Universidad nacional Autónoma de México

Facultad de Ingeniería

Laboratorio de Computación Grafica e Interacción Humano Computadora

Profesor: Román Balbuena Carlos Aldair

Alumno: Gutiérrez Alcibar Ulises

Proyecto Final

Manual técnico

Fecha límite de entrega: 21 de enero del 2021

Objetivos:

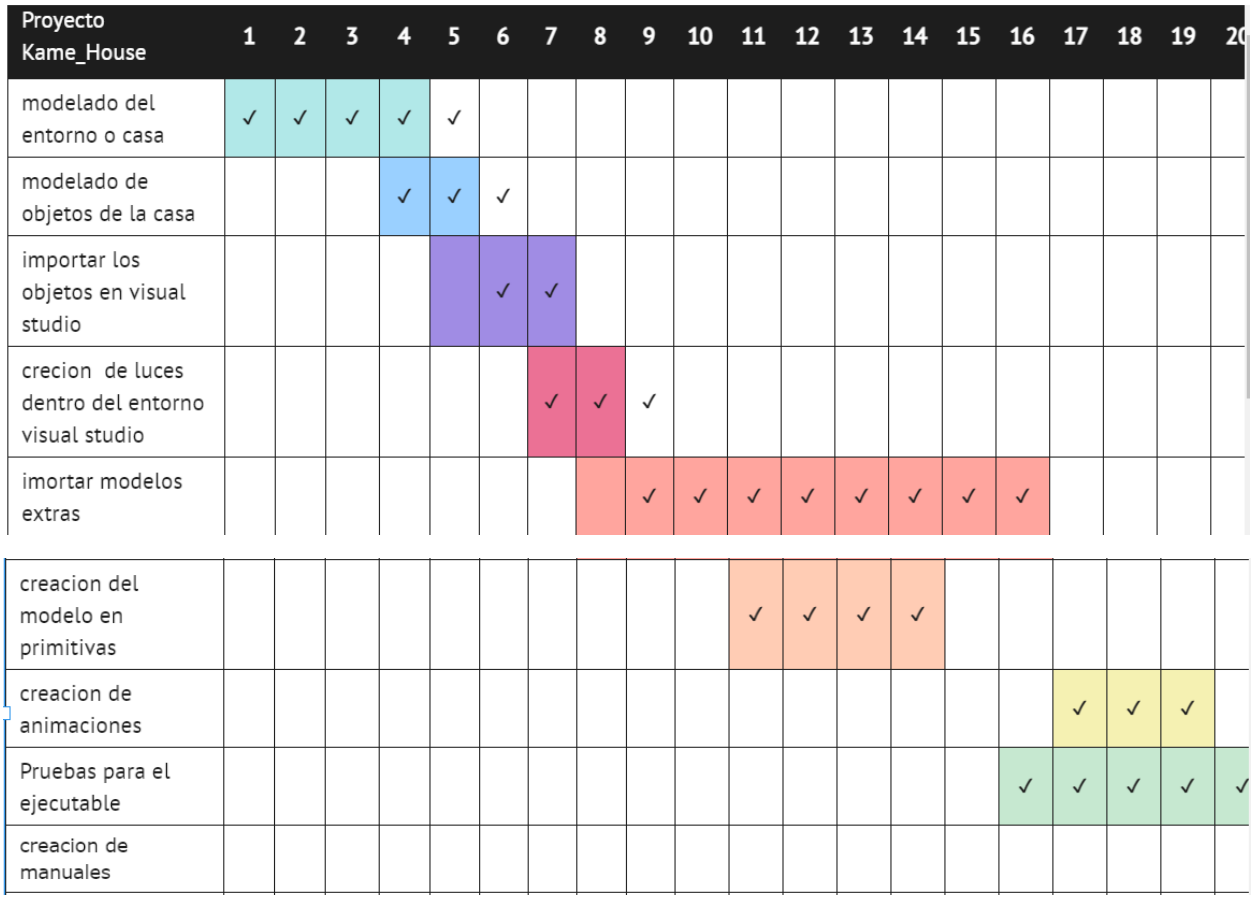
a) Aprender la manipulación apropiada de los conocimientos obtenidos en el laboratorio de computación gráfica, para la elaboración de proyectos posteriores.

b) Aprender a utilizar el modelador maya, para recrear objetos del proyecto.

c) Aprender a animar de distintas maneras modelos dentro de la herramienta de visual studio.

d) Concluir el proyecto de manera exitosa, para autorreconocimiento y aprobación.

Diagrama de Gant:



CODIGO

Cargamos e incluimos todos los archivos necesarios para la ejecución de nuestro proyecto.

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

// Other Libs
#include "stb_image.h"

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

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


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

Declaración de animación con su respectivo nombre y datos que recibe, en el programa se programaron 4 animaciones.

animación: para la manipulación del modelo de la lancha.
animacióntb: para la manipulación del modelo del tiburón martillo.
animaciónpelota: para la manipulación del modelo de la pelota.
Además de una animación implícita para la manipulación del ventilador.

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

Declararemos tanto los parámetro y variables para el manejo apropiado de la cámara, en ejecución de nuestro proyecto con las siguientes líneas de código.

```
// Camera
Camera  camera(glm::vec3(0.0f, 0.0f, 3.0f));
GLfloat lastX = WIDTH / 2.0;
GLfloat lastY = HEIGHT / 2.0;
bool keys[1024];
bool firstMouse = true;
float range = 0.0f;
float rot = 0.0f;
```

Además de declarar las posiciones de inicio de los modelos animados.

```
// posiciones de inicio
glm::vec3 lightPos(0.0f, 0.0f, 0.0f);
glm::vec3 PosIni(28.0f, -1.9f, 30.5f);
glm::vec3 PosInitibu(3.0f, -2.5f, 30.0f);
glm::vec3 PosInipelota(-15.0f, -3.5f, 30.0f);
bool active;
```

Declaramos las posiciones de nuestras points light.

```
 // Positions of the point lights
glm::vec3 pointLightPositions[] = {
    glm::vec3(0.7f,  -60.2f,  2.0f),
    glm::vec3(10.7f,  60.2f,  2.0f),
    glm::vec3(-10.7f,  60.2f,  2.0f),
    glm::vec3(0.7f,  -60.2f,  2.0f)
};
```

Declararemos las siguientes variables para el apropiado manejo de nuestras animaciones.

Para la activación y desactivación del ventilador.

```
bool rotventi = false;
```

Para la manipulación del modelo de la lancha.

```
//Animación lancha
float movKitX = 0.0;
float movKitZ = 0.0;
float rotKit = 0.0;

bool circuito = false;
bool recorrido1 = true;
bool recorrido2 = false;
bool recorrido3 = false;
bool recorrido4 = false;
```

Para la manipulación del modelo del tiburon.

```
//Animacion tibu
float movKitXtb = 0.0;
float movKitYtb = 0.0;
float movKitZtb = 0.0;
float rotKittb = 0.0;

bool circuitotb = false;
bool recorrido1tb = true;
bool recorrido2tb = false;
bool recorrido3tb = false;
bool recorrido4tb = false;
```

Para la manipulación del modelo de la pelota.

```
//Animación pelota

float movKitXa = 0.0;
float movKitZa = 0.0;
float movKitYa = 0.0;
float rotKitaY = 0.0;

bool circuitoa = false;
bool recorrido1a = true;
```

Declaramos variables para nuestra pantalla.

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

Procedemos a definir nuestra función principal main, en la cual se procede a la definición y manejo de la ventana en nuestro proyecto.

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

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

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

        return EXIT_FAILURE;
    }

    glfwMakeContextCurrent(window);

    glfwGetFramebufferSize(window, &SCREEN_WIDTH, &SCREEN_HEIGHT);

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

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

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

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

Mandaremos a llamar a nuestros shader para poder cargar de manera correcta nuestros modelos y manipulación de luces dentro del entorno.

```
Shader lightingShader("Shaders/lighting.vs", "Shaders/lighting.frag");
Shader lampShader("Shaders/lamp.vs", "Shaders/lamp.frag");
```

Procederemos a definir el nombre para cargar nuestros modelos, en las carpetas respectivas.

```
Model lancha((char*)"Models/objetos casa/jetski/jetski.obj");
Model casa((char*)"Models/modelo_house/came.obj");
Model comedor((char*)"Models/objetos casa/silla/comedor.obj");
Model mueble((char*)"Models/objetos casa/mueble/mueble.obj");
Model tele((char*)"Models/objetos casa/tele/tele.obj");
Model sala((char*)"Models/objetos casa/sillon/sala.obj");
Model lampara((char*)"Models/objetos casa/lampara/lampara.obj");
Model tiburon((char*)"Models/objetos casa/tibu/tiburon.obj");
Model venti((char*)"Models/objetos casa/ventilador/ventilador.obj");
Model pelota((char*)"Models/pelota/pelota.obj");
```

Definimos los vértices para la creación de un cubo, el cual texturizaremos para recrear un objeto por medio de primitivas.

```
GLfloat vertices[] =
{
    // Positions           // Normals             // Texture Coords
    -0.5f, -0.5f, -0.5f,    0.0f,  0.0f, -1.0f,    0.0f,  0.0f,
    0.5f, -0.5f, -0.5f,     0.0f,  0.0f, -1.0f,    1.0f,  0.0f,
    0.5f,  0.5f, -0.5f,     0.0f,  0.0f, -1.0f,    1.0f,  1.0f,
    0.5f,  0.5f, -0.5f,     0.0f,  0.0f, -1.0f,    1.0f,  1.0f,
    -0.5f,  0.5f, -0.5f,    0.0f,  0.0f, -1.0f,    0.0f,  1.0f,
    -0.5f, -0.5f, -0.5f,    0.0f,  0.0f, -1.0f,    0.0f,  0.0f,

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

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

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

Definimos nuestro modo de dibujo, para el cubo.

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

Para la correcta construcción de nuestro cubo, estaremos manejando el VAO,EBO, además de unas funciones llamada array_buffer, vertexarray, para las posiciones, normales y texturas.

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

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

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

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

// Then, we set the light's VAO (VBO stays the same. After all, the vertices are the same for the light object (also a 3D cube))
GLuint lightVAO;
glGenVertexArrays(1, &lightVAO);
glBindVertexArray(lightVAO);
// We only need to bind to the VBO (to link it with glVertexAttribPointer), no need to fill it; the VBO's data already contains all we need.
glBindBuffer(GL_ARRAY_BUFFER, VBO);
// Set the vertex attributes (only position data for the lamp))
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (GLvoid *)0); // Note that we skip over the other data in our buffer object (we don't need the normals/textures, only positions)
glEnableVertexAttribArray(0);
glBindVertexArray(0);
```

Definiremos nuestra matriz de proyección para la cámara.

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

Definimos las texturas utilizadas en el cubo para recrear nuestro objeto en primitivas.

```
GLuint texture1,texture2;
glGenTextures(1, &texture1);
glGenTextures(1, &texture2);


int textureWidth, textureHeight, nrChannels;
stbi_set_flip_vertically_on_load(true);
unsigned char* image;
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR_MIPMAP_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST_MIPMAP_NEAREST);
// Diffuse map
image = stbi_load("images/venti.jpg", &textureWidth, &textureHeight, &nrChannels, 0);
glBindTexture(GL_TEXTURE_2D, texture1);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, textureWidth, textureHeight, 0, GL_RGB, GL_UNSIGNED_BYTE, image);
glGenerateMipmap(GL_TEXTURE_2D);
if (image)
{
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, textureWidth, textureHeight, 0, GL_RGB, GL_UNSIGNED_BYTE, image);
    glGenerateMipmap(GL_TEXTURE_2D);
}
else
{
    std::cout << "Failed to load texture" << std::endl;
}
stbi_image_free(image);

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR_MIPMAP_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST_MIPMAP_NEAREST);
// Diffuse map
image = stbi_load("images/blanca.jpg", &textureWidth, &textureHeight, &nrChannels, 0);
glBindTexture(GL_TEXTURE_2D, texture2);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, textureWidth, textureHeight, 0, GL_RGB, GL_UNSIGNED_BYTE, image);
glGenerateMipmap(GL_TEXTURE_2D);
if (image)
{
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, textureWidth, textureHeight, 0, GL_RGB, GL_UNSIGNED_BYTE, image);
    glGenerateMipmap(GL_TEXTURE_2D);
}
else
{
    std::cout << "Failed to load texture" << std::endl;
}
stbi_image_free(image);
```

Creamos un ciclo while para la ejecución continua de nuestro programa, en el cual mandaremos a llamar a nuestras funciones y el movimiento en la ventana.

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

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

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

Posteriormente procedemos a definir las luces utilizadas, que son una direcciónal, una ambiental y 4 puntos de luz.

```
// Directional light
glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.direction"), -0.2f, -1.0f, -0.3f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.ambient"), 0.93f, 0.93f, 0.93f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.diffuse"), 0.4f, 0.4f, 0.4f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.specular"), 0.5f, 0.5f, 0.5f);


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



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

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

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

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

Procedemos a cargar los modelos en sus respectivas posiciones.

```
//Carga de modelo

//lancha

view = camera.GetViewMatrix();
model = glm::mat4(1);
model = glm::translate(model, PosIni + glm::vec3(movKitX, 0,movKitZ));
model = glm::rotate(model, glm::radians(rotKit), glm::vec3(0.0f, 1.0f, 0.0));
model = glm::scale(model, glm::vec3(0.1f, 0.1f, 0.1f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
lancha.Draw(lightingShader);

//casa

glm::mat4 model1(1);
model1 = glm::translate(model1, glm::vec3(0.0f, -1.75f, 0.0f));
model1 = glm::scale(model1, glm::vec3(0.02f, 0.02f, 0.02f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model1));
casa.Draw(lightingShader);

//modelo mesa

glm::mat4 model2(1);
model2 = glm::translate(model2, glm::vec3(-2.0f, -1.2f, -1.5f));
model2 = glm::scale(model2, glm::vec3(0.025f, 0.025f, 0.025f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model2));
comedor.Draw(lightingShader);

//cargando modelo mueble
glm::mat4 model3(1);
model3 = glm::translate(model3, glm::vec3(6.5f, -1.2f, 2.0f));
model3 = glm::scale(model3, glm::vec3(0.01f, 0.025f, 0.025f));
model3 = glm::rotate(model3, 11.0f, glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model3));
mueble.Draw(lightingShader);


//cargando modelo tele
glm::mat4 model4(1);
model4 = glm::translate(model4, glm::vec3(6.5f, -0.2f, 2.0f));
model4 = glm::scale(model4, glm::vec3(0.01f, 0.015f, 0.025f));
model4 = glm::rotate(model4, 11.0f, glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model4));
tele.Draw(lightingShader);

//cargando modelo sala
glm::mat4 model5(1);
model5 = glm::translate(model5, glm::vec3(1.0f, -1.2f, 2.0f));
model5 = glm::scale(model5, glm::vec3(0.040f, 0.04f, 0.04f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model5));
sala.Draw(lightingShader);

//cargando modelo lampara
glm::mat4 model6(1);
model6 = glm::translate(model6, glm::vec3(3.0f, -1.2f, 6.5f));
model6 = glm::scale(model6, glm::vec3(0.015f, 0.015f, 0.015f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model6));
lampara.Draw(lightingShader);

//cargando modelo tiburon
glm::mat4 model7(1);
model7 = glm::translate(model7, PosInitibu + glm::vec3(movKitXtb, movKitYtb, movKitZtb));
model7 = glm::rotate(model7, glm::radians(rotKittb), glm::vec3(0.0f, 1.0f, 0.0));
model7 = glm::scale(model7, glm::vec3(0.15f, 0.15f, 0.15f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model7));
tiburon.Draw(lightingShader);
```

```
//cargando modelo ventilador
glm::mat4 model9(1);
model9 = glm::translate(model9, glm::vec3(1.0f, 2.3f, 2.5f));
model9 = glm::scale(model9, glm::vec3(0.017f, 0.017f, 0.017f));
if (rotventi==true) {

    model9 = glm::rotate(model9, (float)glfwGetTime(), glm::vec3(0.0f, 1.0f, 0.0f));
}
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model9));
venti.Draw(lightingShader);

//cargando modelo pelota
glm::mat4 model10(1);
model10 = glm::translate(model10, PosInipelota+ glm::vec3(movKitXa, movKitYa, movKitZa));
model10 = glm::scale(model10, glm::vec3(0.020f, 0.020f, 0.020f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model10));
pelota.Draw(lightingShader);
```

Crearemos nuestro modelo a primitivas utilizando el cubo anteriormente definido, además de asignarle tanto la textura1 o textura2.

```
//obj primitivas basicas
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, texture2);
glBindVertexArray(VAO);
glm::mat4 model11(1);
model11 = glm::translate(model11, glm::vec3(1.4f, -0.8f, 1.2f));
model11 = glm::scale(model11, glm::vec3(0.15f, 0.7f, 0.15f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model11));
glDrawArrays(GL_TRIANGLES, 0, 36);

glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, texture2);
glBindVertexArray(VAO);
glm::mat4 model13(1);
model13 = glm::translate(model13, glm::vec3(1.4f, -0.8f, 3.0f));
model13 = glm::scale(model13, glm::vec3(0.15f, 0.7f, 0.15f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model13));
glDrawArrays(GL_TRIANGLES, 0, 36);

glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, texture2);
glBindVertexArray(VAO);
glm::mat4 model14(1);
model14 = glm::translate(model14, glm::vec3(3.6f, -0.8f, 1.2f));
model14 = glm::scale(model14, glm::vec3(0.15f, 0.7f, 0.15f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model14));
glDrawArrays(GL_TRIANGLES, 0, 36);

glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, texture2);
glBindVertexArray(VAO);
glm::mat4 model15(1);
model15 = glm::translate(model15, glm::vec3(3.6f, -0.8f, 3.0f));
model15 = glm::scale(model15, glm::vec3(0.15f, 0.7f, 0.15f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model15));
glDrawArrays(GL_TRIANGLES, 0, 36);
```

Posteriormente pasamos a la definió de funciones, en primer caso la función Devomet, que nos permite recibir o leer entradas en nuestro teclado, estaremos definiendo ciertos funcionamientos al presionar, teclas específicas.

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

    if (keys[GLFW_KEY_1])
    {
        range += 0.1;
        rot += 1;
        printf("El rango es %f\n", range);
    }

    if (keys[GLFW_KEY_2])
    {
        range -= 0.1;
        printf("El rango es %f\n", range);
    }

    if (keys[GLFW_KEY_3])
    {
        range += 0.1;
        printf("El spotangle es %f\n", range);
    }

    if (keys[GLFW_KEY_4])
    {
        range -= 0.1;
        printf("El spotangle es %f\n", range);
    }

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

    }
```

Dentro de esta función estaremos definiendo como teclas de control para la cámara a W,S,A,D para poder movernos adelante ,atrás, izquierda y derechas, respectivamente.

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

}

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


}

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


}

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


}
```

Las siguientes teclas nos sirven para activar o desactivar animaciones:

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

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

if (keys[GLFW_KEY_N])
{
    circuitotb = false;
}

if (keys[GLFW_KEY_M])
{
    circuitotb = true;
}

if (keys[GLFW_KEY_V])
{
    circuitoa = false;
}

if (keys[GLFW_KEY_B])
{
    circuitoa = true;
}

if (keys[GLFW_KEY_T])
{
    rotventi = false;
}

if (keys[GLFW_KEY_G])
{
    rotventi = true;
}
```

I,0 : para activar y desactivar la animación de la lancha.

N,M : para activar y desactivar la animación del tiburón.

B,V : para activar y desactivar la animación de la pelota.

T,G : para activar y desactivar la animación del ventilador.

Procedemos a la definición de las animaciones:

Animación para manipular la lancha

```
void animacion()
{   //Movimiento lancha
    if (circuito)
    {   if (recorrido1)
        {   rotKit = 0;
            movKitZ -= 0.1f;
            if (movKitZ < -62.0)
            {   recorrido1 = false;
                recorrido2 = true;
            }
        }
        if (recorrido2)
        {   rotKit = 90;
            movKitX -= 0.1f;
            if (movKitX < -57.0)
            {   recorrido2 = false;
                recorrido3 = true;
            }
        }
        if (recorrido3)
        {   rotKit = 180;
            movKitZ += 0.1f;
                if (movKitZ > 5.0)
                {
                    recorrido3 = false;
                    recorrido4 = true;
                }
        }
        if (recorrido4)
        {   rotKit = 270;
            movKitX += 0.1f;
            if (movKitX > 5.0)
            {   recorrido4 = false;
                recorrido1 = true;
            }
        }
    }
}
```

Definición de la animación del tiburón

```
void animaciontb()
{   //Movimiento tibu
    if (circuitotb)
    {   if (recorrido1tb)
        {   rotKittb = 0;
            movKitXtb -= 0.1f;
            movKitYtb -= 0.01f;
            if (movKitXtb < -35.0 && movKitYtb < -0.2)
            {   recorrido1tb = false;
                recorrido2tb = true;
            }
        }
        if (recorrido2tb)
        {   rotKittb = 90;
            movKitZtb += 0.1f;
            movKitYtb += 0.1f;
            if (movKitZtb > 2.0 && movKitYtb > 0.0)
            {   recorrido2tb = false;
                recorrido3tb = true;
            }
        }
        if (recorrido3tb)
        {   rotKittb = 180;
            movKitXtb += 0.1f;
            movKitYtb -= 0.01;
            if (movKitXtb > 0.0 && movKitYtb < -2.0)
            {recorrido3tb = false;
            recorrido4tb = true;
            }
        }
        if (recorrido4tb)
        {   rotKittb = 270;
            movKitZtb -= 0.1f;
            movKitYtb += 0.1f;
            if (movKitZtb < 0.0 && movKitYtb > 0.0)
            {   recorrido4tb = false;
                recorrido1tb = true;
            }
        }
    }
}
```

Definición de la animación de la pelota:

```
void animacionpelota()
{   //Movimiento pelota
    if (circuitoa)
    {
        if (recorrido1a) {
            movKitXa = 5 * cos((float)glfwGetTime()*0.3 );
            movKitYa = 1 * sin((float)glfwGetTime()*3 );
            movKitZa = 5 * cos((float)glfwGetTime() * 3);
        }

    }
}
```