1);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
Escalera.Draw(lightingShader);
```

Se carga el modelo de la escalera sin transformaciones y con la transparencia desactivada.

```
//Parte metálica (no transparente) de la lámpara
model = glm::mat4(1);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
LamparaMetal.Draw(lightingShader);
```

Se carga el modelo de la base de la lámpara sin transformaciones y con la transparencia desactivada.

```
//RADIO
model = glm::mat4(1);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
Radio.Draw(lightingShader);
```

Se carga el modelo base de la radio, sin las perillas ni la pantalla, y sin transformaciones y con la transparencia desactivada.

```
//Perilla izquierda
model = glm::mat4(1);
model = translate(model, glm::vec3(-4.42f, 1.1f, 5.6f));
model = glm::rotate(model, glm::radians(44.f), glm::vec3(0.0f, 1.f, 0.f));
model = glm::rotate(model, glm::radians(rotRI), glm::vec3(1.0f, 0.f, 0.f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
RadioPI.Draw(lightingShader);
```

Se carga el modelo de la perilla izquierda sin transparencia, la cual es trasladada a su posición y con dos rotaciones: la primera se encarga de darle la misma inclinación que tiene la radio en el espacio, la segunda es para su animación de giro, la cual es controlada por la variable rotRI. Este movimiento es explicado en la sección “Código – animaciones Radio”.

```
//Perilla derecha
model = glm::mat4(1);
model = translate(model, glm::vec3(-4.9f, 1.1f, 5.1f));
model = glm::rotate(model, glm::radians(44.f), glm::vec3(0.0f, 1.f, 0.f));
model = glm::rotate(model, glm::radians(rotRD), glm::vec3(1.0f, 0.f, 0.f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
RadioPD.Draw(lightingShader);
```

Se carga el modelo de la perilla derecha sin transparencia, la cual es trasladada a su posición y con dos rotaciones: la primera se encarga de darle la misma inclinación que tiene la radio en el espacio, la segunda es para su animación de giro, la cual es controlada por la variable rotRD. Este movimiento es explicado en la sección “Código – animaciones Radio”.

```
//Sofá para 1
model = glm::mat4(1);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
Sofa1.Draw(lightingShader);
```

Se carga el modelo del sofá para 1, sin transformaciones y con la transparencia desactivada.

```
//Sofá para 3
model = glm::mat4(1);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
Sofa3.Draw(lightingShader);
```

Se carga el modelo del sofá para 3, sin transformaciones y con la transparencia desactivada.

```
//OBJETOS TRANSPARENTES
```

```
//Ventanas de la puerta del frente
model = glm::mat4(1);
model = translate(model, glm::vec3(1.05f, 3.5f, 9.f));
model = glm::rotate(model, glm::radians(rotPF), glm::vec3(.0f, 1.f, 0.f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 1);
glUniform4f(glGetUniformLocation(lightingShader.Program, "colorAlfa"), 0.0, 0.215, 0.29, 0.99);
VentanaPF.Draw(lightingShader);
```

Se carga el modelo de las ventanas de la puerta frontal, con una traslación hacia su ubicación y una rotación que sigue el movimiento de la puerta, con la transparencia activada.

```
//Ventanas de la puerta lateral
model = glm::mat4(1);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 1);
glUniform4f(glGetUniformLocation(lightingShader.Program, "colorAlfa"), 0.0, 0.215, 0.29, 0.99);
VentanaPL.Draw(lightingShader);
```

Se carga el modelo de las ventanas de la puerta lateral, con una traslación hacia su ubicación, con la transparencia activada.

```
//Ventana frontal
model = glm::mat4(1);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 1);
glUniform4f(glGetUniformLocation(lightingShader.Program, "colorAlfa"), 0.0, 0.215, 0.29, 0.99);
VentanaF.Draw(lightingShader);
```

Se carga el modelo de la ventana frontal, sin transformaciones y con la transparencia activada.

```
//Ventana lateral
model = glm::mat4(1);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 1);
glUniform4f(glGetUniformLocation(lightingShader.Program, "colorAlfa"), 0.0, 0.215, 0.29, 0.99);
VentanaL.Draw(lightingShader);
```

Se carga el modelo de la ventana lateral, sin transformaciones y con la transparencia activada.

```
//Parte lumínica de la lámpara (cristal)
model = glm::mat4(1);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 1);
glUniform4f(glGetUniformLocation(lightingShader.Program, "colorAlfa"), 1.0, 1.0, 1.0, 0.99);
Lampara.Draw(lightingShader);
```

Se carga el modelo de la parte de cristal de la lámpara, sin transformaciones y con la transparencia activada.

```
Anim.Use();
modelLoc = glGetUniformLocation(Anim.Program, "model");
viewLoc = glGetUniformLocation(Anim.Program, "view");
projLoc = glGetUniformLocation(Anim.Program, "projection");
glUniformMatrix4fv(viewLoc, 1, GL_FALSE, glm::value_ptr(view));
glUniformMatrix4fv(projLoc, 1, GL_FALSE, glm::value_ptr(projection));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
model = glm::mat4(1);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1f(glGetUniformLocation(Anim.Program, "time"), tiempo);
RadioPantalla.Draw(Anim);
```

Shader Anim:

```
#version 330 core
layout (location = 0) in vec3 aPos;
layout (location = 1) in vec3 aNormal;
layout (location = 2) in vec2 aTexCoords;

const float amplitude = 0.1;
const float frequency = 10.0;
const float PI = 3.14159;
out vec2 TexCoords;

uniform mat4 model;
uniform mat4 view;
uniform mat4 projection;
uniform float time;

void main()
{
 float distance = length(aPos);
 float effect = amplitude*sin(-PI*distance*time+frequency);
 gl_Position = projection*view*model*vec4(aPos.x,aPos.y, aPos.z,1);
 TexCoords=vec2(aTexCoords.x,aTexCoords.y+effect);
}
```

Se carga el modelo de la pantalla del radio, la cual funciona con el shader “Anim()”, el cual controla el efecto (con la variable effect) que es una senoidal para provocar el movimiento que se ve en la pantalla.

Código de la función “doMovement”.

Código – animación puerta.

```
//Abre y cierra la puerta
    if (rotPF >= 85.f) {
        dirPF = false;
    }
    else if (rotPF <= 0.f) {
        dirPF = true;
    }
```

En este segmento de código tenemos dos variables de tipo booleano: dirPF y activePF. La primera se encarga de asignar la dirección a la que rotara la puerta, mientras que la segunda activa la animación. Además, existe una variable de tipo flotante: rotPF, la cual aumentará o disminuirá dependiendo del movimiento de la puerta. En el primero IF se tiene una condicional, donde, si rotPF es igual o mayor a 85 (llega a su límite de rotación), la variable dirPF pasa a falso; es decir, cambia de dirección; caso contrario, si rotPF es igual o menor que 0 (llega a su otro límite de rotación), la variable dirPF cambia a “verdadero” (cambia de dirección).

```
    if (activePF) {
        if (dirPF)
                rotPF += 1.f;
    }
```

En el siguiente IF se pregunta por la variable “activePF”, la cual cambia según lo desee el usuario con la tecla “P”. Si la variable activePF se encuentra como “verdadero”, se pregunta por el valor de la variable dirPF, y, en caso de ser “verdadero” crecerá el valor de rotPF de forma cíclica para que rote la puerta.

```
    else {
        if(!dirPF)
        rotPF -= 1.f;
    }
```

Finalmente, si la variable activePF se encuentra en falso, se pregunta por la variable dirPF, y, de ser falso, el valor de la variable rotPF se decrementará de forma cíclica, provocando que la puerta se cierre.

Código – animaciones Radio

```
//Enciende // apaga la radio
    if (activeR) {
        tiempo = glfwGetTime();

        if (rotRI >= 180.f)
                dirRI = false;
        else if (rotRI <= 0.f)
                dirRI = true;
```

Lo primero que vemos es un IF con la variable “activeR” (boolean), la cual se cambia su valor con la tecla R por su negado. En el caso de que activeR se verdadero, se asigna a la variable “tiempo” (float) el reloj que lleva “glfwGetTime()”. Después, con otro IF se evalúa la dirección de la perilla izquierda. Si la variable “rotRI” (float) pasa o iguala 180 (los grados que gira la perilla), la variable “dirRI” (boolean) cambia su valor a falso; es decir, cambia de dirección. En el caso de que rotRI sea menor o igual a 0, dirRI cambia a verdadero (cambia de dirección).

```
        if (dirRI)
                rotRI += 0.5f;
        else
                rotRI -= 0.8f;
```

En el caso de que la variable sea dirRI sea verdadero, la variable rotRI aumentará su valor (tendrá dirección positiva) en 0.5 unidades de forma cíclica. Caso contrario, decrementará (dirección negativa) su valor en 0.8 unidades de forma cíclica.

```
        if (rotRD >= 85.f)
                dirRD = false;
        else if (rotRD <= 0.f)
                dirRD = true;
```

De igual forma, para la perilla derecha, si la variable "rotRD" (float) pasa o iguala las 85 unidades, la variable "dirRD" (boolean) cambia de dirección el giro de la rueda. Si es menor o igual a 0, dirRD pasa a verdadero.

```
        if (dirRD)
                rotRD += 1.5f;
        else
                rotRD -= 1.f;
    }
```

Cuando la variable dirRD es verdadero, la variable rotRD aumenta su valor (sentido positivo), caso contrario, decrementa su valor (sentido negativo).

```
    else {
        tiempo = 0.f;
    }
```

Finalmente, si la variable activeR tiene valor negativo, todas las variables relacionadas con las perillas no son alteradas y la variable tiempo es asignada a 0 (evitando que el shader fluctúe), simulado que el radio está apagado.

Código de la función "animacion_bajo()".

```
void animacion_bajo(){
    //Movimiento del bajo
    if (circuito){
```

Para que el recorrido pueda iniciar, es necesario que la variable "circuito" (boolean), controlada por el botón "B" sea Verdadero. En caso contrario, el recorrido no inicia o se pausa.

```
        if (recorrido1){
                rotKitZ += 0.08+(rotKitZ/10);
                if (rotKitZ > 13.6){
                        recorrido1 = false;
                        recorrido2 = true;
                }
        }
```

De serlo, se pregunta si la variable "recorrido1", (boolean) la cual fue declarada como verdadero, es verdadero. De serlo, el primer paso de la animación es simular el primer choque del bajo con el muro, el cual se produce a los 13.6°. Para rotar el bajo se utiliza la variable "rotKitZ", la cual crece de forma cada vez más rápida para simular la aceleración de la caída. Cuando se alcanza la posición, pasa al siguiente paso del recorrido, cambiando recorrido1 a falso y recorrido2 a verdadero.

```
if (recorrido2){
        rotKitY -= 2.88 - (rotKitY/20);
        if (rotKitY < -15.1){
                recorrido2 = false;
                recorrido3 = true;
        }
}
```

En el segundo paso del recorrido, resuelve la inclinación que tiene el bajo en Y, alineándose con muro. Una vez alcanza los 15.1° en el eje Y, a través del aumento de la variable rotKitY (float), avanza al siguiente paso, cambiando recorrido2 a falso y recorrido3 a verdadero.

```
if (recorrido3){
        if(movKitX < 0.911)
                movKitX += (0.05061111111112 -movKitX/20);
        rotKitZ += (0.9969444444444444445 - rotKitZ/50);
        if (rotKitZ > 32.5) {
                recorrido3 = false;
                recorrido4 = true;
        }
}
```

En el tercer paso se simula cómo cae el bajo a través del muro, por lo que se simula cómo decrementa la velocidad a través de la variable movKitX para su desplazamiento y rotKitZ para su rotación, o bien, el movimiento a través del muro. Una vez alcanzada la posición deseada, avanza al siguiente paso, cambiando recorrido3 a falso y recorrido4 a verdadero.

```
if (recorrido4) {
        movKitX += 0.035;
        rotKitZ -= (0.25 + rotKitZ/130);
        if (rotKitZ < 31.0) {
                recorrido4 = false;
                recorrido5 = true;
        }
}
```

En el cuarto paso, simula un rebote, por lo que la variable rotKitZ provoca que el bajo decrementa un grado y medio, y el desplazamiento en X, a través de la variable movkitX, incrementa mientras se llega al ángulo deseado. Una vez alcanzada la posición deseada, avanza al siguiente paso, cambiando recorrido4 a falso y recorrido5 a verdadero.

```
if (recorrido5) {
        movKitX += 0.0083;
        rotKitZ += (0.05 + (rotKitZ/120));
        if (rotKitZ > 38.8) {
                recorrido5 = false;
                recorrido6 = true;
        }
}
```

En el quito paso, el bajo vuelve a caer hasta los 38.8°, por lo que se aumenta nuevamente la variable rotKitZ de forma exponencial para simular la aceleración. Una vez alcanzada la posición deseada, avanza al siguiente paso, cambiando recorrido5 a falso y recorrido6 a verdadero.

```
        if (recorrido6) {
                rotKitZ += (0.05 + (rotKitZ / 120));
                if (rotKitZ > 40.) {
                        recorrido6 = false;
                        recorrido7 = true;
                }
        }
```

En el sexto paso, el bajo solo rota hasta chocar con el muro nuevamente, por lo que la variable rotKitZ aumenta de forma exponencial hasta llegar a los 40°. Una vez alcanzada la posición deseada, avanza al siguiente paso, cambiando recorrido6 a falso y recorrdio7 a verdadero.

```
        if (recorrido7) {
                if(movKitX <= 1.9)
                        movKitX += 0.005 + (movKitX/40);
                rotKitZ += (0.05 + (rotKitZ / 40));
                if (rotKitZ > 72.) {
                        recorrido7 = false;
                        recorrido8 = true;
                }
        }
```

En el séptimo paso, el bajo es desplazado en X hasta que llega a 1.9, mientas que rota hasta los 72° con un aumento que crece de forma exponencial. Una vez alcanzada la posición deseada, avanza al siguiente paso, cambiando recorrido7 a falso y recorrido8 a verdadero.

```
        if (recorrido8) {
                rotKitZ = 72.8;
                recorrido8 = false;
                recorrido9 = true;
                circuito = !circuito;
        }
```

En el octavo paso, se ajusta la rotación del bajo para que toque el suelo. Después avanza al siguiente paso, cambiando recorrido8 a falso y recorrido9 a verdadero. Además, se cambia la variable circuito a falso para que no se reinicie la animación.

```
        if (recorrido9) {
                rotKitY = -13.8;
                recorrido9 = false;
                recorrido10 = true;
        }
```

En el noveno paso, se ajusta la rotación del objeto en el eje Y con la variable rotKitY para que coincida con el suelo. Después avanza al siguiente paso, cambiando recorrido9 a falso y recorrido10 a verdadero.

```
        if (recorrido10 && circuito) {
                movKitX = 0.0;
                rotKitY = 0.0;
                rotKitZ = 0.0;
                recorrido10 = false;
                recorrido1 = true;
        }
    }
}
```

En el último paso reinicia todas las variables usadas. Después regresa al primero paso, cambiando recorrido10 a falso y recorrida1 a verdadero.

Código de la función “keycallback”.

Código – animación lámpara:

```
if (keys[GLFW_KEY_L]){
		activeL = !activeL;
		if (activeL){
			lampOn.x = 1.0f;
			lampOn.y = 1.0f;
			lampOn.z = 0.717647f;
			Light1 = glm::vec3(1.0f, 1.0f, .0f);
		}
		else{
			lampOn = glm::vec3(0.0f);
			Light1 = glm::vec3(0);//Cuado es solo un valor en los 3 vectores pueden dejar solo una componente
		}
	}
```

Esta condicional está destinada a la animación de la lámpara, la cual puede activarse con la tecla “L”. Cuando se presiona la tecla y la condición es verdadera, cambia la variable “activeL” (boolean) a su negado; es decir, si esta se encuentra como “verdadero”, cambiaría a “falso”. Cuando activeL es verdadero, se asigna al vector (vec3) lampOn con los valores correspondientes para el color de la iluminación, y al vector (vec3) Light1 el color amarillo. En caso contrario, los vectores (vec3) lampOn y Light1 se asignan en 0, por lo que se mostraría como apagada.

Código – animación Puerta.

```
	if (keys[GLFW_KEY_P]){
		activePF = !activePF;
	}
```

En esta condición se cambia la variable “activePF” (boolean), encargada de controlar cuándo se activa la animación de la puerta, por su negado. Es decir, si se encuentra como “verdadero” pasa a “falso” al presionar la tecla “P”.

Código – animación Radio.

```
	if (keys[GLFW_KEY_R]) {
		activeR = !activeR;
	}
```

En esta condición se cambia la variable “activeR” (boolean), encargada de controlar cuándo se activan las animaciones de la Radio, por su negado. Es decir, si se encuentra como “verdadero” pasa a “falso” al presionar la tecla “R”.

Código – animación Bajo.

```
	if (keys[GLFW_KEY_B]) {
		circuito = !circuito;
	}
}
```

En esta condición se cambia la variable “activeB” (boolean), encargada de controlar cuando se activa la animación del bajo, por su negado. Es decir, si se encuentra como “verdadero” pasa a “falso” al presionar la tecla “B”.

Conclusiones.

Durante el desarrollo del proyecto, incluso en el modelado de los objetos, se utilizaron los conocimientos adquiridos a lo largo del curso. Esto puede notarse en el avance de los objetos, y, en especial en el tiempo en que se realizaron; pues algunas figuras al principio tomaron mucho tiempo pese a ser sencillas, mientras que otras se realizaron en tiempos muy reducidos. De igual forma, el manejo de la iluminación y la creación de animaciones denotan un claro manejo de los conocimientos, por lo que los objetivos se han alcanzado de manera satisfactoria, y se quedan a la espera de nuevos en materias posteriores.

Facultad de Ingeniería

Final Project of Computación Gráfica e Interacción Humano Computadora

Marceline's house, from the animated series Adventure Time

Technical manual

Account number: 316170040. Group: 12

Index.

Objective.

The student must apply and demonstrate the knowledge acquired throughout the course.

Project scope.

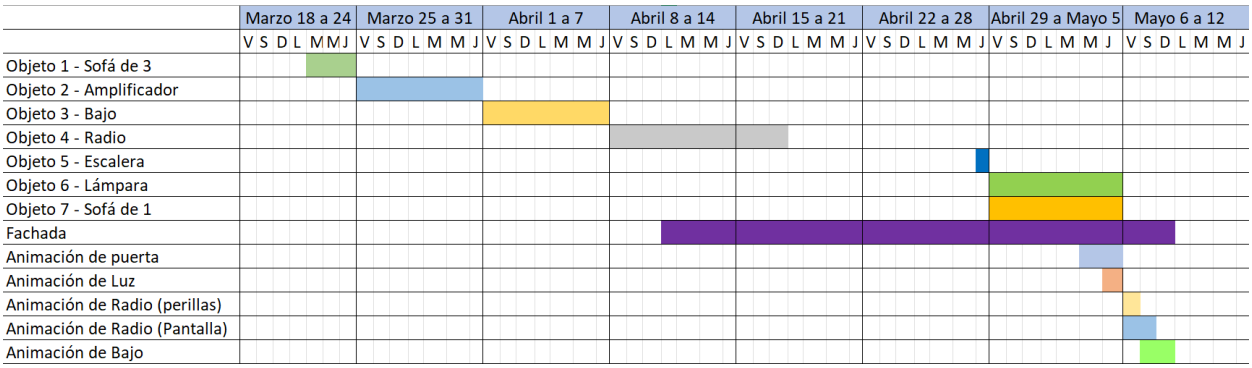
Make a 3D recreation of the chosen house with 7 objects of the house and transport them to OpenGL to make 5 animations (3 simple and 2 complex), applying the knowledge in the course in approximately 3 months.

Limitations.

There will be no collisions.

Don’t use modeling and animation methods than doesn’t seen in the course.

Gantt diagram.



Source code documentation.

To the easy understanding of the source code, have a section where the complete source code appears (with the code comments) and a part dedicated to documenting the parts that were added to the base code.

Source code:

```
#include <iostream>

#include <cmath>


// GLEW

#include <GL/glew.h>


// GLFW

#include <GLFW/glfw3.h>


// Other Libs

#include "stb_image.h"
```

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

// function declaration
void KeyCallback(GLFWwindow* window, int key, int scancode, int action, int mode);
void MouseCallback(GLFWwindow* window, double xPos, double yPos);
void DoMovement();
void animacion_bajo();

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

// Camera
Camera  camera(glm::vec3(5.0f, 5.0f, 20.0f));
GLfloat lastX = WIDTH / 2.0;
GLfloat lastY = HEIGHT / 2.0;
bool keys[1024];
bool firstMouse = true;

// Light attributes
glm::vec3 lightPos(.0f, .0f, 0.0f);

// Animation variables
```

```
//Lamp//
bool activeL = false; //Boolean that works as a switch for the animation of the lamp.
glm::vec3 lampOn;              //It is a vec3 with the colors produced by the light. It is the consequence of the previous bool .
glm::vec3 pointLightPositions(0.0f, 8.2f, 0.0f); // Position of the pointlight (in the lamp)
glm::vec3 Light1 = glm::vec3(0); //Light of the pointlight

//Front door
bool activePF = false; //Boolean that works as a switch for the door animation.
float rotPF = 0.f;     //Variable used in the rotate of the door to produce the animation.
bool dirPF = false;    //Boolean that decides the direction in which the gate will move.

//Radio
bool activeR = false;          //Boolean that works as a switch for the animation of the radio.
bool dirRI = false;            //Boolean that decides the direction in which the left radio knob will move.
bool dirRD = false;            //Boolean that decides the direction in which the right radio knob will move.
float rotRI = 0.f;             //Rotation angle of the left radius knob.
float rotRD = 0.f;             //Rotation angle of the right radius knob.
float tiempo;                  //Independent variable in the sine function of the anim.vs file

//Bass
glm::vec3 PosIni(-3.95f, .55f, 0.0f); //Posición inicial del objeto para iniciar la animación. Punto de origen para el objeto
float movKitX = 0.0;           //Variable that will be added to its position on the X axis.
float movKitY = 0.0;           //Variable that will be added to its position on the Y axis.
float movKitZ = 0.0;           //Variable that will be added to its position on the Z axis.
float rotKitX = 0.0;           //Variable that will rotate the object on the X axis.
float rotKitY = 0.0;//Variable that will rotate the object on the Y axis.
float rotKitZ = 0.0;           //Variable that will rotate the object on the Z axis.

bool circuito = false;         //Booleano que funciona como switch para la animación del bajo. Permite que el recorrido se lleve a
cabo.
bool recorrido1 = true;              //Step 1 in the circuit.
bool recorrido2 = false;       //Step 2 in the circuit.
bool recorrido3 = false;       //Step 3 in the circuit.
bool recorrido4 = false;       //Step 4 in the circuit.
bool recorrido5 = false;       //Step 5 in the circuit.
bool recorrido6 = false;       //Step 6 in the circuit.
```

```
bool recorrido7 = false;       //Step 7 in the circuit.
bool recorrido8 = false;       //Step 8 in the circuit.
bool recorrido9 = false;       //Step 9 in the circuit.
bool recorrido10 = false;      //Step 10 in the circuit.
////////////////////////////////////////////////////////////////////////////////////

float vertices[] = {
            -0.5f, -0.5f, -0.5f,  0.0f,  0.0f, -1.0f,
                    0.5f, -0.5f, -0.5f,  0.0f,  0.0f, -1.0f,
                    0.5f,  0.5f, -0.5f,  0.0f,  0.0f, -1.0f,
                    0.5f,  0.5f, -0.5f,  0.0f,  0.0f, -1.0f,
             -0.5f,  0.5f, -0.5f,  0.0f,  0.0f, -1.0f,
             -0.5f, -0.5f, -0.5f,  0.0f,  0.0f, -1.0f,

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

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

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

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

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




// Deltatime
GLfloat deltaTime = 0.0f;    // Time between the last frame and the current one
GLfloat lastFrame = 0.0f;    // Last frame time



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

        //Create a GLFWwindow object that can be used for GLFW functions
        GLFWwindow* window = glfwCreateWindow(WIDTH, HEIGHT, "Proyecto Final", 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 necessary callback functions
glfwSetKeyCallback(window, KeyCallback);
glfwSetCursorPosCallback(window, MouseCallback);

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

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

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



//Declaración de funciones de tipo Shader
Shader lightingShader("Shaders/lighting.vs", "Shaders/lighting.frag");
Shader lampShader("Shaders/lamp.vs", "Shaders/lamp.frag");
Shader Anim("Shaders/anim.vs", "Shaders/anim.frag");

//Definition of the models

//Facade
```

```
Model Casa((char*)"Models/casaMarceline/casa.obj");
Model VentanaPF((char*)"Models/casaMarceline/vpf.obj");
Model VentanaPL((char*)"Models/casaMarceline/vpl.obj");
Model PuertaF((char*)"Models/casaMarceline/puertaFrente.obj");
Model PuertaL((char*)"Models/casaMarceline/puertaLado.obj");
Model PuertaP((char*)"Models/casaMarceline/puertaPerro.obj");
Model VentanaF((char*)"Models/casaMarceline/ventanaFrente.obj");
Model VentanaL((char*)"Models/casaMarceline/ventanaLado.obj");

//Objects
// Model type declarations to be able to bring the OBJ to OpenGL
Model Bajo((char*)"Models/bajoMarceline/bajoMarceline.obj");
Model Amp((char*)"Models/ampMarceline/ampMarceline.obj");
Model Escalera((char*)"Models/escaleraMarceline/escaleraMarceline.obj");
Model LamparaMetal((char*)"Models/lamparaMarceline/lamparaMarcelineMetal.obj");
Model Lampara((char*)"Models/lamparaMarceline/lamparaMarceline.obj");
Model Radio((char*)"Models/radioMarceline/radioMarceline.obj");
Model RadioPantalla((char*)"Models/radioMarceline/pantalla.obj");
Model RadioPI((char*)"Models/radioMarceline/perillaI.obj");
Model RadioPD((char*)"Models/radioMarceline/perillaD.obj");
Model Sofa3((char*)"Models/sofaMarceline/sofaMarceline.obj");
Model Sofa1((char*)"Models/sofaMarceline1/sofaMarceline1.obj");



// VAO and VBO configuration
GLuint VBO, VAO;
glGenVertexArrays(1, &VAO);
glGenBuffers(1, &VBO);
glBindVertexArray(VAO);
glBindBuffer(GL_ARRAY_BUFFER, VBO);
glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW);

//Position attributes
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(GLfloat), (GLvoid*)0);
glEnableVertexAttribArray(0);
```

```
        //Atributos de la normal
        glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(float), (void*)(3 * sizeof(float)));
        glEnableVertexAttribArray(1);

        //Configuration of texture units
        lightingShader.Use();
        glUniform1i(glGetUniformLocation(lightingShader.Program, "material.diffuse"), 0);
        glUniform1i(glGetUniformLocation(lightingShader.Program, "material.specular"), 1);

        // Perspective projection
        glm::mat4 projection = glm::perspective(camera.GetZoom(), (GLfloat)SCREEN_WIDTH / (GLfloat)SCREEN_HEIGHT, 0.1f,
100.0f);

        //Cyclic configuration where light and model attributes are set. It stops when the user decides.
        while (!glfwWindowShouldClose(window)) {

                // DeltaTime of the current frame
                GLfloat currentFrame = glfwGetTime();
                deltaTime = currentFrame - lastFrame;
                lastFrame = currentFrame;

        // Check if any event has occurred (such as key press or mouse movement)
                //Keep animations in loop ( animacion_bajo ( ) )
                glfwPollEvents();
                DoMovement();
                animacion_bajo();

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

                //Enable openGL options
                glEnable(GL_DEPTH_TEST);

                        //Use the corresponding 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);

//Lightning
//Directional light
glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.direction"), -0.2f, -1.0f, -0.3f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.ambient"), 0.4f, 0.4f, 0.4f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.diffuse"), 0.1f, 0.1f, 0.1f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.specular"), 1.f, 1.f, 1.f);

//Point light
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].position"), pointLightPositions.x,
pointLightPositions.y, pointLightPositions.z);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].ambient"), lampOn.x, lampOn.y,
lampOn.z);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].diffuse"), lampOn.x, lampOn.y,
lampOn.z);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].specular"), 1.0f, 1.0f, 0.0f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[0].constant"), 1.0f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[0].linear"), 0.35f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[0].quadratic"), .44f);

// 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"), 1.0f, 1.0f, 1.0f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight.diffuse"), 1.0f, 1.0f, 1.0f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight.specular"), 1.0f, 1.0f, 1.0f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight.constant"), 1.0f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight.linear"), 0.35f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight.quadratic"), .44f);
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 the properties of the material
```

```
glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 16.0f);

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

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

//Pass from arrays to shaders
glUniformMatrix4fv(viewLoc, 1, GL_FALSE, glm::value_ptr(view));
glUniformMatrix4fv(projLoc, 1, GL_FALSE, glm::value_ptr(projection));

//Load model
glm::mat4 model(1);
view = camera.GetViewMatrix();

//Fachada
model = glm::mat4(1);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
Casa.Draw(lightingShader);

//Puerta del frente
model = glm::mat4(1);
model = translate(model, glm::vec3(1.05f, 3.5f, 9.f));
model = glm::rotate(model, glm::radians(rotPF), glm::vec3(.0f, 1.f, 0.f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
PuertaF.Draw(lightingShader);

//Puerta lateral
model = glm::mat4(1);
```

```
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
PuertaL.Draw(lightingShader);

//Puerta para el perro
model = glm::mat4(1);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
PuertaP.Draw(lightingShader);

//Objetos
                        //Amplificador
model = glm::mat4(1);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
Amp.Draw(lightingShader);

//Bajo
model = glm::mat4(1);
model = glm::translate(model, PosIni + glm::vec3(movKitX, movKitY, movKitZ));
model = glm::rotate(model, glm::radians(rotKitZ), glm::vec3(0.0f, 0.0f, 1.0));
model = glm::rotate(model, glm::radians(rotKitX), glm::vec3(1.0f, 0.0f, 0.0));
model = glm::rotate(model, glm::radians(rotKitY), glm::vec3(0.0f, 1.0f, 0.0));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
Bajo.Draw(lightingShader);

//Escalera
model = glm::mat4(1);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
Escalera.Draw(lightingShader);

//Parte metálica (no transparente) de la lámpara
model = glm::mat4(1);
```

```
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
LamparaMetal.Draw(lightingShader);

//RADIO
model = glm::mat4(1);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
Radio.Draw(lightingShader);

//Perilla izquierda
model = glm::mat4(1);
model = translate(model, glm::vec3(-4.42f, 1.1f, 5.6f));
model = glm::rotate(model, glm::radians(44.f), glm::vec3(0.0f, 1.f, 0.f));
model = glm::rotate(model, glm::radians(rotRI), glm::vec3(1.0f, 0.f, 0.f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
RadioPI.Draw(lightingShader);

//Perilla derecha
model = glm::mat4(1);
model = translate(model, glm::vec3(-4.9f, 1.1f, 5.1f));
model = glm::rotate(model, glm::radians(44.f), glm::vec3(0.0f, 1.f, 0.f));
model = glm::rotate(model, glm::radians(rotRD), glm::vec3(1.0f, 0.f, 0.f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
RadioPD.Draw(lightingShader);

//Sofá para 1
model = glm::mat4(1);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
Sofa1.Draw(lightingShader);

//Sofá para 3
```

```
model = glm::mat4(1);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
Sofa3.Draw(lightingShader);

//TRANSPARENT OBJECTS

glEnable(GL_BLEND);//Avtiva la funcionalidad para trabajar el canal alfa
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);

//Ventanas de la puerta del frente
model = glm::mat4(1);
model = translate(model, glm::vec3(1.05f, 3.5f, 9.f));
model = glm::rotate(model, glm::radians(rotPF), glm::vec3(.0f, 1.f, 0.f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 1);
glUniform4f(glGetUniformLocation(lightingShader.Program, "colorAlfa"), 0.0, 0.215, 0.29, 0.99);
VentanaPF.Draw(lightingShader);

//Ventanas de la puerta lateral
model = glm::mat4(1);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 1);
glUniform4f(glGetUniformLocation(lightingShader.Program, "colorAlfa"), 0.0, 0.215, 0.29, 0.99);
VentanaPL.Draw(lightingShader);

//Ventana frontal
model = glm::mat4(1);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 1);
glUniform4f(glGetUniformLocation(lightingShader.Program, "colorAlfa"), 0.0, 0.215, 0.29, 0.99);
VentanaF.Draw(lightingShader);

//Ventana lateral
model = glm::mat4(1);
```

```
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 1);
glUniform4f(glGetUniformLocation(lightingShader.Program, "colorAlfa"), 0.0, 0.215, 0.29, 0.99);
VentanaL.Draw(lightingShader);

//Parte lumínica de la lámpara (cristal)
model = glm::mat4(1);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 1);
glUniform4f(glGetUniformLocation(lightingShader.Program, "colorAlfa"), 1.0, 1.0, 1.0, 0.99);
Lampara.Draw(lightingShader);

glDisable(GL_BLEND);  //Desactiva el canal alfa
glUniform4f(glGetUniformLocation(lightingShader.Program, "colorAlfa"), 1.0, 1.0, 1.0, 1.0); //Reset para el
resto de cosas

//Pantalla de la radio
Anim.Use();
modelLoc = glGetUniformLocation(Anim.Program, "model");
viewLoc = glGetUniformLocation(Anim.Program, "view");
projLoc = glGetUniformLocation(Anim.Program, "projection");
glUniformMatrix4fv(viewLoc, 1, GL_FALSE, glm::value_ptr(view));
glUniformMatrix4fv(projLoc, 1, GL_FALSE, glm::value_ptr(projection));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
model = glm::mat4(1);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1f(glGetUniformLocation(Anim.Program, "time"), tiempo);
RadioPantalla.Draw(Anim);

glBindVertexArray(0);

//End transparency

                        // Redraw the lamp object for the appropriate shader
lampShader.Use();
    //Get location objects for the arrays in the lampshader
```

```
		modelLoc = glGetUniformLocation(lampShader.Program, "model");
		viewLoc = glGetUniformLocation(lampShader.Program, "view");
		projLoc = glGetUniformLocation(lampShader.Program, "projection");

		//Set up the arrays
		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
		model = glm::mat4(1);
		model = glm::translate(model, pointLightPositions);
		model = glm::scale(model, glm::vec3(0.2f)); // Make it a smaller cube
		glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
		glBindVertexArray(VAO);
		glDrawArrays(GL_TRIANGLES, 0, 36);
		glBindVertexArray(0);

			//Swap screen buffers
		glfwSwapBuffers(window);
	}

	//Ends GLFW. Clean all resources used by GLFW
	glfwTerminate();

	return 0;
}

// Moves/alters the camera positions based on user input
void DoMovement() {
	//Controles de la cámara
	if (keys[GLFW_KEY_W]) {
		camera.ProcessKeyboard(FORWARD, deltaTime * 2);
```

```
}

if (keys[GLFW_KEY_S] || keys[GLFW_KEY_DOWN]) {
	camera.ProcessKeyboard(BACKWARD, deltaTime * 2);
}

if (keys[GLFW_KEY_A] || keys[GLFW_KEY_LEFT]) {
	camera.ProcessKeyboard(LEFT, deltaTime * 2);
}

if (keys[GLFW_KEY_D] || keys[GLFW_KEY_RIGHT]) {
	camera.ProcessKeyboard(RIGHT, deltaTime * 2);
}

//Abre y cierra la puerta
if (rotPF >= 85.f) {
	dirPF = false;
}
else if (rotPF <= 0.f) {
	dirPF = true;
}

if (activePF) {
	if (dirPF)
		rotPF += 1.f;
}
else {
	if (!dirPF)
		rotPF -= 1.f;
}

//Enciende // apaga la radio
if (activeR) {
	tiempo = glfwGetTime();
```

```
		if (rotRI >= 180.f)
			dirRI = false;
		else if (rotRI <= 0.f)
			dirRI = true;

		if (dirRI)
			rotRI += 0.5f;
		else
			rotRI -= 0.8f;

		if (rotRD >= 85.f)
			dirRD = false;
		else if (rotRD <= 0.f)
			dirRD = true;

		if (dirRD)
			rotRD += 1.5f;
		else
			rotRD -= 1.f;
	}
	else {
		tiempo = 0.f;
	}
}

void animacion_bajo() {
	//Movimiento del bajo
	if (circuito) {
		if (recorrido1) {
			rotKitZ += 0.08 + (rotKitZ / 10);
			if (rotKitZ > 13.6) {
				recorrido1 = false;
				recorrido2 = true;
			}
		}
```

```
if (recorrido2) {
		rotKitY -= 2.88 - (rotKitY / 20);
		if (rotKitY < -15.1) {
			recorrido2 = false;
			recorrido3 = true;
		}
}

if (recorrido3) {
		if (movKitX < 0.911)
			movKitX += (0.05061111111112 - movKitX / 20);
		rotKitZ += (0.996944444444444445 - rotKitZ / 50);
		if (rotKitZ > 32.5) {
			recorrido3 = false;
			recorrido4 = true;
		}
}

if (recorrido4) {
		movKitX += 0.035;
		rotKitZ -= (0.25 + rotKitZ / 130);
		if (rotKitZ < 31.0) {
			recorrido4 = false;
			recorrido5 = true;
		}
}

if (recorrido5) {
		movKitX += 0.0083;
		rotKitZ += (0.05 + (rotKitZ / 120));
		if (rotKitZ > 38.8) {
			recorrido5 = false;
			recorrido6 = true;
		}
}
```

```
if (recorrido6) {
		rotKitZ += (0.05 + (rotKitZ / 120));
		if (rotKitZ > 40.) {
			recorrido6 = false;
			recorrido7 = true;
		}
}

if (recorrido7) {
		if (movKitX <= 1.9)
			movKitX += 0.005 + (movKitX / 40);
		rotKitZ += (0.05 + (rotKitZ / 40));
		if (rotKitZ > 72.) {
			recorrido7 = false;
			recorrido8 = true;
		}
}

if (recorrido8) {
		rotKitZ = 72.8;
		recorrido8 = false;
		recorrido9 = true;
		circuito = !circuito;
}

if (recorrido9) {
		rotKitY = -13.8;
		recorrido9 = false;
		recorrido10 = true;
}

if (recorrido10 && circuito) {
		movKitX = 0.0;
		rotKitY = 0.0;
```

```
                rotKitZ = 0.0;
                recorrido10 = false;
                recorrido1 = true;
            }
        }
}

//Se llama cuando alguna tecla es presionada vía 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_L]) {
            activeL = !activeL;
            if (activeL) {
                lampOn.x = 1.0f;
                lampOn.y = 1.0f;
                lampOn.z = 0.717647f;
                Light1 = glm::vec3(1.0f, 1.0f, .0f);
            }
            else {
                lampOn = glm::vec3(0.0f);
                Light1 = glm::vec3(0);//Cuado es solo un valor en los 3 vectores pueden dejar solo una componente
            }
        }
```

```
	if (keys[GLFW_KEY_P]) {
		activePF = !activePF;
	}

	if (keys[GLFW_KEY_R]) {
		activeR = !activeR;
	}

	if (keys[GLFW_KEY_B]) {
		circuito = !circuito;
	}
}

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);
}
```

Code added to base code:

The following lines represent the code added to the base code for the correct visualization of the 3D recreation of the selected house. This code is commented in detail to understand the correct operation of the animations and the loading of the models.

Model loading code (main () function).

```
//Facade
	model = glm :: mat4(1);
	glUniformMatrix4 fv( modelLoc , 1, GL_FALSE, glm :: value_ptr ( model ));
	glUniform1 i( glGetUniformLocation ( lightingShader.Program , " activateTransparency "), 0);
	House.Draw ( lightingShader );
```

In this model, compose the facade (walls, door and window frames, static objects that are part of the facade) the model is loaded without transformations and transparency disable.

```
	//Front door
	model = glm :: mat4(1);
	model = translate ( model,glm :: vec3(1.05f,3.5f,9.f));
	model = glm :: rotate ( model , glm :: radians ( rotPF ), glm ::vec3(.0f, 1.f, 0.f));
	glUniformMatrix4 fv( modelLoc , 1, GL_FALSE, glm :: value_ptr ( model ));
	glUniform1 i( glGetUniformLocation ( lightingShader.Program , " activateTransparency "), 0);
	GateF.Draw ( lightingShader );
```

The model of the front door is loaded, without windows, with a translation to its position and with the rotation activated, which will be controlled by the variable " rotPF ".

```
	//Side door
	model = glm :: mat4(1);
	glUniformMatrix4 fv( modelLoc , 1, GL_FALSE, glm :: value_ptr ( model ));
	glUniform1 i( glGetUniformLocation ( lightingShader.Program , " activateTransparency "), 0);
	DoorL.Draw ( lightingShader );
```

The side door model is loaded, without its windows, without transformations, and the transparency is turned off.

```
	//door for the dog
	model = glm :: mat4(1);
	glUniformMatrix4 fv( modelLoc , 1, GL_FALSE, glm :: value_ptr ( model ));
	glUniform1 i( glGetUniformLocation ( lightingShader.Program , " activateTransparency "), 0);
	DoorP.Draw ( lightingShader );
```

The model of the door for the dog (which is on the side door) is loaded, without transformations and the transparency is turned off.

```
//Objects
	//Amplifier
	model = glm :: mat4(1);
	glUniformMatrix4 fv( modelLoc , 1, GL_FALSE, glm :: value_ptr ( model ));
	glUniform1 i( glGetUniformLocation ( lightingShader.Program , " activateTransparency "), 0);
	Amp.Draw ( lightingShader );
```

The amp model is loaded without transformations and transparency.

```
//Bass
model = glm :: mat4(1);
model = glm :: translate ( model , PosIni + glm ::vec3( movKitX , movKitY , movKitZ ));
model = glm :: rotate ( model , glm :: radians ( rotKitZ ), glm ::vec3(0.0f, 0.0f, 1.0));
model = glm :: rotate ( model , glm :: radians ( rotKitX ), glm ::vec3(1.0f, 0.0f, 0.0));
model = glm :: rotate ( model , glm :: radians ( rotKitY ), glm ::vec3(0.0f, 1.0f, 0.0));
glUniformMatrix4 fv( modelLoc , 1, GL_FALSE, glm :: value_ptr ( model ));
glUniform1 i( glGetUniformLocation ( lightingShader.Program , " activateTransparency "), 0);
Low.Draw ( lightingShader );
```

The bass model is loaded, without transparency. The bass is moved to its original position with the variable PosIni (vec3), and a vec3 is added that alters the position in all directions. The variables "movKitX" (float), "movKitY" (float) and "movKitZ" (float) are altered in the function "animacion_bajo ( )" to change the position of the bass. Also, it has 3 rotation transformations (each in different directions), where the variable "rotKitZ" (float) affects the rotation of the bass on the Z axis, the variable "rotKitX" (float) affects the rotation of the bass on the Z axis X and the variable "rotKitY" (float) affect the rotation of the bass on the Y axis, all modified on the "animacion_bajo( )" function, which is detailed in the section <<Code of the "animacion_bajo( )" function> >.

```
//Ladder
model = glm :: mat4(1);
glUniformMatrix4 fv( modelLoc , 1, GL_FALSE, glm :: value_ptr ( model ));
glUniform1 i( glGetUniformLocation ( lightingShader.Program , " activateTransparency "), 0);
Ladder.Draw ( lightingShader );
```

The stair model is loaded without transformations and transparency.

```
//Metal part (not transparent) of the lamp
model = glm :: mat4(1);
glUniformMatrix4 fv( modelLoc , 1, GL_FALSE, glm :: value_ptr ( model ));
glUniform1 i( glGetUniformLocation ( lightingShader.Program , " activateTransparency "), 0);
LampMetal.Draw ( lightingShader );
```

The lamp base model is loaded without transformations and transparency.

```
//RADIO
model = glm :: mat4(1);
glUniformMatrix4 fv( modelLoc , 1, GL_FALSE, glm :: value_ptr ( model ));
glUniform1 i( glGetUniformLocation ( lightingShader.Program , " activateTransparency "), 0);
Radio.Draw ( lightingShader );
```

The base model of the radio is loaded, without the knobs and display, and without transformations and transparency.

```
//Left Knob
model = glm :: mat4(1);
model = translate ( model , glm ::vec3(-4.42f, 1.1f, 5.6f));
model = glm :: rotate ( model , glm :: radians (44.f), glm ::vec3(0.0f, 1.f, 0.f));
model = glm :: rotate ( model , glm :: radians ( rotRI ), glm ::vec3(1.0f, 0.f, 0.f));
glUniformMatrix4 fv( modelLoc , 1, GL_FALSE, glm :: value_ptr ( model ));
glUniform1 i( glGetUniformLocation ( lightingShader.Program , " activateTransparency "), 0);
RadioPI.Draw ( lightingShader );
```

The model of the left knob is loaded without transparency, The knob is moved to its position and with two rotations: the first giving it the same inclination that the radio, the second is for its rotation animation, which is controlled by the variable rotRI. This movement is explained in the "Code – Radio animations" section.

```
//Right knob
model = glm :: mat4(1);
model = translate ( model , glm ::vec3(-4.9f, 1.1f, 5.1f));
model = glm :: rotate ( model , glm :: radians (44.f), glm ::vec3(0.0f, 1.f, 0.f));
model = glm :: rotate ( model , glm :: radians ( rotRD ), glm ::vec3(1.0f, 0.f, 0.f));
glUniformMatrix4 fv( modelLoc , 1, GL_FALSE, glm :: value_ptr ( model ));
glUniform1 i( glGetUniformLocation ( lightingShader.Program , " activateTransparency "), 0);
RadioPD.Draw ( lightingShader );
```

The model of the right knob is loaded without transparency, The knob is moved to its position and with two rotations: the first giving it the same inclination that the radio, the second is for its rotation animation, which is controlled by the variable rotRD . This movement is explained in the “Code – Radio animations” section.

```
//Sofa for 1
model = glm :: mat4(1);
glUniformMatrix4 fv( modelLoc , 1, GL_FALSE, glm :: value_ptr ( model ));
glUniform1 i( glGetUniformLocation ( lightingShader.Program , " activateTransparency "), 0);
Sofa1.Draw( lightingShader );
```

The model of the sofa for 1 is loaded, without transformations and transparency.

```
//Sofa for 3
model = glm :: mat4(1);
glUniformMatrix4 fv( modelLoc , 1, GL_FALSE, glm :: value_ptr ( model ));
glUniform1 i( glGetUniformLocation ( lightingShader.Program , " activateTransparency "), 0);
Sofa3.Draw( lightingShader );
```

The model of the sofa for 3 is loaded, without transformations and transparency.

```
//TRANSPARENT OBJECTS

//front door windows
model = glm :: mat4(1);
model = translate ( model , glm ::vec3(1.05f, 3.5f, 9.f));
model = glm :: rotate ( model , glm :: radians ( rotPF ), glm ::vec3(.0f, 1.f, 0.f));
glUniformMatrix4 fv( modelLoc , 1, GL_FALSE, glm :: value_ptr ( model ));
glUniform1 i( glGetUniformLocation ( lightingShader.Program , " activateTransparency "), 1);
glUniform4 f( glGetUniformLocation ( lightingShader.Program , " colorAlpha "), 0.0, 0.215, 0.29, 0.99);
WindowPF.Draw ( lightingShader );
```

The model of the front door windows is loaded, with a translation to its location and a rotation that follows the movement of the door, with transparency turned on.

```
//Side door windows
model = glm :: mat4(1);
glUniformMatrix4 fv( modelLoc , 1, GL_FALSE, glm :: value_ptr ( model ));
glUniform1 i( glGetUniformLocation ( lightingShader.Program , " activateTransparency "), 1);
glUniform4 f( glGetUniformLocation ( lightingShader.Program , " colorAlpha "), 0.0, 0.215, 0.29, 0.99);
WindowPL.Draw ( lightingShader );
```

The model of the side door windows is loaded, with a translation to its location, with transparency turned on.

```
//front window
model = glm :: mat4(1);
glUniformMatrix4 fv( modelLoc , 1, GL_FALSE, glm :: value_ptr ( model ));
glUniform1 i( glGetUniformLocation ( lightingShader.Program , " activateTransparency "), 1);
glUniform4 f( glGetUniformLocation ( lightingShader.Program , " colorAlpha "), 0.0, 0.215, 0.29, 0.99);
WindowF.Draw ( lightingShader );
```

The front window model is loaded, without transformations and with transparency enabled.

```
//side window
model = glm :: mat4(1);
glUniformMatrix4 fv( modelLoc , 1, GL_FALSE, glm :: value_ptr ( model ));
glUniform1 i( glGetUniformLocation ( lightingShader.Program , " activateTransparency "), 1);
glUniform4 f( glGetUniformLocation ( lightingShader.Program , " colorAlpha "), 0.0, 0.215, 0.29, 0.99);
WindowL.Draw ( lightingShader );
```

The side window model loads, without transformations and with transparency turned on.

```
// Light part of the lamp (glass)
model = glm :: mat4(1);
glUniformMatrix4 fv( modelLoc , 1, GL_FALSE, glm :: value_ptr ( model ));
glUniform1 i( glGetUniformLocation ( lightingShader.Program , " activateTransparency "), 1);
glUniform4 f( glGetUniformLocation ( lightingShader.Program , " colorAlpha "), 1.0, 1.0, 1.0, 0.99);
Lamp.Draw ( lightingShader );
```

The model of the glass part of the lamp is loaded, without transformations and with transparency activated.

```
Anima.Use ();
modelLoc = glGetUniformLocation ( Anim.Program , " model ");
viewLoc = glGetUniformLocation ( Anim.Program , " view ");
projLoc = glGetUniformLocation ( Anim.Program , " projection ");
glUniformMatrix4 fv( viewLoc , 1, GL_FALSE, glm :: value_ptr ( view ));
glUniformMatrix4 fv( projLoc , 1, GL_FALSE, glm :: value_ptr ( projection ));
glUniformMatrix4 fv( modelLoc , 1, GL_FALSE, glm :: value_ptr ( model ));
model = glm :: mat4(1);
glUniformMatrix4 fv( modelLoc , 1, GL_FALSE, glm :: value_ptr ( model ));
glUniform1 f( glGetUniformLocation ( Anim.Program , "time"), time);
RadioScreen.Draw ( Anim );
```

shader Anime :

```
#version 330 core
layout ( location = 0) in vec3 aPos ;
layout ( location = 1) in vec3 aNormal ;
layout ( location = 2) in vec2 aTexCoords ;

build float amplitude = 0.1;
build float frequency = 10.0;
build float PI = 3.14159;
out vec2 TexCoords ;

uniform mat4 model ;
uniform mat4 view ;
uniform mat4 projection ;
uniform floattime ;

void main ()
{
 float distance = length ( aPos );
 float effect = amplitude *sin(-PI* distance * time+frequency );
 gl_Position = projection * view * model *vec4( aPos.x,aPos.y , aPos.z,1);
 TexCoords =vec2( aTexCoords.x,aTexCoords.y +effect );
}
```

The radio screen model is loaded, it works with the “Anim( )” shader, which controls the sinusoidal effect (with the effect variable) to cause the movement that is seen on the screen.

Code of the “ doMovement ” function.

Code – door animation.

```
//Open and close the door
    if ( rotPF >= 85.f) {
        dirPF = false;
    }
    else if ( rotPF <= 0.f) {
        addrPF = true;
    }
```

In this code segment we have two boolean type variables: dirPF and activePF . The first is responsible for assigning the direction to the door will rotate, while the second activates the animation. Also, there is a variable of type float: rotPF, which will increase or decrease depending on the movement of the door. In the first IF there is a conditional, where, if rotPF is equals or greater than 85 (it reaches its rotation limit), the variable dirPF becomes false; it changes direction. Else, if rotPF is equals or less than 0 (it reaches its other rotation limit), the variable dirPF changes to "true" (changes direction).

```
    if ( activePF ) {
        if ( PFdir )
                rotPF += 1.f;
    }
```

In the following IF asked for the variable “activePF”, which changes with the “P” key. If the variable activePF its "true", it asks for the value of the variable dirPF, and, if it is "true", the value of rotPF will grow to do the door rotates.

```
    else {
        if (! dirPF )
        rotPF -= 1.f;
    }
```

Finally, if the variable activePF is false, it asks for the variable dirPF, and, if false, the value of the variable rotPF will be decremented cyclically, causing the gate to close.

Code – Radio animations

```
//turn on //turn off the radio
    if ( activeR ) {
        time = glfwGetTime ( );

        if ( rotRI >= 180.f)
                dirRI = false;
        else if ( rotRI <= 0.f)
                dirRI = true;
```

The first thing we see is an IF with the variable “activeR” (boolean), whose value is changed with the R key to its negation. In case activeR is true, the variable “time” (float) is assigned the clock that carries “glfwGetTime ( )”. Then another IF evaluates the direction of the left knob. If the variable “rotRI” (float) passes or equals 180 (the degrees the knob turns), the variable “dirRI” (boolean) changes its value to false; it changes direction. Case that rotRI is less than or equal to 0, dirRI changes to true (changes direction).

```
        if ( RIdir )
                rotRI += 0.5f;
        else
                rotRI -= 0.8f;
```

Case that the variable is dirRI is true, the variable rotRI will increase its value (it will have a positive direction) by 0.5 units cyclically. Else, it will decrease (negative direction) its value by 0.8 units cyclically.

```
        if ( rotRD >= 85.f)
                dirRD = false;
        else if ( rotRD <= 0.f)
                addrRD = true;
```

Similarly, for the right knob, if the variable “rotRD” (float) exceeds or equals 85 units, the variable “dirRD” (boolean) changes the direction of the wheel rotation. If less than or equal to 0, dirRD is set to true.

```
        if ( RDdir )
                rotRD += 1.5f;
        else
                rotRD -= 1.f;
    }
```

When the dirRD variable is true, the rotRD variable increases its value (positive direction), else it decreases its value (negative direction).

```
    else {
        time = 0.f;
    }
```

Finally, if the activeR variable is false, all the variables related to the knobs are not altered and the time variable is assigned to 0 (preventing the shader from fluctuating), simulating that the radio is off.

Code of the “ animacion_bajo()” function.

```
void animacion_bajo(){
    //Movimiento del bajo
    if (circuito){
```

To start the circuit, its necessary that the variable "circuit" (boolean), controlled by the button "B" be True. Else, the tour doesn’t start or be pauses.

```
        if (recorrido1){
                rotKitZ += 0.08+(rotKitZ/10);
                if (rotKitZ > 13.6){
                        recorrido1 = false;
                        recorrido2 = true;
                }
        }
```

If circuit be true, asks if the variable "recorrido1", (boolean) which was declared true, is true. Then, the first step in the animation is to simulate the bass's first collision with the wall, which occurs at 13.6°. To rotate the bass, the variable “rotKitZ” is used, which grows faster and faster to simulate the acceleration of the fall. When the position is reached, it proceeds to the next step in the circuit, changing recorrido1 to false and recorrido2 to true.

```
if (recorrido2){
        rotKitY -= 2.88 - (rotKitY/20);
        if (rotKitY < -15.1){
                recorrido2 = false;
                recorrido3 = true;
        }
}
```

In the second step of the circuit, solve the slope of the bass in the Y axis, aligning with the wall. Once it reaches 15.1° on the Y axis, by increasing the variable rotKitY (float), it moves to the next step, changing recorrido2 to false and recorrido3 to true.

```
if (recorrido3){
        if(movKitX < 0.911)
                movKitX += (0.05061111111112 -movKitX/20);
        rotKitZ += (0.9969444444444444445 - rotKitZ/50);
        if (rotKitZ > 32.5) {
                recorrido3 = false;
                recorrido4 = true;
        }
}
```

In the third step, the bass falls through the wall is simulated, then is simulated the speed decreases through the variable movKitX for its displacement and rotKitZ for its rotation. Once the desired position is reached, it proceeds to the next step, changing recorrido3 to false and recorrido4 to true.

```
if (recorrido4) {
        movKitX += 0.035;
        rotKitZ -= (0.25 + rotKitZ/130);
        if (rotKitZ < 31.0) {
                recorrido4 = false;
                recorrido5 = true;
        }
}
```

In the fourth step, it simulates a bounce. The rotKitZ variable decrease by one and a half degrees, and the offset in X, through the movkitX variable, increase while the desired angle is reached. When the desired position is reached, it proceeds to the next step, changing recorrido4 to false and recorrido5 to true.

```
if (recorrido5) {
        movKitX += 0.0083;
        rotKitZ += (0.05 + (rotKitZ/120));
        if (rotKitZ > 38.8) {
                recorrido5 = false;
                recorrido6 = true;
        }
}
```

In the fifth step, the bass down to 38.8°. The variable rotKitZ is increased exponentially to simulate acceleration. Once the desired position is reached, it proceeds to the next step, changing recorrido5 to false and recorrido6 to true.

```
        if (recorrido6) {
                rotKitZ += (0.05 + (rotKitZ / 120));
                if (rotKitZ > 40.) {
                        recorrido6 = false;
                        recorrido7 = true;
                }
        }
```

In the sixth step, the bass only rotates until it hits the wall again, then, the variable rotKitZ increases exponentially until it reaches 40°. Once the desired position is reached, it proceeds to the next step, changing recorrido6 to false and recorrido7 to true.

```
        if (recorrido7) {
                if(movKitX <= 1.9)
                        movKitX += 0.005 + (movKitX/40);
                rotKitZ += (0.05 + (rotKitZ / 40));
                if (rotKitZ > 72.) {
                        recorrido7 = false;
                        recorrido8 = true;
                }
        }
```

In the seventh step, the bass is shifted in X until it reaches 1.9, while rotating to 72° with exponentially increasing. Once the desired position is reached, it proceeds to the next step, changing recorrido7 to false and recorrido8 to true.

```
        if (recorrido8) {
                rotKitZ = 72.8;
                recorrido8 = false;
                recorrido9 = true;
                circuito = !circuito;
        }
```

In the eighth step, the rotation of the bass is adjusted to touches the ground. Next, proceeds to the next step, changing recorrido8 to false and recorrido9 to true. In addition, the “circuito” variable is changed to false to the animation doesn’t restart.

```
        if (recorrido9) {
                rotKitY = -13.8;
                recorrido9 = false;
                recorrido10 = true;
        }
```

In the ninth step, the rotation of the object on the Y axis is adjusted with the variable rotKitY to the bass touch the ground. It then proceeds to the next step, changing recorrido9 to false and recorrido10 to true.

```
        if (recorrido10 && circuito) {
                movKitX = 0.0;
                rotKitY = 0.0;
                rotKitZ = 0.0;
                recorrido10 = false;
                recorrido1 = true;
        }
}
```

In the last step it resets all used variables. It then returns to the first step, changing recorrido10 to false and recorrido1 to true.

Code of the “ keycallback ” function.

Code – lamp animation:

```
if ( keys [ GLFW_KEY_L]) {
        activeL = ! activeL ;
        if ( activeL ){
                lampOn.x = 1.0f;
                lampOn.y = 1.0f;
                lampOn.z = 0.717647f;
                Light1 = glm :: vec3(1.0f, 1.0f, .0f);
        }
        else {
                lampOn = glm :: vec3(0.0f);
                Light1 = glm :: vec3(0);// When it is only one value in the 3 vectors can leave only one component
        }
    }
```

This conditional is for the animation of the lamp, which can be activated with the “L” key. When the key is pressed and the condition is true, it changes the variable “activeL” (boolean) to its negated; if it is "true", it would change to "false". When activeL is true, the vector (vec3) lampOn is assigned the corresponding values for the lighting color, and the vector (vec3) Light1 is assigned the color yellow. Else, the vectors (vec3) lampOn and Light1 are set to 0, it shows off.

Code – Door animation.

```
    if ( keys [ GLFW_KEY_P ]) {
        activePF = ! activePF ;
    }
```

In this condition, the variable “activePF” (boolean), the responsible for controlling when is the door animation activated. is changed to its negation. If it is "true" it becomes "false" when pressing the "P" key.

Code – Radio animation.

```
    if ( keys [ GLFW_KEY_R]) {
        activeR = ! activeR ;
    }
```

In this condition, the variable “ activeR ” (boolean), responsible for controlling when the Radio animations are activated, is changed to its negated. That is, if it is found as "true" it becomes "false" when pressing the "R" key.

Code – Bass animation.

```
    if ( keys [ GLFW_KEY_B ]) {
        circuit = !circuit ;
    }
}
```

In this condition, the variable “activeB” (boolean), responsible for controlling when the animation of the bass is activated, is changed to its negation. If it is "true" it becomes "false" when pressing the "B" key.

Conclusions.

During the development of the project, including the modeling of the objects, the knowledge acquired throughout the course was used. This will be seen in the progress of the objects, and, especially in the time they were made; some figures at the beginning took a long time even being simple, while others were made in very short times. Also, the management of lighting and the creation of animations denote a good management of knowledge, so that the objectives have been achieved satisfactorily, and they are waiting for new ones in later subjects.