



UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

DIVISIÓN DE INGENIERÍA ELÉCTRICA

INGENIERÍA EN COMPUTACIÓN

COMPUTACIÓN GRÁFICA e INTERACCIÓN HUMANO
COMPUTADORA



PROYECTO FINAL: MANUAL TÉCNICO

NÚMERO DE CUENTA: 421039953

GRUPO DE TEORÍA: 05

SEMESTRE 2024-2

FECHA DE ENTREGA LÍMITE: 21/05/2024

CALIFICACIÓN: _____

Introducción.....	1
Objetivos del Proyecto.....	1
Descripción General del Código.....	2
Técnicas y Conceptos Utilizados.....	3
Descripción del código, Gantt y análisis de costos.....	5
Modelos, texturas y skybox.....	5
Recorrido.....	11
Iluminación.....	13
Animaciones.....	15
Diagrama de Gantt.....	21
Bitácora.....	22
Análisis de costos.....	23
Conclusión.....	24
Créditos.....	25
Modelos.....	25
Texturas.....	27

Introducción

El proyecto final desarrollado en este código en C++ se centra en la creación de una escena 3D compleja utilizando OpenGL. Este proyecto no solo demuestra las capacidades de renderizado de OpenGL, sino que también integra diversas técnicas avanzadas de gráficos por computadora, como la iluminación dinámica, el mapeo de texturas, la carga de modelos 3D y la animación de objetos.

Objetivos del Proyecto

El objetivo principal de este proyecto es crear una escena 3D realista y animada que incluya múltiples objetos, texturas y luces, proporcionando una experiencia visualmente atractiva. Para lograr esto, el proyecto se ha dividido en varias tareas clave:

Carga y Renderizado de Modelos 3D:

- Implementación de un sistema para cargar y renderizar modelos 3D en formatos comunes como OBJ.
- Uso de la librería stb_image para la carga de texturas.

Implementación de Iluminación Dinámica:

- Creación de diferentes tipos de luces, incluyendo luces direccionales, puntuales y spotlights, para añadir realismo a la escena.
- Ajuste de los parámetros de iluminación para simular efectos como la especularidad y la intensidad de la luz.

Animación de Objetos:

- Programación de animaciones para varios objetos en la escena, tales como la apertura de una puerta, el movimiento del cajón, la rotación del asiento y el movimiento del poster.
- Control de las animaciones mediante variables de tiempo para asegurar movimientos suaves y realistas.

Interacción del Usuario:

- Implementación de una cámara controlada por el usuario que permite explorar la escena desde diferentes ángulos y perspectivas.
- Manejo de eventos de teclado y ratón para mover la cámara y activar animaciones.

Descripción General del Código

El código se estructura en varias secciones que trabajan juntas para crear la escena final. A continuación se describen las partes más importantes del código:

- Dependencias y Bibliotecas: Se incluyen todas las bibliotecas necesarias para el funcionamiento del programa, tales como GLEW, GLFW, GLM y stb_image.
- Variables Globales: Se definen variables globales para gestionar la ventana, la cámara, las texturas, los modelos, las luces y las animaciones.
- Funciones de Utilidad:
 - calcAverageNormals: Calcula las normales promedio para los vértices de los modelos, mejorando así la calidad de la iluminación.

- CreateObjects: Crea los objetos que serán renderizados en la escena, como el piso y otros modelos 3D.
 - CreateShaders: Crea y compila los shaders utilizados para el renderizado de la escena.
-
- Función Principal (main): Inicializa la ventana, crea los objetos y shaders, carga las texturas y modelos, y controla el loop de renderizado donde se actualizan las animaciones y se renderiza la escena en cada frame.

Técnicas y Conceptos Utilizados

OpenGL y Shaders:

- OpenGL es la biblioteca principal utilizada para renderizar gráficos en 3D. Se utiliza junto con shaders escritos en GLSL para realizar operaciones de iluminación y texturizado directamente en la GPU.

GLFW y GLEW:

- GLFW se utiliza para crear y gestionar la ventana y los contextos de OpenGL, así como para manejar la entrada del usuario.
- GLEW proporciona un mecanismo para consultar y cargar extensiones de OpenGL, lo que permite utilizar funcionalidades más avanzadas de la API de OpenGL.

GLM:

- GLM (OpenGL Mathematics) es una biblioteca matemática que facilita las operaciones vectoriales y matriciales necesarias para los cálculos de transformaciones y proyecciones en 3D.

Illuminación y Materiales:

- Se implementan diferentes tipos de luces (direccional, puntual y spotlight) para crear una iluminación realista en la escena.
- Los materiales se definen con propiedades específicas de especularidad y brillo, permitiendo simular diferentes tipos de superficies.

Carga de Texturas y Modelos:

- Se utilizan texturas para dar realismo a las superficies de los modelos. La librería stb_image se encarga de cargar las texturas desde archivos de imagen.
- Los modelos 3D se cargan desde archivos OBJ, permitiendo la inclusión de objetos complejos en la escena.

Animación:

- Las animaciones se controlan mediante variables de tiempo, lo que permite movimientos suaves y naturales de los objetos en la escena.
- Se programan varias animaciones, como la apertura de puertas y el movimiento de muebles, para dar vida a la escena.

Descripción del código, Gantt y análisis de costos

Modelos, texturas y skybox

Primeramente se declaran los objetos, texturas y skybox:

```
//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;
```

Captura 1. Declaración de elementos.

De manera continua con primitivas se crea el piso.

```
//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);
```

Captura 2. Piso utilizando primitivas.

En el main se llama a la función para el piso y se cargan texturas, modelos y el skybox.

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

    CreateObjects();
```

Captura 3. Carga de piso.

```
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");
251
```

Captura 4. Carga de textura de piso y modelos.

```

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"); //back
270
271     skybox = Skybox(skyboxFaces);

```

Captura 5. Carga de modelos y skybox.

Ahora en el *while*, agregamos los modelos y los acomodamos en el mapa.

```

//*****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();

```

Captura 6. Implementación de piso y modelos.

```

//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));

```

Captura 7. Implementación de modelos de planta alta.

```

//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();

```

Captura 8. Implementación de modelos de planta alta.

```

//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();

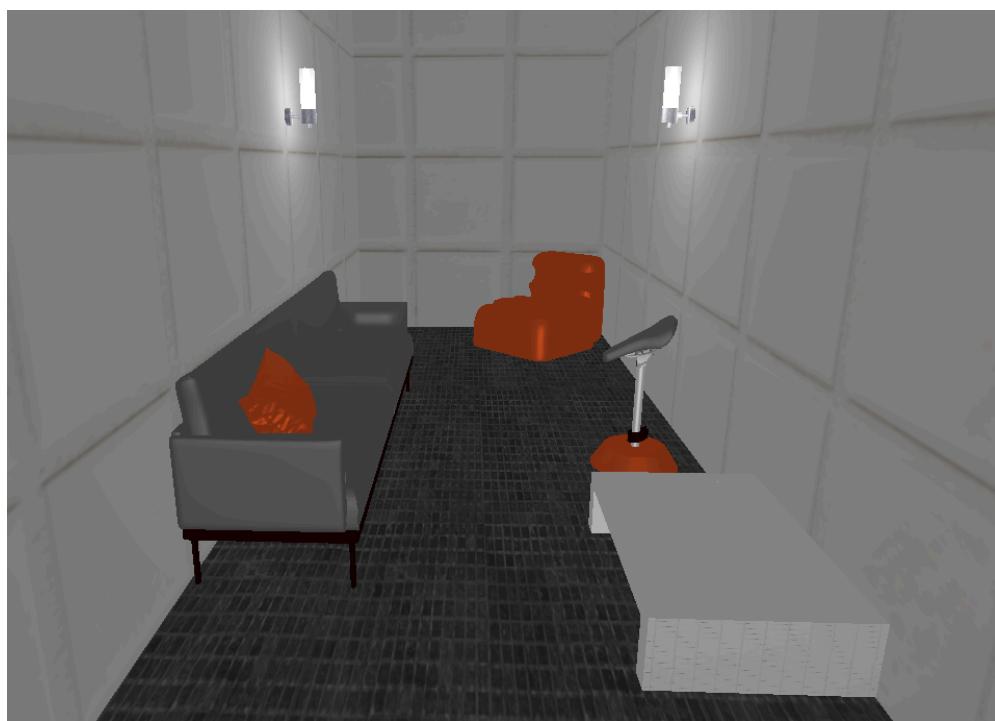
```

Captura 9. Implementación de modelos de planta baja.

Ahora ejecutando el programa se puede apreciar como se genero todo correctamente, modelos, skybox y el piso con su textura.



Captura 10. Resultado de modelo de casa, skybox y piso.



Captura 11. Resultado de objetos de la planta baja.



Captura 12. Resultado de objetos de la planta alta.

Recorrido

Para recorrer el mapa usamos la cámara estática, por lo tanto la declaramos.

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

Captura 13. Declaración de cámara estática.

Ahora en el main le damos los valores a cámara, donde los primeros argumentos son su posición, seguidos por su dirección y ángulo de mira.

```
//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);
```

Captura 14. Valores de cámara.

Dentro del `while` le damos los movimientos de teclado y mouse a la cámara además de pasar el skybox que se podrá observar.

```
//Controles de cámara
camera.keyControl(mainWindow.getKeys(), 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 cámara vea el skybox
skybox.DrawSkybox(camera.calculateViewMatrix(), projection);
```

Captura 15. Movimiento y mira de cámara.

Ahora en el Camera.cpp configuramos las teclas y la velocidad seteada a 5 unidades por movimiento y la visión al mover el mouse.

```
//Controles Camara
void Camera::keyControl(bool* keys, GLfloat deltaTime)
{
    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();
}
```

Captura 16. Teclas para mover la cámara y su velocidad además de su control de mouse.

Por último actualizamos la matriz del shader para la posición y orientación de la cámara con el fin de cálculos de la iluminación.

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

Captura 17. Información para el shader para renderizar desde la perspectiva de la cámara.

Illuminación

Se declaran las luces puntuales y spotlight

```
//*****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++;
```

Captura 18. Declaración de luces puntuales y spot.

Declaramos la luz direccional y le pasamos la información al shader, queda aclarar que la luz dirección se declara en este punto para que actúe de diferente manera si estamos dentro o fuera de la casa.

```
//*****  
  
//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);
```

Captura 19. Declaración de luz main y paso de información al shader.

```
//*****  
  
//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);
```

Captura 20. Declaración de luz main sobre la casa y actualización.

```
//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);
```

Captura 21. Declaración de luz main sobre los objetos y actualización.

Las luces spotlight se verán en el apartado de animaciones ya que dependiendo la animación se mostrará un tipo de luz dependiente del deltaTime.

Animaciones

Primero modificamos *Window.h* y *Window.cpp* para crear funciones con el fin de usar las teclas para activar o desactivar las animaciones.

```
//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;
```

Captura 22. Funciones para obtener los valores de las variables declaradas en *Window.h*.

Ahora en *Window.cpp* le damos un valor a las variables de 1.0f, esto nos servirá como bandera para la implementación de animaciones en *PR.cpp*.

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

Captura 23. Valores de las variables declaradas *Window.cpp*.

Continuando con *Window.cpp* agregamos las acciones que se tomarán en dado caso de que se presionen las teclas Z, X, C y V, las cuales cambiarán su valor de 1.0f a -1.0f cada vez que se presionen.

```
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;  
}
```

Captura 24. Comportamiento de las variables declaradas y teclas asignadas en *Window.cpp*.

Ahora en PR.cpp declaramos las variables globales que se utilizaran para las animaciones de los objetos y luces spot. Se tiene el movimiento que se aplicará a los modelos, el offset que funcionará para modificar la velocidad del movimiento y la dirección del objeto que cambiará dependiendo las variables booleanas dir. En la luz de póster se cuenta con spotlight donde el rango de luces se utilizará para determinar el cambio de colores.

```
//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
```

Captura 25. Declaración de variables.

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

Captura 26. Modificación de offsets.

Dentro del while tenemos los valores para determinar el deltaTime, seguido de la primera animación que aumentará dependiendo el valor de la misma variable deltaTime, esta animación se aplica al modelo de AsBikeSup, corresponde al asiento del banco en forma de asiento de bici.

```

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

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

```

Captura 27. Animación de asiento de bici.

```

//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();

```

Captura 28. Implementacion de movBici en la rotación del modelo en el eje Y.

Para la animación de la puerta y el cajón se utiliza un algoritmo similar, tomando el valor actual al presionar la tecla, valores de offset y rangos de movimiento delimitados por ifs interiores apoyados de banderas para cambiar el sentido del movimiento, además que el movimiento del cajón es de traslación y el de la puerta de rotación como se aprecia en los siguientes bloques de código.

<pre> //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; } } } </pre>	<pre> //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; } } } </pre>
---	--

Captura 28. Animación movPuerta y movCajon.

```
//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();
```

Captura 29. Implementación movPuerta a la rotación del objeto en el eje Y.

```
//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();
```

Captura 30. Implementación movCajon a la traslación del objeto en el eje Z.

Ahora pasando a las animaciones del poster contamos con la rotación que aumenta en función de deltaTime. Además del cambio de luces spot dependiendo el valor de lucesPoster apoyada del offset para la velocidad de cambio multiplicada por el valor de deltaTime, esto dentro de if exterior con la bandera de rangoLuces que cambiará dependiendo el valor de lucesPoster, los rangos son ajustados de -5 a 15, donde cobrará más sentido en la implementación explicada más adelante.

```
//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;
    }
}
```

Captura 31. Animación movPoster y lucesPoster.

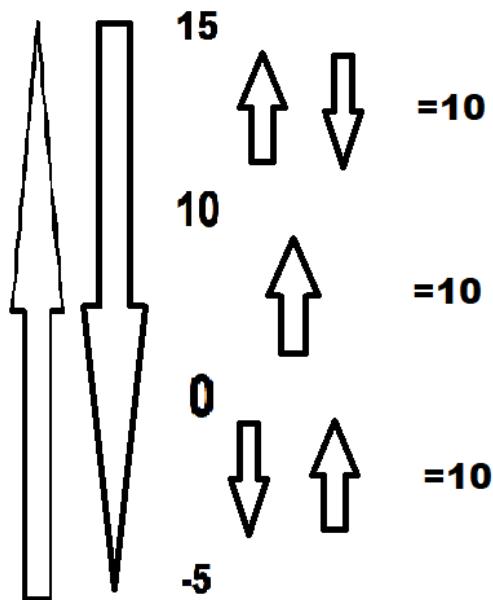
Para la rotación del objeto, se aplica movPoster sobre el eje Z utilizando una función seno para que el movimiento sea constante multiplicado por un valor de 10 para aumentar el giro que se da.

```
//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();
```

Captura 32. Implementación de rotación de lucesPoster al eje Z.

Por último para las luces spotlight se crean las siguientes condicionales para cambiar entre la luz seleccionada (declarada en el apartado de iluminación). Tomando en cuenta que los rangos van de -5 a 15, cada if tiene un rango de 10 (Debido a que lucesPoster al llegar al límite superior o inferior tiene que volver a hacer la disminución o aumento del valor, se muestra un ejemplo a continuación).



Captura 33. Funcionamiento de condicionales con rangos establecidos.

```

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

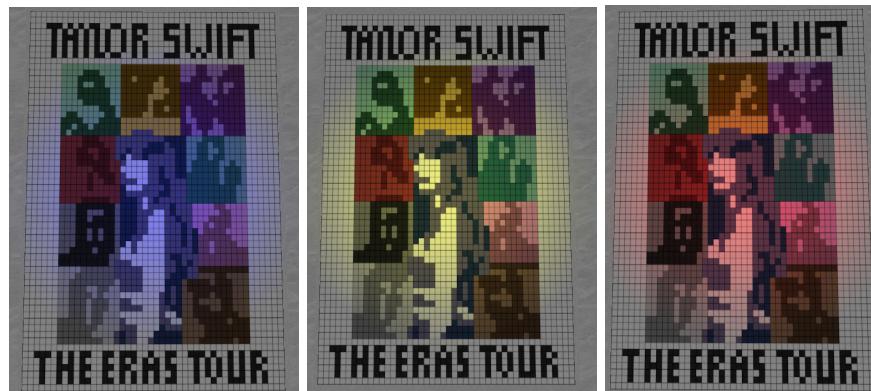
else if (lucesPoster <= 10.0f && lucesPoster > 0.0f)
{
    shaderList[0].SetSpotLights(spotLights, spotLightCount - 1); //Verde
}

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

```

Captura 34. Condicionales con rangos establecidos y asignación de spotlight a mostrar.

De esta manera las luces iteran de la siguiente manera.



Captura 35. Cambio de color dependiendo spotlight usada en el shader.

Diagrama de Gantt

El siguiente diagrama de Gantt representa la planificación detallada y el seguimiento del progreso realizado en el desarrollo de un proyecto durante el periodo del 17 al 20 de mayo de 2024. El proyecto se centra en la creación y adecuación de un entorno virtual, incluyendo la organización de documentos en plataformas digitales y el modelado tridimensional de estructuras y objetos. A lo largo de este documento, se detallan las actividades realizadas, el tiempo empleado en cada una de ellas, y los hitos alcanzados para el avance efectivo del proyecto.

El diagrama se estructura en torno a las fases clave del proyecto, desde la inicial organización de recursos digitales hasta la implementación de elementos visuales y técnicos necesarios para su finalización. Cada actividad se describe con precisión, reflejando el esfuerzo y la dedicación empleados para alcanzar los objetivos establecidos en el cronograma.

Este registro no solo documenta las tareas realizadas, sino que también sirve como herramienta para evaluar la eficiencia del proceso y la gestión del tiempo, proporcionando una visión integral del desarrollo del proyecto en su conjunto.

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

Captura 36. Diagrama de Gantt.

Bitácora

17/05/24

- Propósito: Crear carpeta, documentos en Drive y github.
- Tiempo Empleado: 30 minutos.

17/05/24

- Propósito: Adecuar proyecto base limpio.
- Tiempo Empleado: 40 minutos.

18/05/24 - 19/05/24

- Propósito: Piso textura, modelado de casa, cuartos y objetos de planta alta.
- Tiempo Empleado: 5 horas 20 minutos.

19/05/24

- Propósito: Objetos de planta baja modelado y texturizado, acomodado de todos los objetos y 2 animaciones.
- Tiempo Empleado: 5 horas

19/05/24

- Propósito: Luces, 2 animaciones de póster y luces spot.
- Tiempo Empleado: 3 horas

20/05/24

- Propósito: Gantt, Manual técnico, manual de usuario y archivos ejecutables.
- Tiempo Empleado: 3 horas

Análisis de costos

Costos Variables:

- Tarifa por hora de programador de videojuegos: \$19,345 MXN al mes (Aproximado).
- Horas dedicadas al proyecto: 25 horas.

Costos Fijos Mensuales:

- Licencia de Windows 10: \$3,609 MXN.
- Licencia de 3D Max (para tres años): \$5,100 MXN.
- GIMP: \$200 MXN.
- Comidas diarias (6 días a \$100 pesos por día): \$600 MXN.
- Luz (cada dos meses, se trabajó 6 días): \$200 MXN (proporcional a un mes).

Costo de Venta:

- Porcentaje adicional al costo operativo: 33%.
- Impuestos 16% de IVA

Cálculo de Costos y Precio de Venta:

Costos Variables:

- Costo por hora: \$19,345 MXN / 160 horas (160 al mes se trabajan) = \$120.91 MXN por hora.
- Horas dedicadas: 25 horas.
- Costo variable total = \$120.91 MXN/hora * 25 horas = \$3,022.75 MXN.

Costos Fijos:

- Licencia de Windows 10: \$3,609 MXN.
- Licencia de 3D Max (anualizado a un mes): \$5,100 MXN
- GIMP: \$200 MXN.
- Comidas diarias: \$600 MXN.
- Luz : \$200 MXN.

Total costos fijos mensuales = \$3,609 MXN + \$5,100 MXN + \$200 MXN + \$600 MXN + \$200 MXN = \$9,709 MXN.

Costo Operativo Total:

- Costos variables + Costos fijos mensuales = \$3,022.75 MXN + \$9,709 MXN = \$12,731.75 MXN.

Precio de Venta:

- Costo operativo total * (1 + Porcentaje adicional para ganancia) = \$12,731.75 MXN * (1 + 0.33) = \$16,933.22 MXN.

Aplicación de Impuestos:

- Precio final con 16% de IVA = \$16,933.22 MXN * 1.16 = **\$19,642.54 MXN.**

Conclusión

Este proyecto me enseñó la importancia de una planificación meticulosa y la autogestión en el desarrollo de software. A lo largo del desarrollo de este proyecto, se aplicaron diversas técnicas de gráficos por computadora utilizando OpenGL, lo cual permitió crear una escena 3D compleja y visualmente atractiva. La implementación de iluminación dinámica, carga de modelos 3D y animaciones de objetos fueron aspectos clave para lograr una buena experiencia.

El proceso comenzó con la definición clara de objetivos, donde se estableció la meta de crear una escena 3D realista y animada. Para lograr esto, se dividió el proyecto en tareas manejables, cada una con su propio conjunto de problemas y aprendizajes. Este enfoque modular permitió un desarrollo más organizado y menos propenso a errores.

La implementación de la iluminación dinámica fue uno de los aspectos más complicados a medias del proyecto. Crear diferentes tipos de luces, incluyendo luces direccionales, puntuales y spotlights, añadió un nivel de realismo significativo

a la escena. Este aspecto del proyecto no solo mejoró mis habilidades, sino que también profundizó mi comprensión de cómo la luz interactúa con los objetos en un entorno tridimensional.

La programación de animaciones para los objetos en la escena fue otro componente fundamental. Controlar las animaciones mediante variables de tiempo para asegurar movimientos suaves y realistas demandó precisión y atención al detalle. La implementación de una cámara controlada por el usuario, que permite explorar la escena desde diferentes ángulos y perspectivas, mejoró aún más la interactividad y el atractivo visual del proyecto.

Finalmente, la gestión efectiva del tiempo y los recursos fue esencial para el éxito del proyecto. Utilizar herramientas como el diagrama de Gantt y realizar un análisis de costos detallado permitió una planificación precisa y un seguimiento continuo del progreso. Estas herramientas no solo ayudaron a mantener el proyecto dentro de los plazos establecidos, sino que también proporcionaron una visión clara de la eficiencia del proceso y la gestión del tiempo. En resumen, este proyecto fue una experiencia muy enriquecedora que me ayudó a mejorar mis capacidades técnicas como mis habilidades de organización y autogestión, preparándome mejor para futuros proyectos en el campo de la ingeniería en computación.

Créditos

Modelos

- Cama
 - Nombre del archivo: 3D Bed Free Obj Model Free 3D model
 - Procedencia: cgtrader
 - Autor: 3dmodelbase
 - Tipo de Licencia: Royalty Free No Ai License
 - Enlace: [3D Bed Free Obj Model free 3D model | CGTrader](https://www.cgtrader.com/3d-models/free-3d-models/3d-bed-free-obj-model-free-3d-model)
- Jarrón

- Nombre del archivo: Vase 01
 - Procedencia: cgtrader
 - Autor: hend-z
 - Tipo de Licencia: Royalty Free No Ai License
 - Enlace: [Vase 01 free 3D model | CGTrader](#)
- Buro
 - Nombre del archivo: Fermo Double Dresser
 - Procedencia: cgtrader
 - Autor: nmcygni
 - Tipo de Licencia: Royalty Free No Ai License
 - Enlace: [Fermo Double Dresser free VR / AR / low-poly 3D model | CGTrader](#)
- Silla
 - Nombre del archivo: Ring Stool
 - Procedencia: cgtrader
 - Autor: litat
 - Tipo de Licencia: Royalty Free No Ai License
 - Enlace: [Ring Stool free 3D model | CGTrader](#)
- Sillon Doble
 - Nombre del archivo: Furniture Applaryd IKEA Sofa Couch
 - Procedencia: cgtrader
 - Autor: timur560
 - Tipo de Licencia: Royalty Free No Ai License
 - Enlace: [Furniture Applaryd IKEA Sofa Couch free 3D model | CGTrader](#)
- Sillon Individual
 - Nombre del archivo: Sofa modular
 - Procedencia: cgtrader
 - Autor: kirmisas
 - Tipo de Licencia: Royalty Free No Ai License
 - Enlace: [Sofa modular free 3D model | CGTrader](#)

- Lámpara
 - Nombre del archivo: Candelabro de pared modelo 3d
 - Procedencia: turbosquid
 - Autor: Piotr Zatarski
 - Tipo de Licencia: 3D Model License
 - Enlace: [modelo 3d Candelabro de pared gratis - TurboSquid 1539085](https://www.turbosquid.com/3d-models/candelabro-de-pared-gratis-turbo-squid-1539085)
- Asiento Bicicleta
 - Nombre del archivo: asiento de bicicleta modelo 3d
 - Procedencia: turbosquid
 - Autor: d.w.designs
 - Tipo de Licencia: 3D Model License
 - Enlace: [modelo 3d asiento de bicicleta gratis - TurboSquid 729638](https://www.turbosquid.com/3d-models/asiento-de-bicicleta-gratis-turbo-squid-729638)

Texturas

- Taylor.tga
 - Nombre del archivo: Alpha pattern #148909
 - Procedencia: braceletbook
 - Autor: ketchupfly
 - Tipo de Licencia: Royalty Free No Ai License
 - Enlace: [Alpha pattern #148909 | BraceletBook](https://www.braceletbook.com/alpha-pattern-148909)
- Piso.tga
 - Procedencia: Freepik.
 - Autor: freepik.
 - Tipo de Licencia: Gratis para uso personal y comercial con atribución.
 - Enlace: [Cerrar textura de asfalto | Foto Gratis \(freepik.es\)](https://www.freepik.es/fotos-gratis/cerrar-textura-de-asfalto)
- casa.tga
 - Procedencia: admagazine
 - Autor: Katie Schultz

- Tipo de Licencia: Gratis para uso personal y comercial con atribución.
 - Enlace: [Taylor Swift: La casa de “Cornelia Street” está en renta | Architectural Digest \(admagazine.com\)](#)
- GravaTs.tga
 - Procedencia: Freepik.
 - Autor: freepik
 - Tipo de Licencia: Gratis para uso personal y comercial con atribución.
 - Enlace: [Fondo de textura granulada | Foto Gratis \(freepik.es\)](#)
- pisoPB.tga
 - Procedencia: Freepik.
 - Autor: kues1
 - Tipo de Licencia: Gratis para uso personal y comercial con atribución.
 - Enlace: [Textura de la pared del embutido negro | Foto Gratis \(freepik.es\)](#)
- azulejoBlanco.tga
 - Procedencia: Freepik.
 - Autor: lifeforstock
 - Tipo de Licencia: Gratis para uso personal y comercial con atribución.
 - Enlace: [Pared de azulejos blancos | Foto Gratis \(freepik.es\)](#)
- metalico.tga
 - Procedencia: Freepik.
 - Autor: rawpixel.com
 - Tipo de Licencia: Gratis para uso personal y comercial con atribución.
 - Enlace: [Fondo plateado metalizado texturado. | Foto Gratis \(freepik.es\)](#)