Manual técnico

1.- Introducción

Este documento proporciona una visión detallada de la metodología, la arquitectura, el diccionario de variables y funciones, los fundamentos matemáticos de las animaciones complejas, el diagrama de Gantt, las limitaciones, los objetivos y las conclusiones de nuestro proyecto. En él, hemos modelado la Casa de "Up" de la película "Una aventura de altura" utilizando OpenGL y Maya como motor para los modelos.

2.- Objetivos

1. Elaborar un manual técnico y de usuario detallado que incluya la metodología de software aplicada, proporcionando una guía completa para la interacción dentro del ambiente virtual recreado.
2. Mantener actualizado y funcional un repositorio del proyecto, mostrando de forma clara los cambios y avances realizados durante el semestre, sin utilizar archivos comprimidos.
3. Lograr un alto grado de realismo en el espacio virtual recreado, comparado con la foto de referencia proporcionada.
4. Desarrollar un archivo ejecutable que pueda abrirse y ejecutarse correctamente, garantizando que no se incluya el código fuente en la entrega final.
5. Modelar con precisión un mínimo de 7 elementos especificados en el documento de referencia, asegurando un texturizado correcto y una geometría adecuada para los muebles y edificios.
6. Crear 5 animaciones, incluyendo 3 sencillas y 2 complejas, que sean coherentes con el contexto del espacio recreado.
7. Ambientar el espacio virtual con un uso correcto de la iluminación, contribuyendo a la atmósfera general del proyecto.
8. Implementar un manejo de cámara adecuado que permita una experiencia de usuario fluida y natural dentro del ambiente virtual.

3.- Requerimientos:

3.1 Requerimientos Funcionales:

1. **Recreación 3D:** El sistema debe permitir la recreación en 3D de una fachada y un espacio seleccionados, basados en imágenes de referencia proporcionadas por el usuario.
2. **Objetos a Recrear**: Deben ser recreados 7 objetos específicos, lo más parecido posible a su imagen de referencia, así como su ambientación.
3. **Manuales:** Se debe proporcionar un manual de usuario que explique cada interacción dentro del ambiente virtual recreado, así como un manual técnico que contenga la documentación del proyecto, objetivos, diagrama de flujo del software, diagrama de Gantt, alcance del proyecto, limitantes, metodología de software, documentación del código y conclusiones.
4. **Entrega y Plataforma:** El proyecto debe entregarse de forma individual en un repositorio en GitHub y subirse a la plataforma de classroom antes de la fecha límite especificada.
5. **Manejo de Cámara**: Se deben implementar controles para el manejo de la cámara dentro del espacio virtual recreado.
6. **Ejecutable:** Se debe proporcionar un archivo ejecutable que exista, abra, funcione y no contenga el código fuente.
7. **Ambientación y Realismo:** Se debe tener un uso correcto de la iluminación dentro del espacio y lograr un realismo del espacio virtual contra la foto de referencia.
8. **Modelado y Texturizado:** Se debe recrear un mínimo de 7 elementos especificados en las imágenes de referencia, con un texturizado correcto y geometría adecuada.

3.2 Requerimientos No Funcionales:

1. **Idioma de Documentación:** La documentación del proyecto debe entregarse tanto en español como en inglés, sin utilizar Google Translate.
2. **Uso de Código Base:** Se debe utilizar el código base proporcionado durante el curso; de lo contrario, el proyecto será anulado y el alumno no acreditará el laboratorio.

3. **Prohibiciones en el Tema:** Queda prohibido recrear espacios pertenecientes a ciertas temáticas específicas, como universidades, series animadas, videojuegos con bajas características gráficas, entre otros.
4. **Complejidad de Animaciones:** Las animaciones deben tener contexto y no pueden ser lineales; se deben realizar 5 animaciones, donde 3 sean sencillas y 2 complejas.
5. **Visualización de Cambios y Avances:** El repositorio debe mostrar una visualización de cambios y avances durante el semestre, sin archivos comprimidos y funcional.

4.- Metodología

El proyecto "La Casa de Up: Una Aventura de Altura" se desarrolló siguiendo la metodología ágil Scrum. Scrum es un marco de trabajo que nos permite gestionar de manera eficiente el desarrollo de este proyecto complejo y creativo, garantizando la entrega de un producto de alta calidad en tiempos y costos controlados.

4.1 Equipos de desarrollo

- El equipo de desarrollo está compuesto por 1 persona, el cual desarrolló las acciones de modelador, programador, tester y animación.

4.2 Backlog del producto

- Al comienzo del proyecto, se creó un backlog del producto que contenía todas las características y funcionalidades que se deseaban incluir en "La Casa de Up: Una Aventura de Altura".
- El backlog se prioriza en función del valor para el usuario y la complejidad técnica.

Modelado de la Casa

- **Modelado de la Casa de Up:** incluyen todos los detalles exteriores e interiores.

Modelado de Objetos

- **Modelado de los siguientes objetos interiores de la casa:** sofá, mesa de centro, lámpara, sillón de Carl, silla de Ellie, tapete y cojines.

Animaciones Sencillas

- Animación de la lámpara encendiéndose y apagándose.
- Animación de la puerta principal abriéndose y cerrándose.
- Animación del cajón abriéndose y cerrándose.

Animaciones Complejas

- Animación de los globos flotando en la parte superior de la fachada
- Animación del globo flotando en el florero al interior de la casa.

Fachada de la Casa

- Modelado y texturizado de la fachada de la Casa de Up, incluyendo la pintura desgastada y las tejas del tejado.

Interacción del Usuario

- Implementación de la interacción del usuario para permitirle abrir y cerrar la puerta y la ventana, encender y apagar la lámpara y abrir y cerrar el cajón del mueble.

Optimización y Pruebas

- Optimización de la escena mediante la limpieza, baja topología y texturizado para evitar tiempos de carga altos.

Sprints

- El trabajo se dividió en sprints de duración fija de una semana.
- Al comienzo de cada sprint, se realizó una planificación en la que se seleccionan las tareas a realizar en ese sprint a partir del backlog del producto y apoyados del diagrama de Gantt.

Reuniones de revisión y retrospectiva

- Al final de cada sprint, se realizó una revisión en la que se demostró el trabajo realizado.

Entrega incremental

- Siguiendo el enfoque ágil, se realizaron entregas incrementales de funcionalidades, lo que permitió obtener retroalimentación temprana y ajustar el desarrollo según las necesidades del proyecto.

5.- Arquitectura del Sistema

El programa está diseñado para ofrecer una experiencia interactiva a los usuarios. Está compuesto por varios componentes clave que trabajan juntos para crear un entorno virtual detallado y realista. A continuación, se describen los principales componentes de la arquitectura del sistema:

1. **OpenGL**: Se utiliza como API gráfica para renderizar los gráficos en tiempo real. OpenGL proporciona las herramientas necesarias para crear efectos visuales avanzados y renderizar modelos 3D de manera eficiente.
2. **Maya**: Se utiliza como software de modelado y animación para crear los modelos 3D de la Casa de Up y otros objetos dentro del entorno. Maya se integra con OpenGL para exportar los modelos y animaciones al formato compatible con el proyecto.
3. **Interacción del usuario:** Se implementa una interfaz de usuario intuitiva que permite a los usuarios interactuar con el entorno virtual. Esta interfaz se basa en eventos de entrada del usuario, como movimiento del ratón y pulsaciones de teclas, para controlar la cámara y activar animaciones y efectos especiales.
4. **Gestión de activos**: Se utiliza un sistema de gestión de activos como SOIL 2 para cargar y gestionar los modelos 3D, texturas, animaciones y otros recursos utilizados en el proyecto. Esto garantiza que los recursos se carguen de manera eficiente y se mantengan en memoria durante el uso del programa.

5. **Optimización y rendimiento:** Se han aplicado técnicas de optimización para garantizar un rendimiento fluido en una amplia gama de dispositivos. Esto incluye el uso de técnicas de renderizado eficientes, como triangulación, texturizado bajo mapas uv, iluminación direccional de bajo costo y topología baja de elementos.

6.- Diccionario de variables y funciones

Variables globales:

- rotp: Controla la rotación de un objeto.
- activaanim1, activaanim2, activaanim3: Variables booleanas que controlan la activación de diferentes animaciones.
- tras1: Controla la traslación de un objeto.
- WIDTH, HEIGHT: Dimensiones de la ventana.
- SCREEN_WIDTH, SCREEN_HEIGHT: Dimensiones del framebuffer.
- camera: Objeto de la clase Camera que controla la cámara en la escena.
- lastX, lastY: Últimas coordenadas del cursor.
- keys: Arreglo de teclas para controlar el movimiento.
- firstMouse: Booleano que indica si es la primera vez que se mueve el cursor.
- lightPos: Posición de la luz en la escena.
- active, active2, active3: Variables booleanas para controlar la activación de luces.
- tiempo: Variable para almacenar el tiempo actual de la aplicación.
- pointLightPositions, ubfoc, dirlamp: Posiciones de diferentes luces en la escena.
- vertices: Arreglo que contiene los vértices de un cubo.

Funciones:

- main(): Función principal que inicializa GLFW, GLEW, crea la ventana y el contexto de OpenGL, carga los shaders y modelos, y contiene el bucle principal de la aplicación.
- KeyCallback(): Función de devolución de llamada que maneja los eventos de teclado.

- MouseCallback(): Función de devolución de llamada que maneja los eventos del mouse.
- DoMovement(): Función que procesa el movimiento de la cámara según las teclas presionadas.

7.- Fundamentos matemáticos de animaciones complejas

Animación 1 - Shader Anim

La expresión *sin(time * speed) * 0.1* en el shader representa el cálculo del desplazamiento vertical basado en una función sinusoidal.

- *time* es una variable que proporciona el tiempo actual del sistema en segundos.
- *speed* es un valor que determina la velocidad a la que se produce la oscilación. Cuanto mayor sea speed, más rápido cambiará el desplazamiento.
- *time * speed* multiplica el tiempo por la velocidad, lo que significa que el argumento de la función seno cambia con el tiempo y la velocidad.
- *sin(time * speed)* calcula el seno de time * speed, lo que produce un valor oscilante entre -1 y 1 a lo largo del tiempo.
- *sin(time * speed) * 0.1* multiplica el resultado por 0.1, lo que reduce la amplitud de la oscilación a un rango más pequeño, de modo que el desplazamiento resultante sea más sutil.

Animación 2 - Shader 2

La función *asin(time/100)*PI* calcula el ángulo cuyo seno es time/100, escalando el tiempo para ajustar el rango de entrada de la función asin. Luego, este ángulo se multiplica por PI para convertirlo a radianes. Este cálculo se utiliza para generar un efecto de onda periódica a lo largo del tiempo.

8.- Actualización y versionado

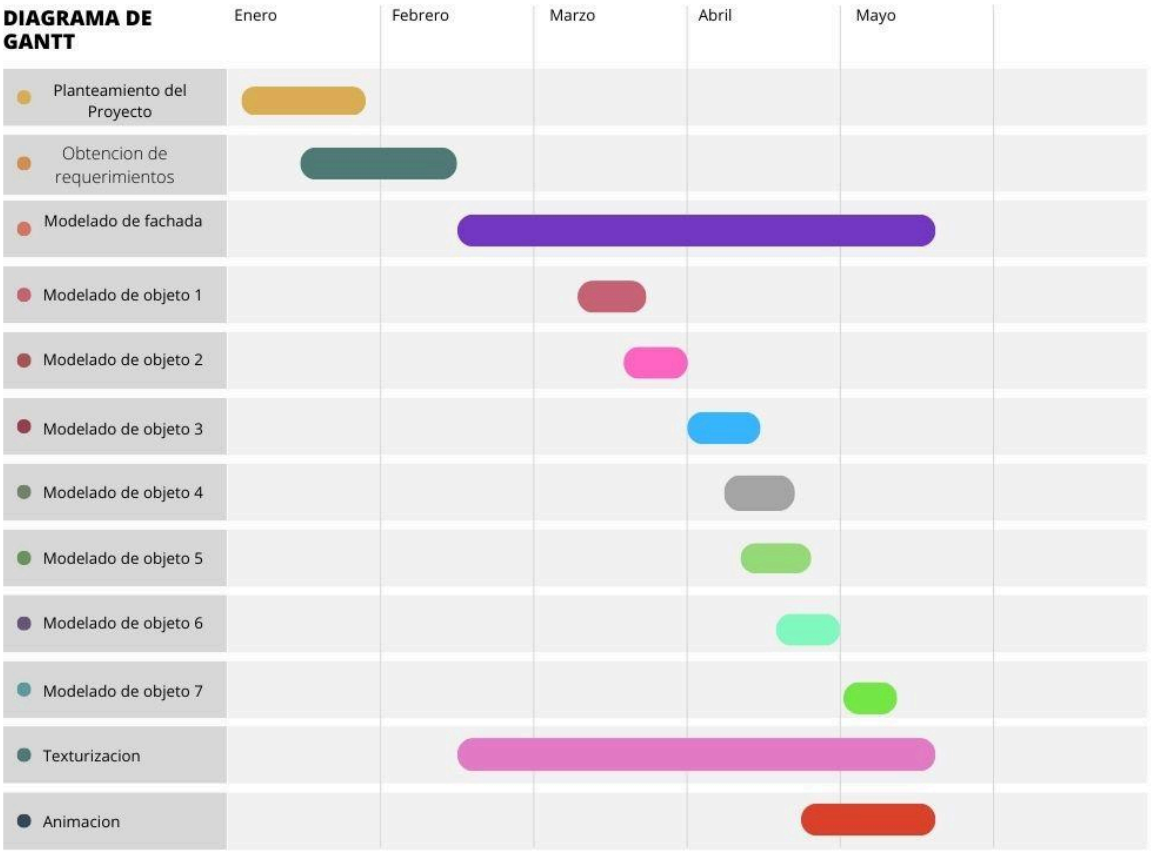
Para el versionamiento del proyecto, se utilizó GitHub junto con Visual Studio. GitHub es una plataforma de alojamiento de código que utiliza Git para el control de versiones. Git es un sistema de control de versiones distribuido que permite

gestionar cambios en el código de manera eficiente, manteniendo un historial completo de todas las modificaciones realizadas.

- Visual Studio, por otro lado, es un entorno de desarrollo integrado (IDE) que ofrece herramientas para escribir, depurar y compilar código. Visual Studio tiene integración con Git, lo que facilita la gestión de versiones directamente desde el IDE.

- Al utilizar GitHub con Visual Studio, se puede llevar un control detallado de los cambios realizados en el código, permitiendo colaborar con otros desarrolladores de forma efectiva y mantener un historial de versiones que facilita la identificación y corrección de errores.

9.- Diagrama de Gantt



10.- Diagrama de flujo

Diagrama de flujo completo

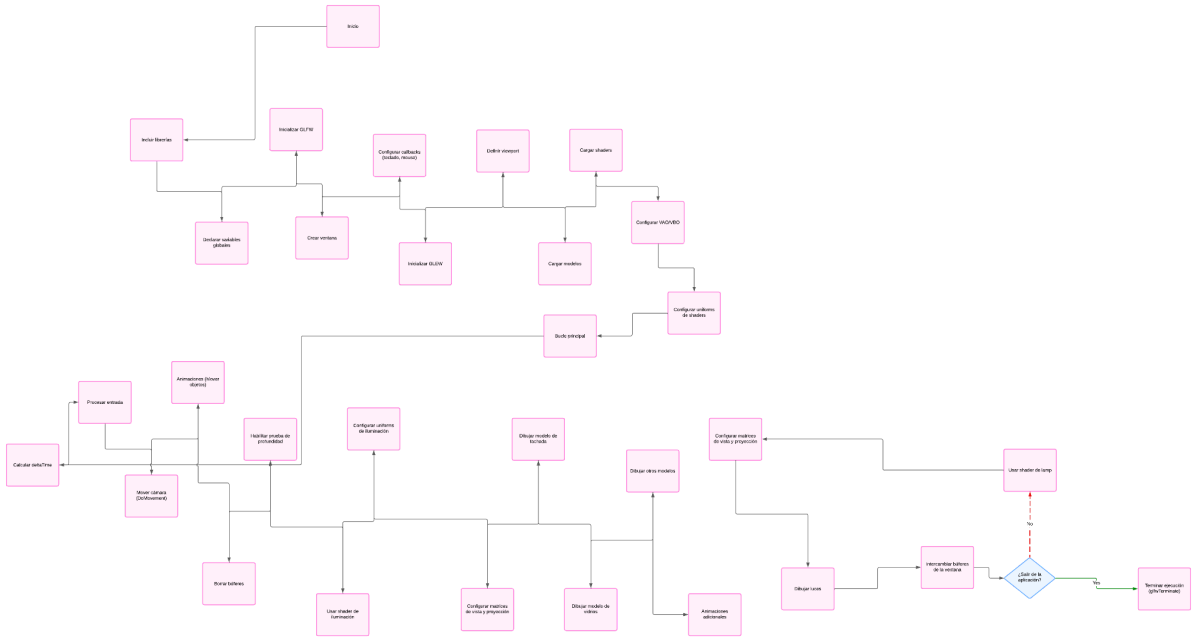
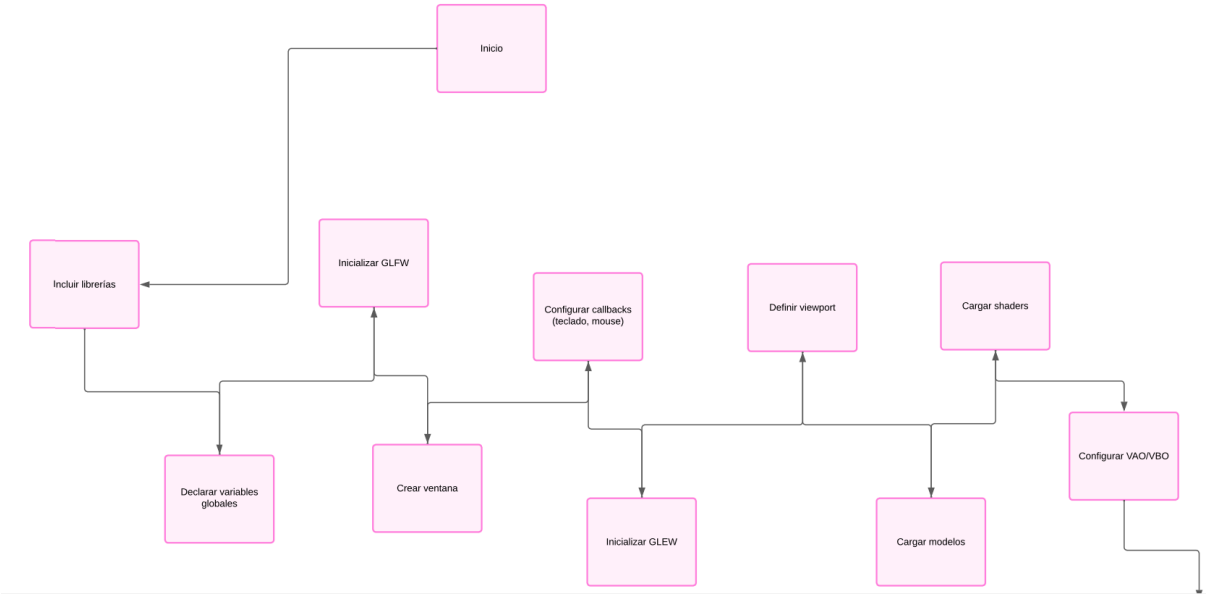
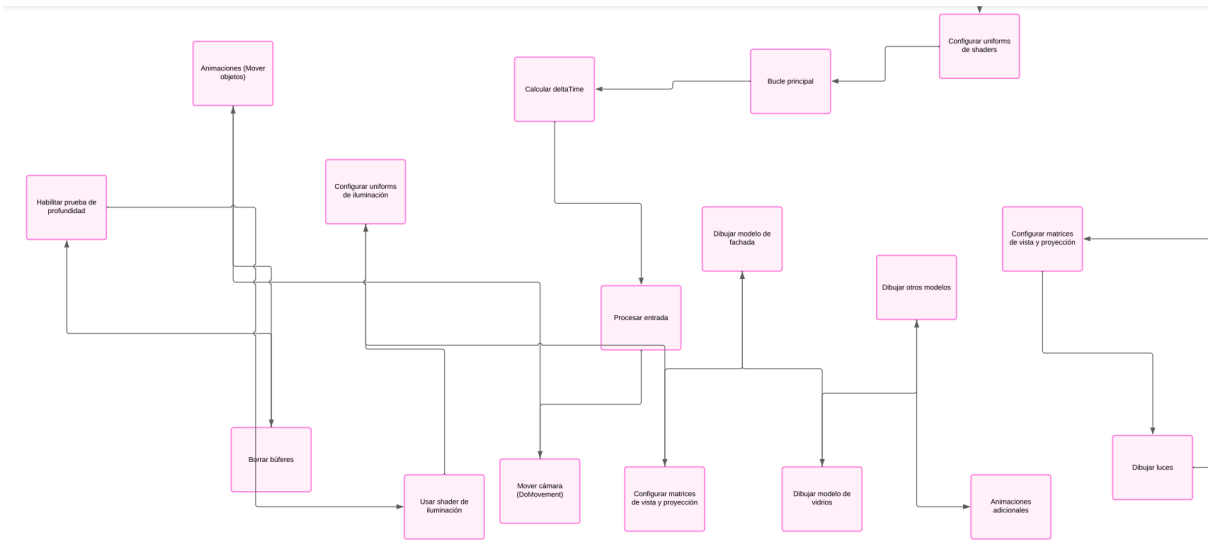
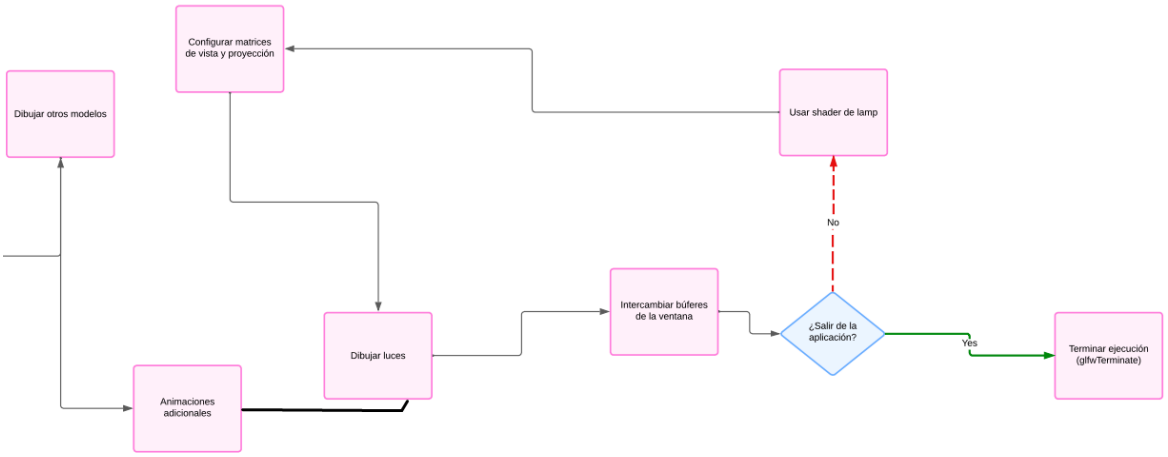


Diagrama de flujo por partes







11.- Explicación del código

Carga de shaders: En esta sección se hace la carga de shaders, los de animación, anim para la animación compleja 1, anim2 para la animación compleja 2, lightning y lampshader para la iluminación.

```
Shader lightingShader("Shaders/lighting.vs", "Shaders/lighting.frag");
Shader lampShader("Shaders/lamp.vs", "Shaders/lamp.frag");
Shader Anim("Shaders/anim.vs", "Shaders/anim.frag");
Shader Anim2("Shaders/anim2.vs", "Shaders/anim2.frag");
```

Carga de modelos:

Aquí se hace la carga de los modelos, se especifica la ruta y el nombre del objeto.

```
// Modelos
Model Modelouno((char*)"Models/Modelo1/modelouno.obj");
Model Modelodos((char*)"Models/Modelo2/modelodos.obj");
Model Modelotres((char*)"Models/Modelo3/modelotres.obj");
Model Modelocuatro((char*)"Models/Modelo4/modelocuatro.obj");
Model Modelocinco((char*)"Models/Modelo5/modelocinco.obj");
Model Modeloseis((char*)"Models/Modelo6/modeloseis.obj");
Model Modelosiete((char*)"Models/Modelo7/Modelosiete.obj");

// Fachada
Model Fachada((char*)"Models/Fachada/Fachada.obj");


// Vidrios
Model vd((char*)"Models/vd/vd.obj");


//Globos
Model globos((char*)"Models/globos/globos.obj");


//Puerta
Model puerta((char*)"Models/puerta/puerta.obj");

//Piso
Model Piso((char*)"Models/Piso/Piso.obj");

//Cajon
Model ca((char*)"Models/ca/ca.obj");

//Cajon
Model Flor((char*)"Models/flor/flor.obj");

//Cortina
Model Cortina((char*)"Models/cortina/cortina.obj");

//Cortina
Model globo((char*)"Models/globo/globo.obj");
```

Carga de modelos:

Aquí se describe el proceso de carga de modelos. Primero se configura la matriz, luego se aplican las características como traslaciones, transparencias y rotaciones.

```
//Carga de modelo
view = camera.GetViewMatrix();


//// Fachada
model = glm::mat4(1);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Fachada.Draw(lightingShader);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"),
glUniform4f(glGetUniformLocation(lightingShader.Program, "colorAlpha"), 1.0, 1.0
glEnable(GL_BLEND);


/// VIDRIOS
model = glm::mat4(1);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
vd.Draw(lightingShader);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1i(glGetUniformLocation(lightingShader.Program, "activaTransparencia"),
glUniform4f(glGetUniformLocation(lightingShader.Program, "colorAlpha"), 1.0, 1.0
glDisable(GL_BLEND);
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);  // Configura la funci?n de m

/// PUERTA
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-1.77f, 2.13f, 2.77f));
model = glm::rotate(model, glm::radians(rotp), glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
puerta.Draw(lightingShader);


/// Cajon
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(tras1, 0.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
ca.Draw(lightingShader);
```

Carga de shader para animaciones:

Aquí se puede hacer el uso de los shaders, se hace empleo de dos distintos donde se cargan las animaciones complejas.

```
if (activaanim3 == true)
{
    Anim2.Use();
    tiempo = glfwGetTime();
    // Get location objects for the matrices on the lamp shader (these could be different on a different shader)
    modelLoc = glGetUniformLocation(Anim2.Program, "model");
    viewLoc = glGetUniformLocation(Anim2.Program, "view");
    projLoc = glGetUniformLocation(Anim2.Program, "projection");

    // Set matrices
    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);
    globos.Draw(Anim2);
    glBindVertexArray(0);


}

Anim.Use();
tiempo = glfwGetTime();
// Get location objects for the matrices on the lamp shader (these could be different on a different shader)
modelLoc = glGetUniformLocation(Anim.Program, "model");
viewLoc = glGetUniformLocation(Anim.Program, "view");
projLoc = glGetUniformLocation(Anim.Program, "projection");

// Set matrices
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);
globo.Draw(Anim);
glBindVertexArray(0);
```

Movimiento de cámara:

En esta sección se hace uso de las teclas y de la función de DoMovement para el movimiento dentro del espacio.

```
void DoMovement()
{

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

    }

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


    }

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


    }

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


    }
```

12.- Limitaciones

- **Capacidad de Procesamiento:** El manejo de topologías de objetos más grandes requiere una capacidad de procesamiento mayor, lo que puede limitar la velocidad y la fluidez de la aplicación en hardware menos potente.
- **Generación de Mapas UV:** La falta de experiencia en la generación de mapas UV puede limitar la calidad y variedad de texturas aplicables a los objetos, afectando la apariencia visual de la escena.
- **Conocimiento en Modelado 3D:** La falta de conocimiento en modelado 3D puede limitar la diversidad y complejidad de los objetos que se pueden crear, restringiendo la creatividad y la variedad en la escena.
- **Complejidad de las Animaciones:** La complejidad de las animaciones puede estar limitada por la capacidad de procesamiento y la velocidad de fotogramas deseada, lo que puede afectar la fluidez y realismo de las animaciones.
- **Interactividad Limitada:** La interactividad de la aplicación puede verse limitada por la capacidad de procesamiento y la complejidad de los cálculos requeridos, lo que puede afectar la experiencia del usuario.

12.- Conclusiones

Este proyecto ha servido para demostrar las habilidades adquiridas durante el curso de computación gráfica, ha sido un proyecto desafiante por el uso de animaciones, transformaciones básicas, modelado 3d, animaciones complejas, texturización, la comprensión del espacio 3d. Se han podido cumplir los objetivos adquiridos durante el curso y los objetivos específicos del proyecto.