UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

DIVISIÓN DE INGENIERÍA ELÉCTRICA

INGENIERÍA EN COMPUTACIÓN

COMPUTACIÓN GRÁFICA e INTERACCIÓN HUMANO COMPUTADORA

FINAL PROJECT: TECHNICAL MANUAL

ACCOUNT NUMBER: 421039953

THEORY GROUP: 05

SEMESTER 2024-2

DEADLINE: 21/05/2024

GRADE: _____

Introduction

The final project developed in this C++ code focuses on creating a complex 3D scene using OpenGL. This project not only demonstrates the rendering capabilities of OpenGL but also integrates various advanced computer graphics techniques such as dynamic lighting, texture mapping, 3D model loading, and object animation.

Project Objectives

The main objective of this project is to create a realistic and animated 3D scene that includes multiple objects, textures, and lights, providing a visually appealing experience. To achieve this, the project has been divided into several key tasks:

Loading and Rendering 3D Models:

- Implementation of a system to load and render 3D models in common formats such as OBJ.
- Use of the stb_image library for texture loading.

Implementation of Dynamic Lighting:

- Creation of different types of lights, including directional lights, point lights, and spotlights, to add realism to the scene.
- Adjustment of lighting parameters to simulate effects like specularity and light intensity.

Object Animation:

- Programming animations for various objects in the scene, such as opening a door, moving a drawer, rotating a seat, and moving a poster.
- Controlling animations using time variables to ensure smooth and realistic movements.

User Interaction:

- Implementation of a user-controlled camera that allows exploring the scene from different angles and perspectives.
- Handling keyboard and mouse events to move the camera and activate animations.

General Description of the Code

The code is structured into several sections that work together to create the final scene. Below are the most important parts of the code:

- Dependencies and Libraries: All the necessary libraries for the program's operation, such as GLEW, GLFW, GLM, and stb_image, are included.
- Global Variables: Global variables are defined to manage the window, camera, textures, models, lights, and animations.
- Utility Functions:
    - calcAverageNormals: Calculates average normals for model vertices, improving lighting quality.

    - CreateObjects: Creates the objects to be rendered in the scene, such as the floor and other 3D models.
    - CreateShaders: Creates and compiles the shaders used for rendering the scene.

- Main Function (main): Initializes the window, creates objects and shaders, loads textures and models, and controls the render loop where animations are updated, and the scene is rendered in each frame.

Techniques and Concepts Used

OpenGL and Shaders:

- GLFW is used to create and manage the window and OpenGL contexts, as well as handle user input.
- GLEW provides a mechanism to query and load OpenGL extensions, allowing the use of more advanced API features.

GLM:

- GLM (OpenGL Mathematics) is a mathematical library that facilitates vector and matrix operations needed for 3D transformations and projections.

Lighting and Materials:

- Different types of lights (directional, point, and spotlight) are implemented to create realistic lighting in the scene.

- Materials are defined with specific specularity and shininess properties, simulating different types of surfaces.

Texture and Model Loading:

- Textures are used to add realism to the surfaces of models. The stb_image library is used to load textures from image files.

- 3D models are loaded from OBJ files, allowing the inclusion of complex objects in the scene.

Animation:

- Animations are controlled using time variables, allowing smooth and natural movements of objects in the scene.

- Various animations are programmed, such as opening doors and moving furniture, to bring the scene to life.

Code Description, Gantt Chart, and Cost Analysis

Models, Textures, and Skybox

First, objects, textures, and the skybox are declared:

```
//Texturas a utilizar en entorno opengl
Texture pisoTexture;    //Textura de piso

//Modelos a utilizar en entorno opengl

//Edificaciones
Model Casa;

//Objetos
//Planta Baja
Model Taylor;
Model Cama;
Model Jarron;
Model Buro;
Model BuroPuerta;
Model BuroCajon;
Model Silla;
Model AsientoSilla;

//Planta Alta
Model SillDobl;
Model LamPared;
Model Mesa;
Model AsBike;
Model AsBikeSup;
Model SillInd;

//Skybox a utilizar en entorno opengl
Skybox skybox;
```

Screenshot 1. Declaration of elements.

Continuously, primitives are used to create the floor.

```
//Para el piso
void CreateObjects()
{
    unsigned int indices[] = {
        0, 3, 1,
        1, 3, 2,
        2, 3, 0,
        0, 1, 2
    };

    GLfloat vertices[] = {
        //  x      y      z         u     v         nx    ny    nz
            -1.0f, -1.0f, -0.6f,    0.0f, 0.0f,     0.0f, 0.0f, 0.0f,
            0.0f, -1.0f, 1.0f,      0.5f, 0.0f,     0.0f, 0.0f, 0.0f,
            1.0f, -1.0f, -0.6f,     1.0f, 0.0f,     0.0f, 0.0f, 0.0f,
            0.0f, 1.0f, 0.0f,       0.5f, 1.0f,     0.0f, 0.0f, 0.0f
    };

    unsigned int floorIndices[] = {
        0, 2, 1,
        1, 2, 3
    };

    GLfloat floorVertices[] = {
        -10.0f, 0.0f, -10.0f,   0.0f, 1.0f, 0.0f, -1.0f, 0.0f,  //0
        10.0f, 0.0f, -10.0f,    1.0f, 1.0f, 0.0f, -1.0f, 0.0f,  //1
        -10.0f, 0.0f, 10.0f,    0.0f, 0.0f, 0.0f, -1.0f, 0.0f,  //2
        10.0f, 0.0f, 10.0f,     1.0f, 0.0f, 0.0f, -1.0f, 0.0f   //3
    };
```

```
    Mesh *obj1 = new Mesh();
    obj1->CreateMesh(vertices, indices, 32, 12);
    meshList.push_back(obj1);

    Mesh *obj2 = new Mesh();
    obj2->CreateMesh(vertices, indices, 32, 12);
    meshList.push_back(obj2);

    Mesh *obj3 = new Mesh();
    obj3->CreateMesh(floorVertices, floorIndices, 32, 6);
    meshList.push_back(obj3);

    calcAverageNormals(indices, 12, vertices, 32, 8, 5);
}
```

Screenshot 2. Floor using primitives.

In the main function, the floor function is called, and textures, models, and the skybox are loaded.

```
int main()
{
    mainWindow = Window(1366, 768); // 1280, 1024 or 1024, 768
    mainWindow.Initialise();

    CreateObjects();
```

Screenshot 3. Floor loading.

```
208     //*******************************CARGA DE TEXTURAS*************************************
209     pisoTexture = Texture("Textures/piso.tga");
210     pisoTexture.LoadTexture();
211
212     //*******************************CARGA DE MODELOS**************************************
213     //Objetos
214     //Planta Alta
215     Taylor = Model();
216     Taylor.LoadModel("Models/Taylor.obj");
217
218     Cama = Model();
219     Cama.LoadModel("Models/Cama.obj");
220
221     Jarron = Model();
222     Jarron.LoadModel("Models/Jarron.obj");
223
224     Buro = Model();
225     Buro.LoadModel("Models/Buro.obj");
226
227     BuroPuerta = Model();
228     BuroPuerta.LoadModel("Models/BuroPuerta.obj");
229
230     BuroCajon = Model();
231     BuroCajon.LoadModel("Models/BuroCajon.obj");
232
233     Silla = Model();
234     Silla.LoadModel("Models/Silla.obj");
235
236     AsientoSilla = Model();
237     AsientoSilla.LoadModel("Models/AsientoSilla.obj");
238
239     //Planta Baja
240     SillDobl = Model();
241     SillDobl.LoadModel("Models/SillDobl.obj");
242
243     LamPared = Model();
244     LamPared.LoadModel("Models/LamPared.obj");
245
246     Mesa = Model();
247     Mesa.LoadModel("Models/Mesa.obj");
248
249     AsBikeSup = Model();
250     AsBikeSup.LoadModel("Models/AsBikeSup.obj");
```

Screenshot 4. Floor texture and model loading.

```
252     AsBike = Model();
253     AsBike.LoadModel("Models/AsBike.obj");
254
255     SillInd = Model();
256     SillInd.LoadModel("Models/SillInd.obj");
257
258     //Cornelias Casa
259     Casa = Model();
260     Casa.LoadModel("Models/Casa.obj");
261
262     std::vector<std::string> skyboxFaces;
263
264     skyboxFaces.push_back("Textures/Skybox/skybox_3.tga"); //right
265     skyboxFaces.push_back("Textures/Skybox/skybox_1.tga"); //left
266     skyboxFaces.push_back("Textures/Skybox/skybox_6.tga"); //down
267     skyboxFaces.push_back("Textures/Skybox/skybox_5.tga"); //up
268     skyboxFaces.push_back("Textures/Skybox/skybox_2.tga"); //front
269     skyboxFaces.push_back("Textures/Skybox/skybox_4.tga"); //bh
270
271     skybox = Skybox(skyboxFaces);
```

Screenshot 5. Model and skybox loading.

Now, in the while loop, models are added and arranged on the map.

```
//*******************************************************************PISO****************
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.0f, -1.0f, 0.0f));
model = glm::scale(model, glm::vec3(30.0f, 1.0f, 30.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniform3fv(uniformColor, 1, glm::value_ptr(color));

pisoTexture.UseTexture();
Material_opaco.UseMaterial(uniformSpecularIntensity, uniformShininess);

meshList[2]->RenderMesh();

//*******************************************************************EDIFICACIONES********

//Luz de casa
mainLight = DirectionalLight(1.0f, 1.0f, 1.0f,
    0.5f, 0.05f,
    0.0f, 0.0f, -1.0f);

shaderList[0].SetDirectionalLight(&mainLight);

//Casa
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.0f, -0.5f, 0.0f));
model = glm::scale(model, glm::vec3(5.0f, 5.0f, 5.0f));

glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
Casa.RenderModel();
```

Screenshot 6. Floor and model implementation.

```
//PLANTA ALTA
//Poster Taylor (1)
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(47.8f, 150.0f, 0.0f));
model = glm::scale(model, glm::vec3(20.0f, 20.0f, 20.0f));
model = glm::rotate(model, -90 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::rotate(model, 10 * sin(glm::radians(movPoster)) * toRadians, glm::vec3(0.0f, 0.0f, 1.0f));

glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
Taylor.RenderModel();

//Cama (2)
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.0f, 112.0f, -59.0f));
model = glm::scale(model, glm::vec3(7.0f, 7.0f, 7.0f));

glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
Cama.RenderModel();

//Jarron (3)
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(-15.0f, 134.5f, 91.0f));
model = glm::scale(model, glm::vec3(3.0f, 3.0f, 3.0f));
```

Screenshot 7. Upper floor model implementation.

```
//Buro (4) Jerarquia
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.0f, 115.0f, 91.0f));
model = glm::scale(model, glm::vec3(30.0f, 30.0f, 30.0f));
model = glm::rotate(model, 180 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));

modelaux = model;
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
Buro.RenderModel();

//Buro Puerta (4) Jerarquia
model = glm::translate(model, glm::vec3(0.75f, 0.1f, 0.2f));
model = glm::rotate(model, movPuerta * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));

glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
BuroPuerta.RenderModel();

//Buro Cajon (4) Jerarquia
model = modelaux;
model = glm::translate(model, glm::vec3(-0.4f, 0.4f , 0.11f + movCajon));

glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
BuroCajon.RenderModel();

//Silla (5) Jerarquia, en caso de que se quiera girar el asiento pero ya hay uno que lo hace
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(40.0f, 119.1f, 91.0f));
model = glm::scale(model, glm::vec3(30.0f, 30.0f, 30.0f));
model = glm::rotate(model, -135 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));

glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
Silla.RenderModel();

//Silla (5) Asiento
model = glm::translate(model, glm::vec3(0.0f, 0.0f, -0.0f));

glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
AsientoSilla.RenderModel();
```

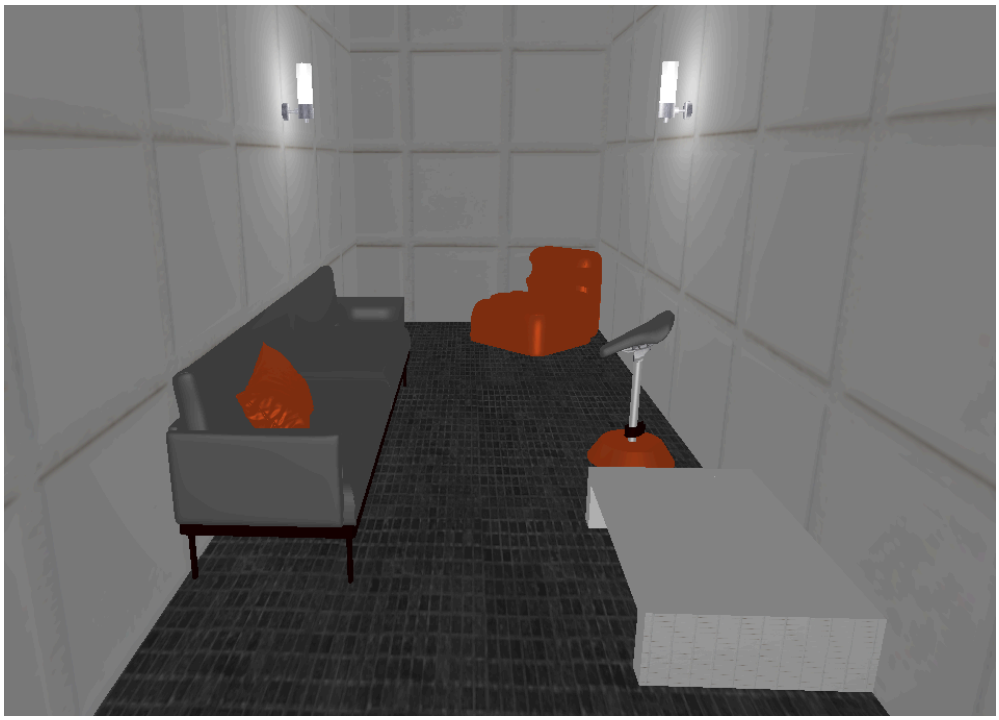
Screenshot 8. Upper floor model implementation.

```
//PLANTA BAJA
//Sillon Doble (6)
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(-23.0f, 10.35f, -5.0f));
model = glm::scale(model, glm::vec3(20.0f, 30.0f, 30.0f));

glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
SillDobl.RenderModel();

//Lampara Pared 1 (7)
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(-30.6f, 55.0f, -25.0f));
model = glm::scale(model, glm::vec3(30.0f, 30.0f, 30.0f));

glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
LamPared.RenderModel();

//Lampara Pared 2 (7)
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(30.6f, 55.0f, -25.0f));
model = glm::scale(model, glm::vec3(30.0f, 30.0f, 30.0f));
model = glm::rotate(model, 180 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));

glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
LamPared.RenderModel();

//Mesa de centro (8)
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(23.0f, -0.5f, 30.0f));
model = glm::scale(model, glm::vec3(10.0f, 10.0f, 15.0f));
model = glm::rotate(model, 90 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));

glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
Mesa.RenderModel();

//Base Bicicleta (9) Jerarquia
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(23.0f, 1.8f, -5.0f));
model = glm::scale(model, glm::vec3(7.0f, 10.0f, 7.0f));
model = glm::rotate(model, 180 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));

glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
AsBike.RenderModel();

//Asiento Silla Bicicleta (9) Jerarquia
model = glm::translate(model, glm::vec3(-0.0f, 0.0f, -0.0f));
model = glm::rotate(model, movBici * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));

glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
AsBikeSup.RenderModel();

//Sillon Individual (10)
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(15.0f, 11.0f, -80.0f));
model = glm::scale(model, glm::vec3(1.5f, 1.5f, 1.5f));
model = glm::rotate(model, -135 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));

glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
SillInd.RenderModel();
```

Screenshot 9. Lower floor model implementation.

Running the program, you can see how everything was generated correctly, models, skybox, and the floor with its texture.

Screenshot 10. House model result, skybox, and floor.



Screenshot 11. Lower floor objects result.

Screenshot 12. Upper floor objects result.

Walkthrough

To explore the map, a static camera is used, so it is declared.

```
//Camara
Camera camera;
```

Screenshot 13. Static camera declaration.

Now, in the main function, camera values are set, where the first arguments are its position, followed by its direction and viewing angle.

```
//Camara estatica ubicacion
camera = Camera(glm::vec3(0.0f, 20.0f, 70.0f), glm::vec3(0.0f, 1.0f, 0.0f), -90.0f, 0.0f, 0.3f, 0.5f);
```

Screenshot 14. Camera values.

Inside the while loop, keyboard and mouse movements are added to the camera, also passing the skybox to be observed.

```
//Controles de camara
camera.keyControl(mainWindow.getsKeys(), deltaTime);
camera.mouseControl(mainWindow.getXChange(), mainWindow.getYChange());

// Clear the window
glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

//Para que la camara vea el skybox
skybox.DrawSkybox(camera.calculateViewMatrix(), projection);
```

Screenshot 15. Camera movement and view.

Now, in Camera.cpp, keys are configured, and the speed is set to 5 units per movement and the vision when moving the mouse.

```
//Controles Camara
void Camera::keyControl(bool* keys, GLfloat delt
{
    GLfloat velocity = moveSpeed * deltaTime;

    if (keys[GLFW_KEY_W])
    {
        position += front * velocity;
    }

    if (keys[GLFW_KEY_S])
    {
        position -= front * velocity;
    }

    if (keys[GLFW_KEY_A])
    {
        position -= right * velocity;
    }

    if (keys[GLFW_KEY_D])
    {
        position += right * velocity;
```

```
void Camera::mouseControl(GLfloat xChange, GLfloat yChange)
{
    xChange *= turnSpeed;
    yChange *= turnSpeed;

    yaw += xChange;
    pitch += yChange;

    if (pitch > 89.0f)
    {
        pitch = 89.0f;
    }

    if (pitch < -89.0f)
    {
        pitch = -89.0f;
    }

    update();
}
```

Screenshot 16. Keys to move the camera and its speed, along with mouse control.

Finally, the shader matrix for camera position and orientation is updated for lighting calculations.

```
glUniformMatrix4fv(uniformView, 1, GL_FALSE, glm::value_ptr(camera.calculateViewMatrix()));
glUniform3f(uniformEyePosition, camera.getCameraPosition().x, camera.getCameraPosition().y, camera.getCameraPosition().z);
```

Screenshot 17. Shader information to render from the camera's perspective.

Lighting

Point lights and spotlights are declared.

```
//*********************************LUZ DE LAMPARA IZQUIERDA*********
pointLights[0] = PointLight(1.0f, 1.0f, 1.0f,
    0.4f, 0.1f,
    -28.6f, 55.0f, -25.0f,
    0.75f, 0.005f, 0.01f);
pointLightCount++;
//*********************************LUZ DE LAMPARA DERECHA***********
pointLights[1] = PointLight(1.0f, 1.0f, 1.0f,
    0.4f, 0.1f,
    28.6f, 55.0f, -25.0f,
    0.75f, 0.005f, 0.01f);
pointLightCount++;

//LUCES SPOTLIGHT
//Contador de luces spotlight
unsigned int spotLightCount = 0;

//Primera luz Spotlight
//*********************************LUCES DEL POSTER*****************
spotLights[0] = SpotLight(1.0f, 0.0f, 0.0f,
    1.0f, 2.0f,
    5.0f, 150.0f, 0.0f,
    1.0f, 0.0f, 0.0f,
    1.0f, 0.003f, 0.0035f,
    30.0f);
spotLightCount++;

spotLights[1] = SpotLight(0.0f, 1.0f, 0.0f,
    1.0f, 2.0f,
    5.0f, 150.0f, 0.0f,
    1.0f, 0.0f, 0.0f,
    1.0f, 0.003f, 0.0035f,
    30.0f);
spotLightCount++;

spotLights[2] = SpotLight(0.0f, 0.0f, 1.0f,
    1.0f, 2.0f,
    5.0f, 150.0f, 0.0f,
    1.0f, 0.0f, 0.0f,
    1.0f, 0.003f, 0.0035f,
    30.0f);
spotLightCount++;
```

Screenshot 18. Point and spotlights declaration.

Directional light is declared, and information is passed to the shader, noting that the directional light is declared at this point to act differently if inside or outside the house.

```
//*********************************************************

//Luz Exterior
mainLight = DirectionalLight(1.0f, 1.0f, 1.0f,
    0.5f, 0.5f,
    0.0f, 0.0f, -1.0f);

//información al shader de fuentes de iluminación
shaderList[0].SetDirectionalLight(&mainLight);
shaderList[0].SetPointLights(pointLights, pointLightCount);
```

Screenshot 19. Main light declaration and shader information passing.

```
//*************************************************

//Luz de casa
mainLight = DirectionalLight(1.0f, 1.0f, 1.0f,
    0.5f, 0.05f,
    0.0f, 0.0f, -1.0f);

shaderList[0].SetDirectionalLight(&mainLight);
```

Screenshot 20. Main light declaration on the house and update.

```
//Luz de cuartos (Objetos)
mainLight = DirectionalLight(1.0f, 1.0f, 1.0f,
    0.5f, 0.005f,
    0.0f, 0.0f, -1.0f);

shaderList[0].SetDirectionalLight(&mainLight);
```

Screenshot 21. Main light declaration on objects and update.

Spotlights will be discussed in the animations section as they will depend on the deltaTime.

Animations

First, modify Window.h and Window.cpp to create functions for using keys to activate or deactivate animations.

```
//Funciones para las teclas
GLfloat getsilla() { return silla; }

GLfloat getpuerta() { return puerta; }

GLfloat getcajon() { return cajon; }

GLfloat getposter() { return poster; }
```

```
//Para las teclas
GLfloat silla;

GLfloat puerta;

GLfloat cajon;

GLfloat poster;
```

Screenshot 22. Functions to get values of variables declared in Window.h.

In Window.cpp, variables are assigned a value of 1.0f, which will serve as a flag for implementing animations in PR.cpp.

```
silla = 1.0f;
puerta = 1.0f;
cajon = 1.0f;
poster = 1.0f;
```

Screenshot 23. Variable values declared in Window.cpp.

Continuing with Window.cpp, actions are added for when keys Z, X, C, and V are pressed, changing their value from 1.0f to -1.0f each time they are pressed.

```
if (key == GLFW_KEY_Z && action == GLFW_PRESS)
{
    theWindow->silla = -1.0f * theWindow->silla;
}

if (key == GLFW_KEY_X && action == GLFW_PRESS)
{
    theWindow->puerta = -1.0f * theWindow->puerta;
}

if (key == GLFW_KEY_C && action == GLFW_PRESS)
{
    theWindow->cajon = -1.0f * theWindow->cajon;
}

if (key == GLFW_KEY_V && action == GLFW_PRESS)
{
    theWindow->poster = -1.0f * theWindow->poster;
}
```

Screenshot 24. Behavior of declared variables and assigned keys in Window.cpp.

In PR.cpp, global variables for object and spot light animations are declared. There is the movement applied to models, the offset for modifying movement speed, and the direction of the object, which changes depending on the boolean dir variables. The poster light has a spotlight where the light range determines color change.

```
//Variables animacion

//Animacion Silla Bici
float movBici = 0.0f;                   //Movimiento en el eje y de asiento
float movOffsetBici;                    //Velocidad

float movPuerta = 0.0f;                 //Movimiento puerta
float movOffsetPuerta;                  //Velocidad
bool dirPuerta = true;                  //direccion Puerta

float movCajon = 0.0f;                  //Movimiento cajon
float movOffsetCajon;                   //Velocidad
bool dirCajon = true;                   //direccion cajon

float movPoster = 0.0f;                 //Movimiento Poster

float lucesPoster = 0.0f;               //Luces Poster
float movOffsetLuces;                   //Velocidad
bool rangoLuces = true;                 //Rangos
```

Screenshot 25. Variable declaration.

```
//Variables animacion Velocidad
movOffsetBici = 1.5f;
movOffsetPuerta = 1.5f;
movOffsetCajon = 0.01f;
movOffsetLuces = 0.1f;
```

Screenshot 26. Offset modification.

In the while loop, values for deltaTime are set, followed by the first animation that increases depending on the deltaTime variable. This animation is applied to the AsBikeSup model, corresponding to the bike seat.

```
GLfloat now = glfwGetTime();
deltaTime = now - lastTime;
deltaTime += (now - lastTime) / limitFPS;
lastTime = now;

//Animaciones
//Bici (1)
if (mainWindow.getsilla() == -1.0f)
{
    movBici += movOffsetBici * deltaTime;
}
```

Screenshot 27. Bike seat animation.

```
//Asiento Silla Bicicleta (9) Jerarquia
model = glm::translate(model, glm::vec3(-0.0f, 0.0f, -0.0f));
model = glm::rotate(model, movBici * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));

glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
AsBikeSup.RenderModel();
```

Screenshot 28. Implementation of movBike in model rotation on the Y-axis.

For door and drawer animations, a similar algorithm is used, taking the current value when the key is pressed, offset values, and movement ranges delimited by internal ifs supported by flags to change movement direction. Additionally, the drawer's movement is translational, and the door's is rotational, as seen in the following code blocks.

```
//Puerta (2)
if (mainWindow.getpuerta() == -1.0f)
{
    if (dirPuerta == true)
    {
        movPuerta -= movOffsetPuerta * deltaTime;
        if (movPuerta <= -0.0f)
        {
            dirPuerta = false;
        }
    }
    else if (dirPuerta == false)
    {
        movPuerta += movOffsetPuerta * deltaTime;
        if (movPuerta >= 90.0f)
        {
            dirPuerta = true;
        }
    }
}
```

```
//Cajon (3)
if (mainWindow.getcajon() == -1.0f)
{
    if (dirCajon == true)
    {
        movCajon -= movOffsetCajon * deltaTime;
        if (movCajon <= -0.0f)
        {
            dirCajon = false;
        }
    }
    else if (dirCajon == false)
    {
        movCajon += movOffsetCajon * deltaTime;
        if (movCajon >= 0.3f)
        {
            dirCajon = true;
        }
    }
}
```

Screenshot 28. Door and drawer animation movDoor and movDrawer.

```
//Buro Puerta (4) Jerarquia
model = glm::translate(model, glm::vec3(0.75f, 0.1f, 0.2f));
model = glm::rotate(model, movPuerta * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));

glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
BuroPuerta.RenderModel();
```

Screenshot 29. Implementation of movDoor for object rotation on the Y-axis.

```
//Buro Cajon (4) Jerarquia
model = modelaux;
model = glm::translate(model, glm::vec3(-0.4f, 0.4f , 0.11f + movCajon));

glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
BuroCajon.RenderModel();
```

Screenshot 30. Implementation of movDrawer for object translation on the Z-axis.

For poster animations, rotation increases as a function of deltaTime. Additionally, the spot light changes depending on the value of lightPoster supported by the offset for change speed multiplied by deltaTime, within an external if with the rangeLight flag changing based on lightPoster value. The ranges are adjusted from -5 to 15, explained further in the implementation section.

```
//Poster para ajustar cuadro (4)
if (mainWindow.getposter() ==-1.0f)
{
    movPoster += 2.5f * deltaTime;
}

//Poster Luces (5)
if (rangoLuces == true)
{
    lucesPoster += movOffsetLuces * deltaTime;
    if (lucesPoster >= 15.0)
    {
        rangoLuces = false;
    }
}
else if (rangoLuces == false)
{
    lucesPoster -= movOffsetLuces * deltaTime;
    if (lucesPoster <= -5.0)
    {
        rangoLuces = true;
    }
}
```

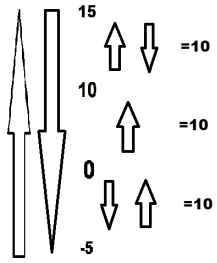
Screenshot 31. Poster animation movPoster and lightPoster.

For object rotation, movPoster is applied on the Z-axis using a sine function for constant movement multiplied by 10 to increase the rotation.

```
//PLANTA ALTA
//Poster Taylor (1)
model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(47.8f, 150.0f, 0.0f));
model = glm::scale(model, glm::vec3(20.0f, 20.0f, 20.0f));
model = glm::rotate(model, -90 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::rotate(model, 10 * sin(glm::radians(movPoster)) * toRadians, glm::vec3(0.0f, 0.0f, 1.0f));

glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
Taylor.RenderModel();
```

Screenshot 32. Implementation of lightPoster rotation on the Z-axis.

Finally, for spotlights, the following conditions are created to switch between selected lights (declared in the lighting section). Given that the ranges are from -5 to 15, each if has a range of 10 (because lightPoster must either decrease or increase the value when it reaches the upper or lower limit, as shown below).

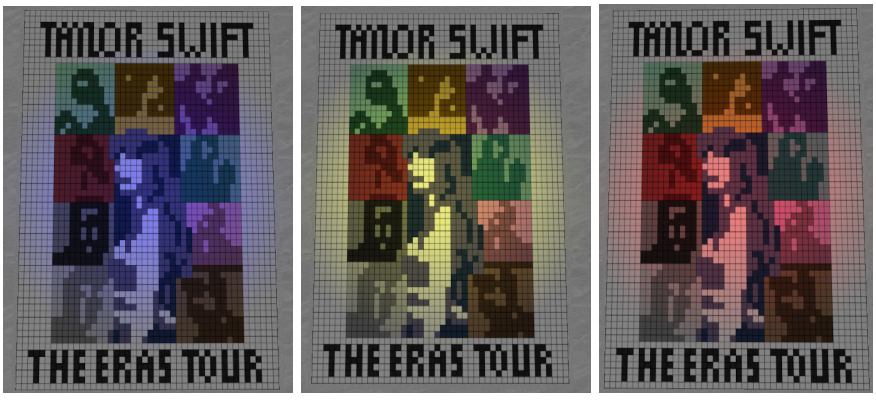


Screenshot 33. Operation of conditionals with established ranges.

```
//Dependiendo valor cambia el color de la luz spot
if (lucesPoster > 10.0f)
{
    shaderList[0].SetSpotLights(spotLights, spotLightCount - 2);//Rojo

}
else if (lucesPoster <= 10.0f && lucesPoster > 0.0f)
{
    shaderList[0].SetSpotLights(spotLights, spotLightCount - 1);//Verde

}
else if (lucesPoster < 0.0f)
{
    shaderList[0].SetSpotLights(spotLights + 2, spotLightCount - 1);//Azul

}
```

Screenshot 34. Conditions with established ranges and spotlight assignment.

Lights iterate as follows.



Screenshot 35. Color change depending on the spotlight used in the shader.

Gantt Chart

The following Gantt chart represents the detailed planning and tracking of the progress made in developing a project during the period from May 17 to May 20, 2024. The project focuses on creating and adapting a virtual environment, including organizing documents on digital platforms and 3D modeling of structures and objects. Throughout this document, the activities carried out, the time spent on each, and the milestones reached for effective project advancement are detailed.

The chart is structured around the key phases of the project, from the initial organization of digital resources to the implementation of the visual and technical elements necessary for its completion. Each activity is described precisely, reflecting the effort and dedication employed to achieve the goals set in the schedule.

This record not only documents the tasks performed but also serves as a tool to evaluate process efficiency and time management, providing a comprehensive view of the project's overall development.

ACTIVIDADES	MAYO			
	17/05/2024	18/05/2024	19/05/2024	20/05/2024
Crear carpeta, documentos en Drive y GitHub	30 minutos			
Adecuar proyecto	40 minutos			
Piso textura, modelado de casa, cuartos y objetos de planta alta		5 horas 20 minutos		
Objetos de planta baja modelado y texturizado, acomodado de todos los objetos y 2 animaciones			5 horas	
Luces, 2 animaciones de póster y luces spot			3 horas	
Documentación				3 horas

Screenshot 36. Gantt Chart.

Logbook

17/05/24

- Purpose: Create folder, documents on Drive, and GitHub.
- Time Spent: 30 minutes.

17/05/24

- Purpose: Set up a clean base project.
- Time Spent: 40 minutes.

18/05/24 - 19/05/24

- Purpose: Texture floor, house modeling, rooms, and upper floor objects.
- Time Spent: 5 hours 20 minutes.

19/05/24

- Purpose: Lower floor object modeling and texturing, arranging all objects and 2 animations.
- Time Spent: 5 hours

19/05/24

- Purpose: Lights, 2 poster animations, and spotlights.
- Time Spent: 3 hours

20/05/24

- Purpose: Gantt, Technical manual, user manual, and executable files.
- Time Spent: 3 hours

Cost Analysis

Variable Costs:

- Video game programmer hourly rate: $19,345 MXN per month (Approximate).
- Hours dedicated to the project: 25 hours.

Fixed Monthly Costs:

- Windows 10 license: $3,609 MXN.
- 3D Max license (for three years): $5,100 MXN.
- GIMP: $200 MXN.
- Daily meals (6 days at $100 pesos per day): $600 MXN.
- Electricity (every two months, worked 6 days): $200 MXN (proportional to one month).

Sales Cost:

- Additional percentage to operating cost: 33%.
- Taxes 16% VAT

Cost and Sale Price Calculation:

Variable Costs:

- Hourly rate: $19,345 MXN / 160 hours (160 worked per month) = $120.91 MXN per hour.
- Hours dedicated: 25 hours.
- Total variable cost: $120.91 MXN/hour * 25 hours = $3,022.75 MXN.

Fixed Costs:

- Windows 10 license: $3,609 MXN.
- 3D Max license (annualized to one month): $5,100 MXN
- GIMP: $200 MXN.
- Daily meals: $600 MXN.
- Electricity: $200 MXN.

Total monthly fixed costs = $3,609 MXN + $5,100 MXN + $200 MXN + $600 MXN + $200 MXN = $9,709 MXN.

Total Operating Cost:

- Variable costs + monthly fixed costs = $3,022.75 MXN + $9,709 MXN = $12,731.75 MXN.

Sale Price:

- Total operating cost * (1 + additional percentage for profit) = $12,731.75 MXN * (1 + 0.33) = $16,933.22 MXN.

Application of Taxes:

- Final price with 16% VAT = $16,933.22 MXN * 1.16 = **$19,642.54 MXN.**

Conclusión

This project taught me the importance of meticulous planning and self-management in software development. Throughout the development of this project, various computer graphics techniques using OpenGL were applied, allowing the creation of a complex and visually appealing 3D scene. Implementing dynamic lighting, loading 3D models, and animating objects were key aspects to achieving a good experience.

The process began with a clear definition of objectives, where the goal of creating a realistic and animated 3D scene was established. To achieve this, the project was divided into manageable tasks, each with its own set of problems and learnings. This modular approach allowed for more organized development and reduced errors.

Implementing dynamic lighting was one of the most challenging aspects of the project. Creating different types of lights, including directional, point, and spotlights, added significant realism to the scene. This aspect of the project not only improved my skills but also deepened my understanding of how light interacts with objects in a three-dimensional environment.

Programming animations for objects in the scene was another fundamental component. Controlling animations using time variables to ensure smooth and realistic movements demanded precision and attention to detail. Implementing a user-controlled camera that allows exploring the scene from different angles and perspectives further enhanced the interactivity and visual appeal of the project.

Finally, effective time and resource management were essential to the project's success. Using tools like the Gantt chart and conducting a detailed cost analysis allowed for precise planning and continuous progress tracking. These tools not only helped keep the project within established deadlines but also provided a clear view of process efficiency and time management. In summary, this project was a very enriching experience that helped me improve both my technical skills and organizational and self-management abilities, better preparing me for future projects in the field of computer engineering.

Credits

Models

- Cama
    - File name: 3D Bed Free Obj Model Free 3D model
    - Source: cgtrader
    - Author: 3dmodelbase
    - License Type: Royalty Free No Ai License
    - Link: 3D Bed Free Obj Model free 3D model | CGTrader
- Jarrón
    - File name: Vase 01
    - Source: cgtrader
    - Author: hend-z
    - License Type: Royalty Free No Ai License
    - Link: Vase 01 free 3D model | CGTrader
- Buro

    - File name: Fermo Double Dresser
    - Source: cgtrader
    - Author: nmcygni
    - License Type: Royalty Free No Ai License
    - Link: Fermo Double Dresser free VR / AR / low-poly 3D model | CGTrader
- Silla
    - File name: Ring Stool
    - Source: cgtrader
    - Author: litat
    - License Type: Royalty Free No Ai License
    - Link: Ring Stool free 3D model | CGTrader
- Sillon Doble
    - File name: Furniture Applaryd IKEA Sofa Couch
    - Source: cgtrader
    - Author: timur560
    - License Type: Royalty Free No Ai License
    - Link: Furniture Applaryd IKEA Sofa Couch free 3D model | CGTrader
- Sillon Individual
    - File name: Sofa modular
    - Source: cgtrader
    - Author: kirmisas
    - License Type: Royalty Free No Ai License
    - Link: Sofa modular free 3D model | CGTrader
- Lámpara
    - File name: Candelabro de pared modelo 3d
    - Source: turbosquid
    - Author: Piotr Zatarski
    - License Type: 3D Model License

    - Link: modelo 3d Candelabro de pared gratis - TurboSquid 1539085
- Asiento Bicicleta
    - File name: asiento de bicicleta modelo 3d
    - Source: turbosquid
    - Author: d.w.designs
    - License Type: 3D Model License
    - Link: modelo 3d asiento de bicicleta gratis - TurboSquid 729638

Textures

- Taylor.tga
    - File name: Alpha pattern #148909
    - Source: braceletbook
    - Author: ketchupfly
    - License Type: Royalty Free No Ai License
    - Link: Alpha pattern #148909 | BraceletBook
- Piso.tga
    - Source: Freepik.
    - Author: freepik.
    - License Type: Free for personal and commercial use with attribution.
    - Link: Cerrar textura de asfalto | Foto Gratis (freepik.es)
- casa.tga
    - Source: admagazine
    - Author: Katie Schultz
    - License Type: Free for personal and commercial use with attribution.
    - Link: Taylor Swift: La casa de “Cornelia Street” está en renta | Architectural Digest (admagazine.com)
- GravaTs.tga

    - Source: Freepik.
    - Author: freepik
    - License Type: Free for personal and commercial use with attribution.
    - Link: Fondo de textura granulada | Foto Gratis (freepik.es)
- pisoPB.tga
    - Source: Freepik.
    - Author: kues1
    - License Type: Free for personal and commercial use with attribution.
    - Link: Textura de la pared del embutido negro | Foto Gratis (freepik.es)
- azulejoBlanco.tga
    - Source: Freepik.
    - Author: lifeforstock
    - License Type: Free for personal and commercial use with attribution.
    - Link: Pared de azulejos blancos | Foto Gratis (freepik.es)
- metalico.tga
    - Source: Freepik.
    - Author: rawpixel.com
    - License Type: Free for personal and commercial use with attribution.
    - Link: Fondo plateado metalizado texturado. | Foto Gratis (freepik.es)