Universidad Nacional Autónoma de México

Facultad de Ingeniería

Proyecto de responsabilidad Social:

Manual técnico

Laboratorio de Computación gráfica e interacción humano-computadora

Maestro: Carlos Aldair

Alumno: López Martínez Genaro

Gpo. 12

Semestre 2022-2

Objetivo

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

The student must apply and show the acquired acknowledgments from the course.

Descripción

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

En la imagen de referencia se debe visualizar 7 objetos que el alumno va a recrear virtualmente y donde dichos objetos deben ser lo más parecido a su imagen de referencia, así como su ambientación. Se debe subir un documento pdf donde muestre claramente su fachada y el cuarto a recrear, así como también un listado de los 7 objetos que se van a desarrollar dentro de dicho. Este documento se debe subir antes del 14 de marzo.

The student must select a facade and a space, that can be real or fiction, and present reference pictures from that space to recreate them on OpenGL as 3D.

In the reference picture, must visualize 7 objects which the student will recreate virtually and will appear as close as the real ones, even its environment. It must upload a pdf file which shows clearly the facade and the room to recreate, with the 7 modeled objects. This file must be uploaded before 03/14/2022.

Desarrollo del proyecto

Diagrama de Gantt

Para el desarrollo del proyecto se necesitó de un esquema para poder organizar las diferentes actividades del proyecto final. Pues como no era un proyecto donde solo se le dedica cierto tiempo a final de semestre sino más bien es un seguimiento diario al proyecto donde se necesitarán de diferentes versiones para su avance.

Es por eso que se toma en cuenta como fecha inicial Marzo 14 del 2022 como partida del proyecto pues se creó una propuesta de proyecto, así como sus objetos y animaciones.

Finalmente se puede ver que se terminó como fecha máxima el 12 de Mayo del 2022 para poder entregar en tiempo y forma el proyecto final.

For the development from the project, it required a scheme to manage the different activities from the final project. This was not only a project where you just invest time at the end of the semester, you need to keep attention and the development everyday to have different versions from the project.

That is why, I tooked as a inicial date 03/14/2022, because that was the day where I uploaded the proposal file with the objects and animations.

Finally I set a deadline 05/12/2022 to deliver it in time.

Proyecto Final	Fecha Inicio	Fecha Fin	14-21 Marzo	21-28 Marzo	1-11 Abril	11-18 Abril	18-25 Abril	25-30 Abril	2-9 Mayo	9-16 Mayo
Proyecto Final										
Planeación Historia	Marzo 2022	Marzo 2022								
Planeación Objetos	Marzo 2022	Marzo 2022								
Planeación Animaciones	Marzo 2022	Marzo 2022								
Desarrollo de proyecto										
Modelar objetos	Marzo 2022	mayo 2022								
Implementar animaciones	Abril 2022	mayo 2022								
Texturizar modelos	Abril 2022	mayo 2022								
Modelar fachada	Abril 2022	mayo 2022								
Documentación										
Documentación del código	mayo 2022	mayo 2022								
Manual de usuario	mayo 2022	mayo 2022								
Manual técnico	mayo 2022	mayo 2022								
Corregir Proyecto										
Modelar objetos	mayo 2022	mayo 2022								
Implementar animaciones	mayo 2022	mayo 2022								
Texturizar modelos	mayo 2022	mayo 2022								
Modelar fachada	mayo 2022	mayo 2022								

Alcance del proyecto

Este proyecto tiene como alcance la demostración de diferentes conceptos importantes dentro de la animación de gráficos por computadoras así como la interacción con el usuario.

Al mostrar el modelado de la caricatura de “los jóvenes titanes” se puede ver en primera instancia el cuarto del interior donde se ven los 7 modelos dentro del cuarto. Al tener los modelos se pueden apreciar el concepto de carga de modelos dentro de OpenGL. También se puede ver la texturización de los modelos por medio del software de modelado para cargar las texturas en OpenGL. Adicionalmente se puede ver el concepto de iluminación dentro de los materiales pues se tiene las propiedades de brillo en algunos.

Por otro lado para poder ver animaciones se pueden ver por ejemplo en el uso de la activación de la alarma de emergencia. Donde solamente se hará una traslación a la pantalla de la televisión y el point light que está sujeto en el modelo de la lámpara se encenderá con una distancia calculada teniendo en cuenta los valores cuadráticos del área.

Entonces se puede decir que el alcance del proyecto puede mostrar aspectos básicos de la animación de modelos por medio de OpenGL y no obstante también se mostrará la interacción que puede tener el usuario dentro del programa. Tal vez algo limitada pero podrá entender el funcionamiento de cada animación dentro del contexto del programa.

This project has to show different and important topics inside the graphics animation by computers and interact with the user.

By showing the modeling cartoon “teen titans” you can see at first sight, the room with the 7 objects where you can see the topic as loading the models into OpenGL. Additionally you can see the texturization topic into OpenGL on the objects and nevertheless you can see in some bright and shining materials from some objects, the light and shininess topic.

So to sum up, the project is able to show basic aspects from the modeling animation into OpenGL and also it’ll show the interaction between the user and the program. Maybe is limited but is understandable the animations into the environment

Limitantes

Las limitaciones del proyecto pueden ser tanto físicas como de implementación. Primero se tiene que ver por la parte física porque esta versión actual del programa sólo tiene de peso 500 MB aproximadamente que para que funcione bien toma bastante tiempo y si se llegará a implementar una mejor versión se necesitaría de un mayor equipo o tratar de eliminar y eficientar el peso del programa para mantenerse dentro de la ejecución.

Otra limitante sería dentro de la implementación pues al solo poder mostrar los objetos y el lugar del modelado tal vez faltó más para poder implementar otras cosas y que se pueda entender de mejor manera el contexto. Por ejemplo, como está basado en el edificio de “los jóvenes titanes” se tiene solamente modelado los muebles y el interior de la base, pero no se tiene los efectos de sonidos, los personajes modelados.

Entonces se puede decir que este programa se puede mejorar, pero también se tiene que controlar los recursos que se invertirán y en qué se invertirán.

The limits from this project can be as physical and as programming.

First, you need to see that the size from the program is approximately 500 MB to work, but it takes a while to load the program and if you can improve the program you will need better equipment or manage the size from the objects into the program.

Another limit would be inside the implementation, because for showing the object and the room, it may miss other things to give more sense to the environment and the model. For example, due it was based on “teen titans' ', it only has the furniture from the main room but you don’t have the sound effects and the characters from the cartoon.

Documentación del código

```
/**
 * @file   316141488_ProyectoFinal_GPO12.cpp
 * @Author Genaro López Martínez
 * @date   11/05/2022
 * @brief  Final project to show the theorical and practical topics from the course
 *
 * This file has the models from the proposed model to animate in 3D with OpenGL
 * Also there are 7 objects with 5 animtaions into the context from the model
 */
// Std. Includes
#include <string>
```

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

// GLFW
#include <GLFW/glfw3.h>
// Other Libs
#include "SOIL2/SOIL2.h"
#include "stb_image.h"
// GL includes
#include "Shader.h"
#include "Camera.h"
#include "Model.h"

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



// Properties
const GLuint WIDTH = 1200, HEIGHT = 800;
int SCREEN_WIDTH, SCREEN_HEIGHT;

// 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 nadar();
void musica();
```

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

//variables to rotate or translate as a nimation some objects
float rot = 0.0f;
float rotem = 0.0f;
float rotna = 0.0f;
float aumento = 0.0f;
float distancia = 0.0f;
float distancia2 = 0.0f;
bool active, active2;
glm::vec3 lightPos(0.0f, 0.0f, 0.0f);
float px = 0, py = 0, pz = 0, dx = 0, dy = 0, dz = 0;

// variables to use as flags or conditions to activate or disactivate some conditional to make a process
bool bocinas = false;

bool circuito = false;
bool recorrido1 = true;
bool recorrido2 = false;
bool recorrido3 = false;
bool recorrido4 = false;
bool recorrido5 = false;
bool recorrido6 = true;
bool recorrido7 = false;
float tiempo;
```

```
// Positions of the point lights
/*<summary> an array which initialize
* the point lights positions </summary>*/
glm::vec3 pointLightPositions[] = {
    glm::vec3(0.0f,16.0f, 0.0f),
    glm::vec3(-11.0f,1.3f, -7.7f),
    glm::vec3(0.0f,0.0f,  12.0f),
    glm::vec3(11.5f,0.0f, 11.5f)
};

/*<summary> an array which initialize
* the faces from the object </summary>*/
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
};



glm::vec3 Light1 = glm::vec3(0);
glm::vec3 Light2 = glm::vec3(0);
glm::vec3 Light3 = glm::vec3(0);
```

```
glm::vec3 Light4 = glm::vec3(0);

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

/*<summary> Main function where ypu can see all the
usage from every variable and procedures with the models </summary>
<returns> return a 0 as execution completed correctly </returns>*/
int main()
{
    //<code>
    // 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 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 required callback functions
    glfwSetKeyCallback(window, KeyCallback);
    glfwSetCursorPosCallback(window, MouseCallback);

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

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

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

    // OpenGL options
    glEnable(GL_DEPTH_TEST);

    // Setup and compile our shaders
    Shader lightingShader("Shaders/lighting.vs", "Shaders/lighting.frag");
```

```
Shader shader("Shaders/modelLoading.vs", "Shaders/modelLoading.frag");
Shader lampshader("Shaders/lamp2.vs", "Shaders/lamp2.frag");
Shader anim("Shaders/anim.vs", "Shaders/anim.frag");
Shader anim2("Shaders/anim2.vs", "Shaders/anim2.frag");



// Load models
Model tv((char*)"Models/tv/tv.obj");
Model emergency((char*)"Models/tv/emergency.obj");
Model fondo((char*)"Models/tv/fondopantalla.obj");
Model nave((char*)"Models/tv/navepantalla.obj");
Model sofa((char*)"Models/sofa/sofa.obj");
Model table((char*)"Models/table/table.obj");
Model cocina((char*)"Models/cocina/cocina.obj");
Model agua((char*)"Models/pecera/agua.obj");
Model base((char*)"Models/pecera/base.obj");
Model reflect((char*)"Models/pecera/reflect.obj");
Model colapez1((char*)"Models/pecera/colapez1.obj");
Model colapez2((char*)"Models/pecera/colapez2.obj");
Model cpez1((char*)"Models/pecera/cpez1.obj");
Model cpez2((char*)"Models/pecera/cpez2.obj");
Model pez1((char*)"Models/pecera/pez1.obj");
Model pez2((char*)"Models/pecera/pez2.obj");
Model vidrio((char*)"Models/pecera/vidrio.obj");
Model bocina((char*)"Models/bocina/bocina.obj");
Model speakers((char*)"Models/bocina/speakers.obj");
Model computadora((char*)"Models/computadora/computadora.obj");
Model torre((char*)"Models/torre/torre.obj");
Model ventanas((char*)"Models/torre/ventanas.obj");
```

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


    // 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(vertices), vertices, GL_STATIC_DRAW);


    // Position attribute
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (GLvoid*)0);
    glEnableVertexAttribArray(0);
    // color attribute
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(float), (void*)(3 * sizeof(float)));
    glEnableVertexAttribArray(1);
    // texture coord attribute
    glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 8 * sizeof(float), (void*)(6 * sizeof(float)));
    glEnableVertexAttribArray(2);


    lightingShader.Use();
    glUniform1i(glGetUniformLocation(lightingShader.Program, "material.diffuse"), 1);
    glUniform1i(glGetUniformLocation(lightingShader.Program, "material.specular"), 1);
```

```
    // Load textures
    Model pokearriba((char*)"Models/pokeball/poke_arriba.obj");
    Model pokeabajo((char*)"Models/pokeball/poke_abajo.obj");


    GLuint texture;
    glGenTextures(1, &texture);
    glBindTexture(GL_TEXTURE_2D, texture);
    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);


    /*image = stbi_load("images/gok.png", &textureWidth, &textureHeight, &nrChannels, 0);
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, textureWidth, textureHeight, 0,
GL_RGBA, GL_UNSIGNED_BYTE, image);
    glGenerateMipmap(GL_TEXTURE_2D);
    if (image)
    {
        glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, textureWidth, textureHeight, 0,
GL_RGBA, GL_UNSIGNED_BYTE, image);
        glGenerateMipmap(GL_TEXTURE_2D);
    }
    else
    {
        std::cout << "Failed to load texture" << std::endl;
    }
```

```
    stbi_image_free(image);*/



    // Game loop
    while (!glfwWindowShouldClose(window))
    {
        // Set frame time
        GLfloat currentFrame = glfwGetTime();
        deltaTime = currentFrame - lastFrame;
        lastFrame = currentFrame;

        // Check and call events
        glfwPollEvents();
        DoMovement();
        nadar();
        musica();

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

        shader.Use();
        lightingShader.Use();
        /*GLint viewPosLoc = glGetUniformLocation(lightingShader.Program, "viewPos");
        glUniform3f(viewPosLoc, camera.GetPosition().x, camera.GetPosition().y,
camera.GetPosition().z);*/
        // Directional light
        glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.direction"), -0.2f,
-1.0f, -0.3f);
        glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.ambient"), 0.75f,
0.75f, 0.75f);
```

```
    glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.diffuse"), 0.1f, 0.1f, 0.1f);
    glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.specular"), 1.0f, 1.0f, 1.0f);



    // Point light 1
    glm::vec3 lightColor;
    lightColor.x = abs(sin(glfwGetTime() * Light1.x));
    lightColor.y = abs(sin(glfwGetTime() * Light1.y));
    lightColor.z = sin(glfwGetTime() * Light1.z);



    glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].position"), pointLightPositions[0].x, pointLightPositions[0].y, pointLightPositions[0].z);
    glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].ambient"), lightColor.x, lightColor.y, lightColor.z);
    glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].diffuse"), lightColor.x, lightColor.y, lightColor.z);
    glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].specular"), 1.0f, 1.0f, 1.0f);
    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
    lightColor.x = abs(sin(glfwGetTime() * Light2.x));
    lightColor.y = abs(sin(glfwGetTime() * Light2.y));
    lightColor.z = sin(glfwGetTime() * Light2.z);
```

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

    glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[1].ambient"), lightColor.x, lightColor.y, lightColor.z);

    glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[1].diffuse"), lightColor.x, lightColor.y, lightColor.z);

    glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[1].specular"), 1.0f, 1.0f, 1.0f);

    glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[1].constant"), 1.0f);

    glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[1].linear"), 3.5f);

    glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[1].quadratic"), 5.4f);


    // Point light 3

    lightColor.x = abs(sin(glfwGetTime() * Light3.x));

    lightColor.y = abs(sin(glfwGetTime() * Light3.y));

    lightColor.z = sin(glfwGetTime() * Light3.z);

    glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[2].position"), pointLightPositions[2].x, pointLightPositions[2].y, pointLightPositions[2].z);

    glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[2].ambient"), lightColor.x, lightColor.y, lightColor.z);

    glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[2].diffuse"), lightColor.x, lightColor.y, lightColor.z);

    glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[2].specular"), 1.0f, 1.0f, 1.0f);

    glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[2].constant"), 1.0f);

    glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[2].linear"), 0.7f);

    glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[2].quadratic"), 1.8f);


    // Point light 4

    lightColor.x = abs(sin(glfwGetTime() * Light4.x));
```

```
        lightColor.y = abs(sin(glfwGetTime() * Light4.y));
        lightColor.z = sin(glfwGetTime() * Light4.z);
        glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[3].position"), pointLightPositions[3].x, pointLightPositions[3].y, pointLightPositions[3].z);
        glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[3].ambient"), lightColor.x, lightColor.y, lightColor.z);
        glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[3].diffuse"), lightColor.x, lightColor.y, lightColor.z);
        glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[3].specular"), 1.0f, 1.0f, 1.0f);
        glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[3].constant"), 1.0f);
        glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[3].linear"), 0.35f);
        glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[3].quadratic"), 0.44f);


        // SpotLight
        glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight.position"), 6.0 + px, 0 + py, 6.0 + pz);
        glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight.direction"), 6.0 + dx, 0 + dy, 6.0 + dz);
        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"), 0.44f);
        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 material properties
glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 32.0f);

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

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

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



view = camera.GetViewMatrix();
glUniformMatrix4fv(glGetUniformLocation(shader.Program, "projection"), 1, GL_FALSE, glm::value_ptr(projection));
glUniformMatrix4fv(glGetUniformLocation(shader.Program, "view"), 1, GL_FALSE, glm::value_ptr(view));

// Draw the loaded model
glm::mat4 model(1);
//model = glm::rotate(model, glm::radians(180.0f), glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(glGetUniformLocation(shader.Program, "model"), 1, GL_FALSE, glm::value_ptr(model));
sofa.Draw(shader);

model = glm::mat4(1);
```

```
	//model = glm::translate(model, glm::vec3(1.0f, 3.0f, -20.0f));
	//model = glm::rotate(model, glm::radians(-90.0f), glm::vec3(0.0f, 1.0f, 0.0f));
	glUniformMatrix4fv(glGetUniformLocation(shader.Program, "model"), 1, GL_FALSE, glm::value_ptr(model));
	tv.Draw(shader);

	model = glm::mat4(1);
	model = glm::translate(model, glm::vec3(0.0f, 0.0f, rotem));
	//model = glm::rotate(model, glm::radians(-90.0f), glm::vec3(0.0f, 1.0f, 0.0f));
	glUniformMatrix4fv(glGetUniformLocation(shader.Program, "model"), 1, GL_FALSE, glm::value_ptr(model));
	emergency.Draw(shader);

	//model = glm::mat4(1);
	////model = glm::translate(model, glm::vec3(1.0f, 3.0f, -20.0f));
	////model = glm::rotate(model, glm::radians(-90.0f), glm::vec3(0.0f, 1.0f, 0.0f));
	//glUniformMatrix4fv(glGetUniformLocation(shader.Program, "model"), 1, GL_FALSE, glm::value_ptr(model));
	//fondo.Draw(shader);

	model = glm::mat4(1);
	//model = glm::translate(model, glm::vec3(1.0f, 3.0f, -20.0f));
	//model = glm::rotate(model, glm::radians(-90.0f), glm::vec3(0.0f, 1.0f, 0.0f));
	model = glm::translate(model, glm::vec3(0.0f, 0.0f, rotna));
	glUniformMatrix4fv(glGetUniformLocation(shader.Program, "model"), 1, GL_FALSE, glm::value_ptr(model));
	nave.Draw(shader);

	model = glm::mat4(1);
	//model = glm::translate(model, glm::vec3(1.0f, 0.1f, -8.0f));
	//model = glm::rotate(model, glm::radians(-180.0f), glm::vec3(0.0f, 1.0f, 0.0f));
	glUniformMatrix4fv(glGetUniformLocation(shader.Program, "model"), 1, GL_FALSE, glm::value_ptr(model));
```

```
	computadora.Draw(shader);


	model = glm::mat4(1);
	//model = glm::translate(model, glm::vec3(-5.0f, 0.0f, 3.0f));
	//model = glm::rotate(model, glm::radians(-90.0f), glm::vec3(0.0f, 1.0f, 0.0f));
	glUniformMatrix4fv(glGetUniformLocation(shader.Program, "model"), 1, GL_FALSE, glm::value_ptr(model));
	table.Draw(shader);


	model = glm::mat4(1);
	//model = glm::translate(model, glm::vec3(8.0f, 0.0f, 3.0f));
	//model = glm::rotate(model, glm::radians(90.0f), glm::vec3(0.0f, 1.0f, 0.0f));
	glUniformMatrix4fv(glGetUniformLocation(shader.Program, "model"), 1, GL_FALSE, glm::value_ptr(model));
	cocina.Draw(shader);


	//model = glm::mat4(1);
	////model = glm::translate(model, glm::vec3(15.0f, 1.4f, 4.3f));
	////model = glm::rotate(model, glm::radians(-90.0f), glm::vec3(0.0f, 1.0f, 0.0f));
	//glUniformMatrix4fv(glGetUniformLocation(shader.Program, "model"), 1, GL_FALSE, glm::value_ptr(model));
	//glUniform1f(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 1);
	//glUniform4f(glGetUniformLocation(lightingShader.Program, "coloralpha"), 1.0f, 1.0f, 1.0f, 0.05f);
	//agua.Draw(shader);


	//model = glm::mat4(1);
	////model = glm::translate(model, glm::vec3(18.45f, 1.4f, 4.3f));
	////model = glm::rotate(model, glm::radians(-90.0f), glm::vec3(0.0f, 1.0f, 0.0f));
	//glUniformMatrix4fv(glGetUniformLocation(shader.Program, "model"), 1, GL_FALSE, glm::value_ptr(model));
	//glUniform1f(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 1);
```

```
	//glUniform4f(glGetUniformLocation(lightingShader.Program, "coloralpha"), 1.0f, 1.0f, 1.0f, 0.05f);

	//vidrio.Draw(shader);


	model = glm::mat4(1);

	//model = glm::translate(model, glm::vec3(15.0f, 1.4f, 4.3f));

	//model = glm::rotate(model, glm::radians(-90.0f), glm::vec3(0.0f, 1.0f, 0.0f));

	glUniformMatrix4fv(glGetUniformLocation(shader.Program, "model"), 1, GL_FALSE, glm::value_ptr(model));

	base.Draw(shader);


	//model = glm::mat4(1);

	////model = glm::translate(model, glm::vec3(15.0f, 1.4f, 4.3f));

	////model = glm::rotate(model, glm::radians(-90.0f), glm::vec3(0.0f, 1.0f, 0.0f));

	//glUniform1f(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 1);

	//glUniform4f(glGetUniformLocation(lightingShader.Program, "coloralpha"), 1.0f, 1.0f, 1.0f, 0.05f);

	//glUniformMatrix4fv(glGetUniformLocation(shader.Program, "model"), 1, GL_FALSE, glm::value_ptr(model));

	//reflect.Draw(shader);


	model = glm::mat4(1);

	//model = glm::translate(model, glm::vec3(15.0f, 1.5f, 4.1f));

	//model = glm::rotate(model, glm::radians(-90.0f), glm::vec3(0.0f, 1.0f, 0.0f));

	model = glm::translate(model, glm::vec3(0.0f, 0.0f, distancia));

	glUniformMatrix4fv(glGetUniformLocation(shader.Program, "model"), 1, GL_FALSE, glm::value_ptr(model));

	colapez1.Draw(shader);


	model = glm::mat4(1);

	//model = glm::translate(model, glm::vec3(15.0f, 1.4f, 4.5f));

	//model = glm::rotate(model, glm::radians(-90.0f), glm::vec3(0.0f, 1.0f, 0.0f));

	model = glm::translate(model, glm::vec3(0.0f, 0.0f, distancia2));
```

```
	glUniformMatrix4fv(glGetUniformLocation(shader.Program, "model"), 1, GL_FALSE, glm::value_ptr(model));
	colapez2.Draw(shader);

	model = glm::mat4(1);
	model = glm::translate(model, glm::vec3(0.0f, 0.0f, distancia));
	//model = glm::rotate(model, glm::radians(rot), glm::vec3(1.0f, 0.0f, 0.0f));
	glUniformMatrix4fv(glGetUniformLocation(shader.Program, "model"), 1, GL_FALSE, glm::value_ptr(model));
	cpez1.Draw(shader);

	model = glm::mat4(1);
	model = glm::translate(model, glm::vec3(0.0f, 0.0f, distancia2));
	//model = glm::rotate(model, glm::radians(rot), glm::vec3(1.0f, 0.0f, 0.0f));
	glUniformMatrix4fv(glGetUniformLocation(shader.Program, "model"), 1, GL_FALSE, glm::value_ptr(model));
	cpez2.Draw(shader);

	model = glm::mat4(1);
	//model = glm::translate(model, glm::vec3(15.0f, 0.6f, -1.0f));
	//model = glm::rotate(model, glm::radians(90.0f), glm::vec3(0.0f, 1.0f, 0.0f));
	glUniformMatrix4fv(glGetUniformLocation(shader.Program, "model"), 1, GL_FALSE, glm::value_ptr(model));
	bocina.Draw(shader);

	model = glm::mat4(1);
	//model = glm::translate(model, glm::vec3(-5.0f, 0.0f, 0.0f));
	model = glm::translate(model, glm::vec3(aumento, aumento, 0.0f));
	//model = glm::rotate(model, glm::radians(90.0f), glm::vec3(0.0f, 1.0f, 0.0f));
	glUniformMatrix4fv(glGetUniformLocation(shader.Program, "model"), 1, GL_FALSE, glm::value_ptr(model));
	speakers.Draw(shader);
```

```
		model = glm::mat4(1);
		//model = glm::translate(model, glm::vec3(0.0f, -3.0f, 0.0f));
		//model = glm::rotate(model, glm::radians(-90.0f), glm::vec3(0.0f, 1.0f, 0.0f));
		glUniformMatrix4fv(glGetUniformLocation(shader.Program, "model"), 1, GL_FALSE, glm::value_ptr(model));
		torre.Draw(shader);

		model = glm::mat4(1);
		//model = glm::translate(model, glm::vec3(0.0f, -3.0f, 0.0f));
		//model = glm::rotate(model, glm::radians(-90.0f), glm::vec3(0.0f, 1.0f, 0.0f));
		glUniformMatrix4fv(glGetUniformLocation(shader.Program, "model"), 1, GL_FALSE, glm::value_ptr(model));
		ventanas.Draw(shader);

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

		model = glm::mat4(1);
		//model = glm::translate(model, glm::vec3(18.45f, 1.4f, 4.3f));
		//model = glm::rotate(model, glm::radians(-90.0f), glm::vec3(0.0f, 1.0f, 0.0f));
		glUniformMatrix4fv(glGetUniformLocation(lightingShader.Program, "model"), 1, GL_FALSE, glm::value_ptr(model));
		glUniform1f(glGetUniformLocation(lightingShader.Program, "activaTransparencia"), 0);
		glUniform4f(glGetUniformLocation(lightingShader.Program, "coloralpha"), 1.0f, 1.0f, 1.0f, 0.15f);
		vidrio.Draw(shader);
		glUniform4f(glGetUniformLocation(lightingShader.Program, "coloralpha"), 1.0f, 1.0f, 1.0f, 0.15f);

		glDisable(GL_BLEND);  //Desactiva el canal alfa
		glBindVertexArray(0);

		glActiveTexture(GL_TEXTURE0);
```

```
        glBindTexture(GL_TEXTURE_2D, texture);
        lampshader.Use();
        //glm::mat4 model(1);
        glUniformMatrix4fv(glGetUniformLocation(shader.Program, "projection"), 1, GL_FALSE, glm::value_ptr(projection));
        glUniformMatrix4fv(glGetUniformLocation(shader.Program, "view"), 1, GL_FALSE, glm::value_ptr(view));
        glUniformMatrix4fv(glGetUniformLocation(shader.Program, "model"), 1, GL_FALSE, glm::value_ptr(model));
        glBindVertexArray(VAO);

        //glDrawElements(GL_TRIANGLES, 6, GL_UNSIGNED_INT, 0);
        modelLoc = glGetUniformLocation(lampshader.Program, "model");
        viewLoc = glGetUniformLocation(lampshader.Program, "view");
        projLoc = glGetUniformLocation(lampshader.Program, "projection");

        // Set matrices
        glUniformMatrix4fv(viewLoc, 1, GL_FALSE, glm::value_ptr(view));
        glUniformMatrix4fv(projLoc, 1, GL_FALSE, glm::value_ptr(projection));
        model = glm::mat4(1);
        model = glm::translate(model, lightPos);
        model = glm::scale(model, glm::vec3(0.2f)); // Make it a smaller cube
        glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
        // Draw the light object (using light's vertex attributes)
        for (GLuint i = 0; i < 4; i++)
        {
            model = glm::mat4(1);
            model = glm::translate(model, pointLightPositions[i]);
            model = glm::scale(model, glm::vec3(0.1f)); // Make it a smaller cube
            glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
            glBindVertexArray(VAO);
            glDrawArrays(GL_TRIANGLES, 0, 36);
```

```
}
glBindVertexArray(0);

anim.Use();
tiempo = glfwGetTime();
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);
reflect.Draw(anim);

anim2.Use();
tiempo = glfwGetTime();
modelLoc = glGetUniformLocation(anim2.Program, "model");
viewLoc = glGetUniformLocation(anim2.Program, "view");
projLoc = glGetUniformLocation(anim2.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(anim2.Program, "time"), tiempo);
fondo.Draw(anim2);
```

```
        glBindVertexArray(0);


        // Swap the buffers
        glfwSwapBuffers(window);
    }

    glDeleteVertexArrays(1, &VAO);
    glDeleteBuffers(1, &VBO);
    glDeleteBuffers(1, &EBO);

    glfwTerminate();
    return 0;
    //<code>
}


// Moves/alters the camera positions based on user input
/*<summary> function that manage the camera positions and
the boolean variables to activate other animations </summary>*/
void DoMovement()
{
    // 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);
    }

    if (keys[GLFW_KEY_O])
    {
        circuito = true;
    }
    if (keys[GLFW_KEY_L])
    {
        circuito = false;
    }
    if (keys[GLFW_KEY_M])
    {
        bocinas = true;
    }
    if (keys[GLFW_KEY_N])
    {
        bocinas = false;
    }
}

/*<summary> function with an if conditional to increment or
```

```
decrement the values from float variables to translate the object
and appear is swimming in constant time </summary>*/
void nadar()
{

    //Movimiento de los peces
    if (circuito)
    {
        if (recorrido1)
        {
            distancia -= 0.01f;
            distancia2 += 0.01f;
            //rot = 180.0f;
            if (distancia <= -1.4)
            {
                recorrido1 = false;
                recorrido2 = true;
            }
        }
        if (recorrido2)
        {
            //rotKit = 90;
            distancia += 0.01f;
            distancia2 -= 0.01f;
            rot = 180.0f;
            if (distancia >= 0.0)
            {
                recorrido2 = false;
                recorrido3 = true;

            }
```

```
        }

        if (recorrido3)
        {
            //rotKit = 180;
            distancia -= 0.01f;
            distancia2 += 0.01f;
            rot = 0.0f;
            if (distancia <= -1.4)
            {
                recorrido3 = false;
                recorrido4 = true;
            }
        }

        if (recorrido4)
        {
            /*rotKit = 315;
            movKitX -= 0.1f;
            movKitZ += 0.1f;*/
            distancia += 0.01f;
            distancia2 -= 0.01f;
            rot = 180.0f;
            if (distancia >= 0.0)
            {
                recorrido4 = false;
                recorrido1 = true;
            }
        }
    }
}
```

```
/*<summary> function with an if conditional to increment or
decrement the values from float variables to translate the object
and appear is playing music through the speakers in constant time </summary>*/
void musica() {

    if (bocinas)
    {
        if (recorrido6)
        {
            aumento += 0.002f;
            if (aumento >= 0.02)
            {
                recorrido6 = false;
                recorrido7 = true;
            }
        }
        if (recorrido7)
        {
            //rotKit = 90;
            aumento -= 0.002f;
            if (aumento <= -0.0)
            {
                recorrido7 = false;
                recorrido6 = true;

            }
        }
    }
}
// Is called whenever a key is pressed/released via GLFW
```

```
/*<summary> function with if conditionals to check if a key was pressed
to activate the boolean variable and to change the value from
other variables. It depends from which key you pressed to activate the proper process.</summary>*/
void KeyCallback(GLFWwindow* window, int key, int scancode, int action, int mode)
{
    if (GLFW_KEY_ESCAPE == key && GLFW_PRESS == action)
    {
        glfwSetWindowShouldClose(window, GL_TRUE);
    }

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

    if (keys[GLFW_KEY_SPACE])
    {
        active = !active;
        if (active)
        {
            rotem = -0.1;
            rotna = 0.3;
            Light1 = glm::vec3(1.0f, 0.0f, 0.0f);
```

```
            //Light2 = glm::vec3(1.0f, 1.0f, 0.0f);
            //Light3 = glm::vec3(1.0f, 1.0f, 0.0f);
            //Light4 = glm::vec3(1.0f, 1.0f, 0.0f);
            //Light4 = glm::vec3(0.717, 0.929, 0.047);
        }
        else
        {
            rotem = 0.0f;
            rotna = 0.0f;
            Light1 = glm::vec3(0);//Cuado es solo un valor en los 3 vectores pueden dejar solo una componente
            /*Light2 = glm::vec3(0);
            Light3 = glm::vec3(0);
            Light4 = glm::vec3(0);*/
        }
    }
    if (keys[GLFW_KEY_P])
    {
        active2 = !active2;
        if (active2)
        {

            Light2 = glm::vec3(1.0f, 1.0f, 0.0f);

        }
        else
        {

            Light2 = glm::vec3(0);

        }
```

```
    }

}

/*<summary> function with an if conditional to initialize the camera sight
with the mouse, if you rotate the mouse, the sight will follow the mouse.</summary>
<param> window: where the mouse with the sight will take effect. </param>
<param> xPos: variable as reference to calculate the position from x. </param>
<param> yPos: variable as reference to calculate the position from y. </param>*/
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);
}
```

Conclusiones

Gracias a la implementación de este proyecto final pude llevarme demasiados conceptos en un solo proyecto y no solo eso sino darle una constante atención al desarrollo de un proyecto.

Personalmente pienso que este proyecto ha sido diferente a la mayoría que he tenido, pues no solamente desarrollé los conceptos vistos en teoría y laboratorio, sino también puede tener una pequeña perspectiva de lo que es un proyecto en el campo laboral. Pues este proyecto partió desde el principio del semestre y hemos tenido que dar reportes semanales y avances de este mismo. Adicionalmente tuvimos que crear una documentación completa como entregables.

Entonces esto me hace imaginar lo que uno como individuo puede hacer haciendo un proyecto como este, la exigencia que se necesitará al tener un proyecto más elaborado junto con un equipo de trabajo.

Thanks to this implementation, I could take many concepts in a single project and not only that, I had the chance to keep the daily attention to develop this project.

Personally, I think that this project has been different from most of it because I could take many concepts from theory and laboratory, nevertheless I could also have a little idea of what is about a real project into the job field. This project started at the beginning of the semester and we had to give weekly reports from the advantage and we had to create a complete documentation as deliveries.

In conclusion, this makes me imagine if one person could make this project, the required attention from a job project it’ll be higher with teamwork.