UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

DIVISIÓN DE INGENIERÍA ELÉCTRICA

INGENIERÍA EN COMPUTACIÓN

LABORATORIO DE COMPUTACIÓN GRÁFICA e INTERACCIÓN HUMANO COMPUTADORA

Manual Técnico Proyecto Final

NOMBRE COMPLETO:	N° de Cuenta:	Gpo Lab
-Hernández Alejo Ximena Gizell	317006126	11
-Hernández Hernández Samuel	316060497	07

GRUPO DE TEORÍA: 6

SEMESTRE 2024-2

FECHA DE ENTREGA LÍMITE: 21/Mayo/ 2024

CALIFICACIÓN: _____

Presentación

El siguiente manual se desarrolló con el objetivo de dar a conocer la información necesaria de la ejecución y usabilidad del proyecto final, el cual es un pequeño croquis de una zona turística, en este caso el parque ecológico de Xochimilco.

El presente manual ofrece la información necesaria para conocer el desarrollo del proyecto, para que en dado caso, donde se busque una nueva actualización, esta se pueda realizar de manera más fácil.

Objetivo

Presentar el desarrollo y funcionamiento del código del proyecto, hablando de los aspectos técnicos de una manera descriptiva e ilustrativa sobre los componentes y funcionalidades.

Requerimientos técnicos.

Requerimientos minimos recomendados de Hardware

- Procesador ARM64 o x64; se recomienda uno de cuatro núcleos o superior. No se admiten procesadores ARM 32.
- 4 GB de memoria, como mínimo. Hay muchos factores que afectan a los recursos usados, se recomiendan 16 GB de RAM para soluciones profesionales típicas.
- Windows 365: 2 vCPU y 8 GB de RAM, como mínimo. Se recomiendan 4 vCPU y 16 GB de RAM.
- Espacio en disco duro: mínimo de 850 MB y hasta 210 GB de espacio disponible, en función de las características instaladas; las instalaciones típicas requieren entre 20 y 50 GB de espacio libre. Se recomienda instalar Windows y Visual Studio en una unidad de estado sólido (SSD) para mejorar el rendimiento.
- Tarjeta de vídeo que admita una resolución de pantalla mínima de WXGA (1366 x 768); Visual Studio funcionará mejor con una resolución de 1920 x 1080 o superior.

Requerimientos minimos recomendados de Software

- OpenGL.
- “Visual Studio” 2017 o 2022.
    - Versión de SO mínima o superior de Windows 11 compatible: Home, Pro, Pro Education, Pro para estaciones de trabajo, Enterprise y Education
    - Versión de SO mínima o superior de Windows 10 compatible: Home, Professional, Education y Enterprise.
- Programa de edición de imágenes. Varían dependiendo del programa a usar.
- Programa de modelado (Recomendado 3ds Max).
    - Sistema operativo. Versión de 64 bits de Microsoft® Windows 7 o superior.
    - Explorador. Versiones mas recientes de:
        - Microsoft® Edge

- Google Chrome™
- Microsoft® Internet Explorer®
- Mozilla® Firefox®

Software de desarrollo

Durante el desarrollo del proyecto se usaron varios software dependiendo de la parte en la que se trabajó.

Visual Studio. Software usado para la creación del código, encargado de crear el plano del croquis.

OpenGL. Herramienta usada para crear aplicaciones gráficas de alto rendimiento con portabilidad y flexibilidad.

3ds Max y Autodesk Maya. Software de modelado, animación y renderización 3D. Este programa se usó para la creación y modificación de los modelos que se usaron en el plano del croquis.

GIMP. Software de edición de imágenes, se usó para la creación de imágenes que se usaron como texturas en las geometrías del proyecto.

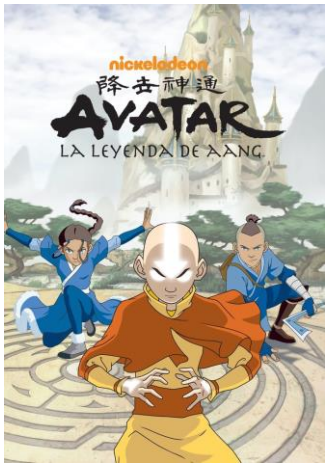
Aspecto técnico del desarrollo del proyecto

Control de versiones

Versión	***Fecha***	***Descripción***	***Autores***
1.0	Mayo 05 de 2024	Versión inicial donde se colocaron la mayor cantidad de modelos.	Todos los participantes del documento
2.0		Versión final	Todos los participantes del documento

Propuesta para el proyecto

Se eligieron los universos de “Avatar la leyenda de Aang” y “Hora de aventura”



Espacio a recrear

Parque ecológico de xochimilco

La idea es dividir el parque en dos mitades, una que está ambientada en el universo de “hora de aventura” y la otra mitad en el de “Avatar la leyenda de Aang”

Elementos utilizados en el plano

- Un río.
- Árboles de algodón de azúcar.
- Avatar de “jake el perro”.
- Barco de madera roto.
- Carretas de coliflor.
- Casas de avatar.
- Casas de “tronquitos”.
- Lirios.
- Muralla.
- Pasto.
- Senderos.
- Pulpo.
- Trajineras.
- Casa del árbol.

Animaciones utilizadas en el entorno.

- Movimiento del agua del río
- Movimiento de la trajinera
- Apertura del las puertas de la muralla
- Movimiento del pulpo flotando en el agua
- Movimiento de los brazos de “jake el perro”

Proceso y adaptación de modelos.

Se usaron principalmente modelos modificados o creados por 3ds Max y Maya, dando como resultado una amplia gama de modelos. Dado que se añadieron modelos de fauna, edificios, vehículos, flora y el avatar.

Para algunos modelos solo fue cargarlos, mientras que otros fue modelarlos desde cero, para después añadir las transformaciones necesarias para ajustarlos al entorno.

Rio

Para realizar este modelo solo se creó una superficie cuadrada dentro del software de modelado, para después añadirle la textura de agua.

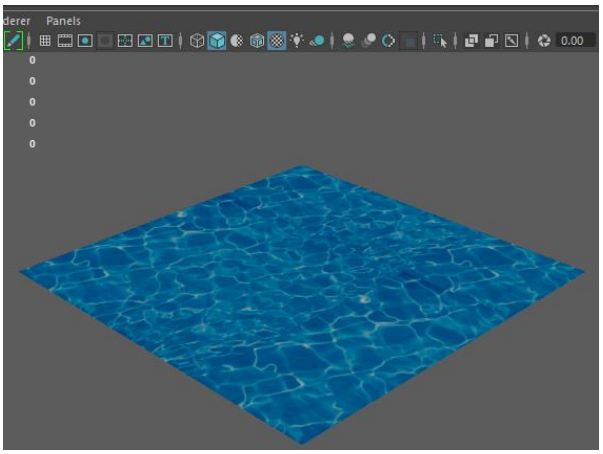


Imagen 01. Modelo del agua dentro del software de modelado

Para lograr la animación de movimiento en el agua se realizó mediante shaders, dentro del archivo anim.vs se usó el código mostrado en la imagen de abajo.

```
#version 330 core
layout (location = 0) in vec3 aPos;
layout (location = 1) in vec3 aNormal;
layout (location = 2) in vec2 aTexCoords;

const float amplitude = 1.0;
const float frequency = 0.01; // Reducir la frecuencia para hacer el movimiento más lento
const float PI = 3.14159;
const float noiseScale = 0.1; // Ajusta este valor según la intensidad deseada del ruido
const float noiseSpeed = 1.0; // Ajusta este valor para controlar la velocidad del ruido

out vec2 TexCoords;

uniform mat4 model;
uniform mat4 view;
uniform mat4 projection;
uniform float time;

float random(vec2 st) {
    return fract(sin(dot(st.xy, vec2(12.9898,78.233))) * 43758.5453123);
}

void main()
{
    float distance = length(aPos);
    float wave = amplitude * sin(-PI * distance * frequency + time);
    float noise = noiseScale * (random(aPos.xy) - 0.5); // Genera un valor de ruido entre -0.5 y 0.5
    float effect = wave + noise;
    gl_Position = projection * view * model * vec4(aPos.x + effect, aPos.y, aPos.z, 1);
    TexCoords = vec2(aTexCoords.x, aTexCoords.y);
}
```

Imagen 02. Código para la animación del agua

El shader crea un efecto de olas del mar al modificar las posiciones de los vértices en base a una combinación de la función sinusoidal (para las olas regulares) y un ruido aleatorio (para añadir variación e irregularidad). La función sinusoidal asegura que las olas oscilan suavemente con el tiempo, mientras que el ruido añade una apariencia más natural y desordenada.

Para realizar la animación dentro del archivo main, se realiza de la siguiente manera.

```
// Carga y dibujo de la animacion de movimiento del agua
Anim01.Use();
tiempo = glfwGetTime() * speed;
modelLoc = glGetUniformLocation(Anim01.Program, "model");
viewLoc = glGetUniformLocation(Anim01.Program, "view");
projLoc = glGetUniformLocation(Anim01.Program, "projection");
glUniformMatrix4fv(viewLoc, 1, GL_FALSE, glm::value_ptr(view));
glUniformMatrix4fv(projLoc, 1, GL_FALSE, glm::value_ptr(projection));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
model = glm::mat4(1);
glUniform1f(glGetUniformLocation(Anim01.Program, "time"), tiempo);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Mar.Draw(Anim01);
Mar2.Draw(Anim01);
glBindVertexArray(0);
```

Imagen 03. Código para realizar la animación dentro de OpenGL

Árboles de algodón de azúcar

Este modelo se generó dentro de la página "https://lumalabs.ai/genie?view=create", ya que la forma de un algodón de azúcar para la plataforma del árbol era algo complicada de moldear de cero.

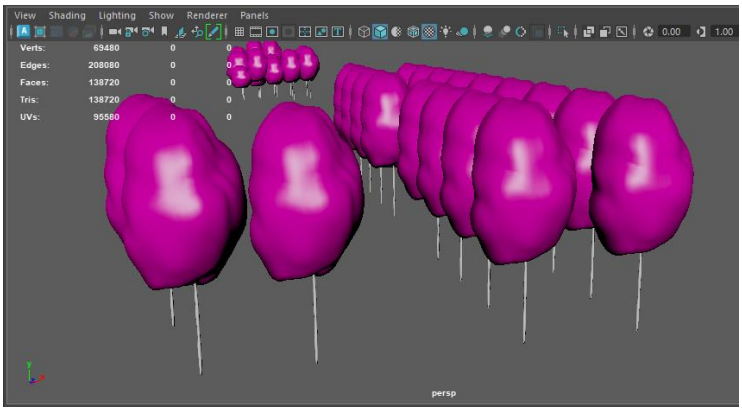


Imagen 04. Modelo del bosque de árboles de algodón de azúcar

Avatar de "jake el perro"

El avatar dentro del ambiente se instancian como un modelo jerárquico para para poder añadir las rotaciones en los brazos para que se más fácil el añadir la animación que se genera cuando este elemento entra en movimiento dando el efecto de movimiento al caminar.

Imagen 05. Avatar del personaje "jake el perro".

Acciones de teclado y mouse

Para las acciones que se pueden realizar con el teclado, las podemos encontrar en el apartado de *Window.cpp* ya que aquí se procesan todas las acciones recibidas del teclado.

- Movimiento personaje:
  - Avanzar: *W*
  - Retroceder: *S*
  - Izquierda: *A*
  - Derecha: *D*

Barco de madera roto

Al igual que el modelo del árbol de algodon de azucar, este tambien se genero dentro de la pagina "https://lumalabs.ai/genie?view=create"

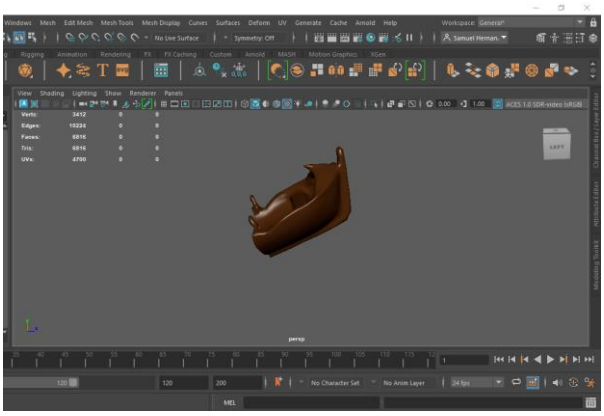


Imagen 06. Modelo del barco roto

Carretas de coliflor

Modelo diseñado de cero, y texturizado en GIMP, el estilo de texturizado es más detallado debido al universo al que pertenece.

Imagen 07. Modelo de las carretas de coliflores.

Casas de avatar

Este modelo al igual que el anterior fue diseñado de cero y texturizado en GIMP siguiendo el mismo estilo.

Imagen 08. Modelo de las casas pertenecientes al universo de avatar.

Casas de "tronquitos"

Modelo creado desde cero, algo que no mencione en el anterior, es que las casas se modelaron para que la camara tambien pueda ver dentro de ellas, es decir, no son solo cubos vacíos, en este caso, se decidió darle una textura interior diferente a la exterior de la casa.

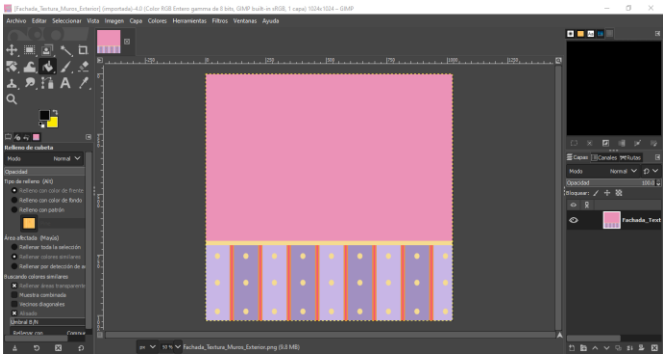


Imagen 09. Textura en GIMP de los muros de la casa.

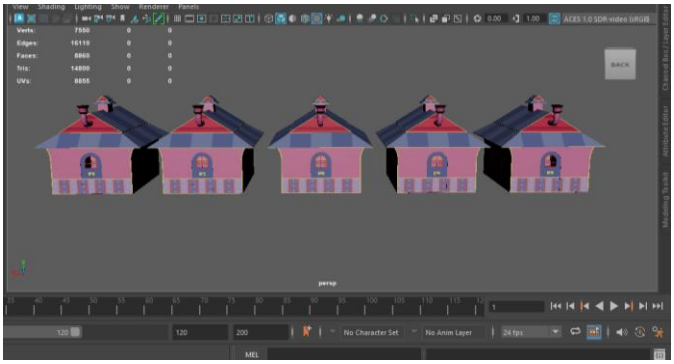


Imagen 11. Modelo texturizado en Maya

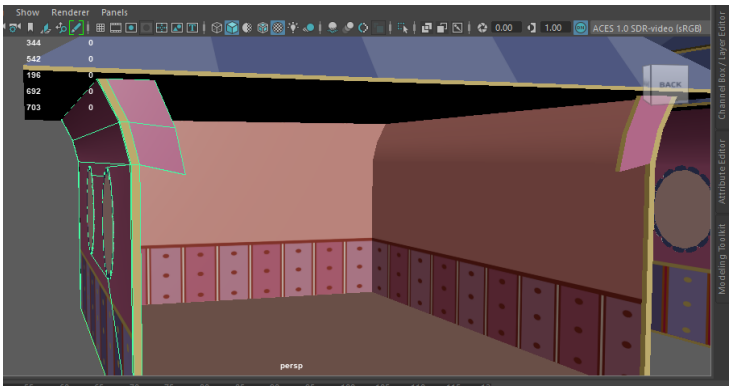


Imagen 12. Interior de la casa.

Lirios.

Modelado creado desde cero y texturizado mediante GIMP, para su importación en OpenGL se decidió escalar su tamaño para tener el mismo tamaño que el de los árboles de algodón de azúcar, para que el parque se viera más “vivo” por así decirlo.

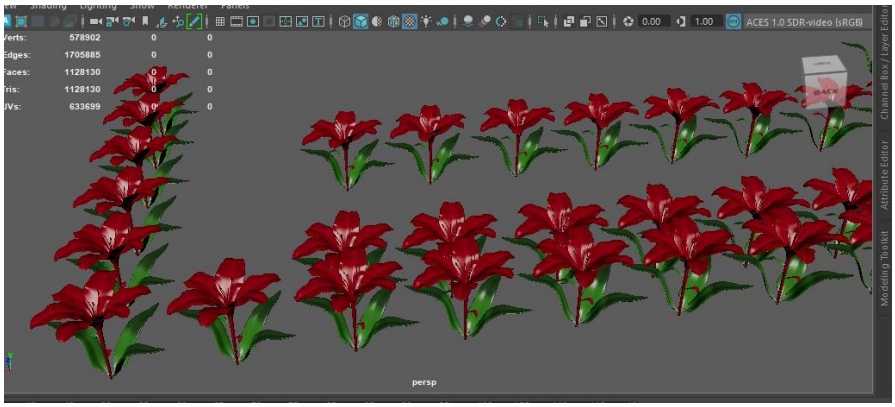


Imagen 13. Modelo del lirio en Maya.

Muralla.

Modelado creado desde cero y texturizado mediante GIMP, para su importación en OpenGL se decidió escalar su tamaño para funcionar como la puerta principal del parque.

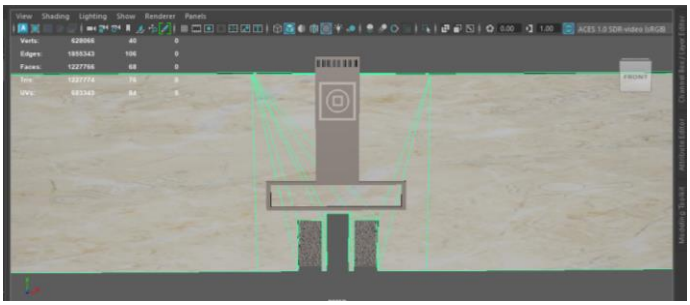


Imagen 14. Modelo de la muralla.

Pasto y senderos.

Las texturas de estos dos modelos se descargaron de internet para tener un mejor acabado lo único que se realizó en gimp fue escalar la imagen para que sus dimensiones sean cuadradas.

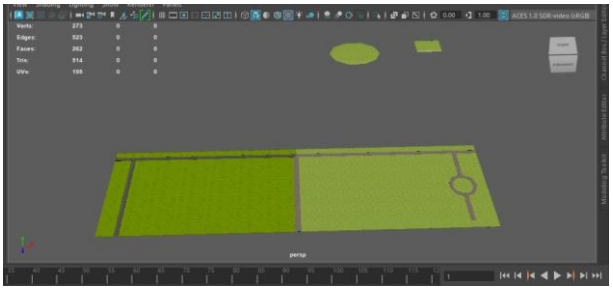


Imagen 15. Modelo del pasto y senderos del parque.

Pulpo.

Se realizó el modelado en 3dsmax y su texturizado en GIMP.

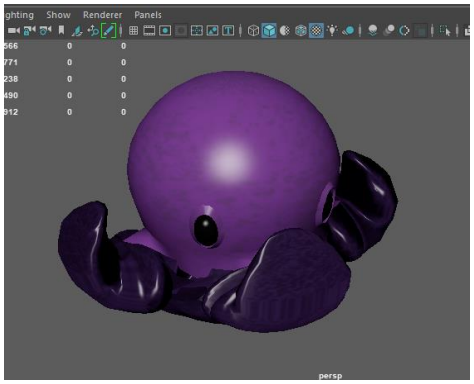


Imagen 16. Modelo del pulpo.

La animación del movimiento del pulpo sobre el agua se realizó mediante shaders, dentro del archivo anim2.vs se usó el código mostrado en la imagen de abajo.

```
#version 330 core
layout (location = 0) in vec3 aPos;
layout (location = 1) in vec3 aNormal;
layout (location = 2) in vec2 aTexCoords;

const float amplitude = 0.3; // Amplitud del movimiento
const float frequency = 0.1; // Frecuencia del movimiento
const float offset = 1.0; // Desplazamiento lateral
const float speed = 4.0; // Velocidad del movimiento

out vec2 TexCoords;

uniform mat4 model;
uniform mat4 view;
uniform mat4 projection;
uniform float time;

void main()
{
    float wave = amplitude * sin(frequency * time * speed); // Función sinusoidal modificada
    vec3 newPosition = aPos + vec3(0.0, 0.0, wave + offset); // Aplicar el desplazamiento lateral

    gl_Position = projection * view * model * vec4(newPosition, 1.0);
    TexCoords = aTexCoords;
}
```

Imagen 17. código para la animación del pulpo.

Una vez definido el vertex shader con las constantes de apoyo dentro de la función y las variables uniformes, para que el movimiento del pulpo, hay que llamarlo en el código fuente, y dibujar y cargar el modelo del pulpo con su respectiva textura.

```
// Carga del pulpo en movimiento
Anim02.Use();
tiempo = glfwGetTime() * speed;
modelLoc = glGetUniformLocation(Anim02.Program, "model");
viewLoc = glGetUniformLocation(Anim02.Program, "view");
projLoc = glGetUniformLocation(Anim02.Program, "projection");
glUniformMatrix4fv(viewLoc, 1, GL_FALSE, glm::value_ptr(view));
glUniformMatrix4fv(projLoc, 1, GL_FALSE, glm::value_ptr(projection));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
model = glm::mat4(1);
glUniform1f(glGetUniformLocation(Anim02.Program, "time"), tiempo);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
//PurplePentapus.Draw(Anim02);
```

Imagen 18. Código para realizar la animación dentro de OpenGL

Retomando el funcionamiento de la animación, este utiliza la función sinusoidal para generar un valor que cambia suavemente con el tiempo, creando así el efecto de movimiento ondulatorio.

La forma básica de la función sinusoidal es:

$y(t) = A * sen(B * t + C)$

Donde:

- A es la amplitud, que determina la altura de la onda.
- B es la frecuencia, que controla la rapidez de oscilación.
- t es el tiempo.
- C es la fase, que desplaza horizontalmente la onda.

En el contexto del shader:

- time se utiliza como el valor de t.
- amplitude corresponde a A, controlando cuánto se mueve el vértice hacia arriba y hacia abajo.

- frequency corresponde a B, controlando la rapidez con la que se completa un ciclo de oscilación.
- Speed corresponde a C en el shader.

En palabras sencillas, la función sinusoidal produce un valor que cambia suavemente entre amplitud negativa y amplitud positiva a medida que avanza el tiempo, creando así el efecto de movimiento de las olas de mar.

Trajineras.

Modelado creado desde cero y texturizado mediante GIMP, para su importación en OpenGL, una aclaración este es el modelo con más detalles, así que puede causar que al ejecutar el archivo, este se tarde en ejecutarse.

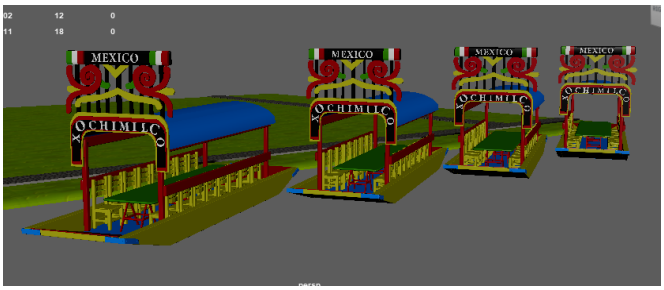


Imagen 19. Modelo de la trajinera

La sección de código que aparece en la imagen de abajo se encarga de realizar la animación de desplazamiento de una trajinera.

```
////INICIO SECCION ANIMACION :: Trajinera Moviendose
if (animTrajinera == true) { // Si anim02 == true realiza lo siguiente
    if (desplazaTrajineraBandera == true) {
        desplazaTrajinera += 0.01f * glfwGetTime();
        if (desplazaTrajinera >= 20.0f) {
            desplazaTrajinera = 20.0f;
            desplazaTrajineraBandera = false;
            animTrajinera = false;
            animTrajineraRev = false;
        }
    }
}
if (animTrajineraRev == true) { // Si anim02 == true realiza lo siguiente
    if (desplazaTrajineraBandera == false) {
        desplazaTrajinera -= 0.01f * glfwGetTime();
        if (desplazaTrajinera <= 0.0f) {
            desplazaTrajinera = 0.0f;
        }
    }
}
////FIN SECCION ANIMACION :: Trajinera moviendose
```

Imagen 20. Código para realizar la animación dentro de OpenGL

Casa del árbol.

Este modelo, al igual que todos los anteriores fue creado desde cero, su texturas fueron diseñadas en GIMP, y su modelado en Maya.

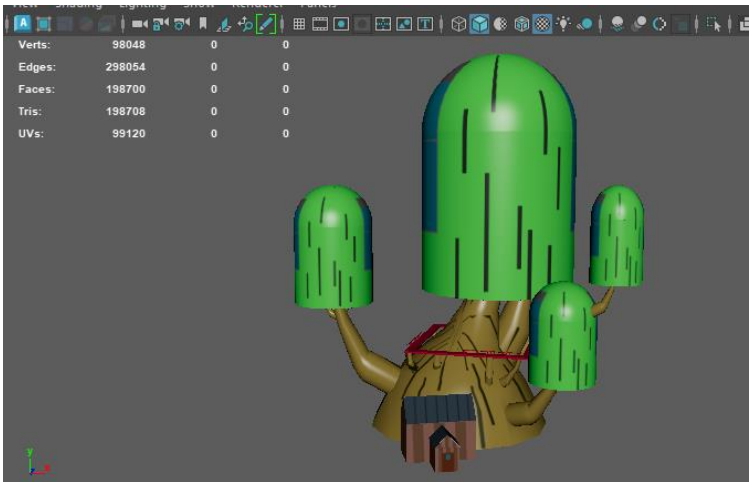


Imagen 21. Modelo de la casa del árbol

Conclusión.

Este Proyecto en definitive puso a prueba todos los conocimientos adquiridos a lo largo del curso, podemos decir que la parte que mejor manejamos fue el modelado de los objetos ya que inicialmente el software MAYA y 3DSMax eran algo complicados pero con la práctica se fue facilitando, es más para antes de la entrega del código del proyecto se decidio optimizar los modelos de los objetos lirios y trajineras, ya que inicialmente estaban mal realizados, pero esto se realizó en la menos de la mitad de tiempo de lo que me tomo inicialmente modelarlos, otra punto a destacar es que en las texturas no se usaron las imágenes de referencia de papel tapiz, sino que se realizó a mano (digital pero se entiende a que nos referimos) cada una de las texturas, cada detalle, si fue algo tedioso ya que eso si requiere tiempo, pero el resultado bajo nuestro punto de vista es aceptable, en cuanto a la parte de programación e iluminación, el tener que dibujar y cargar los modelos no presento dificultad alguna ya que si los modelos están bien modelados y escalados no tienen por qué presentar algún problema, por otro lado la iluminación debido a los universos seleccionados las sombras y tonos oscuros no estaban casi presentes así que no hubo mucha dificultad a la hora de ambientar los objetos y las estructuras, para finalizar, la parte que más tiempo tomo fue la animación, si bien la mayoría son animaciones sencillas la lógica detrás de su codificación tomo algo de tiempo descifrarla pero al final se logró, en especial el tratar de que las animaciones se reinicien la cantidad de veces que desee el usuario, sin más que decir, agradecemos este reto profesor, y si bien lo pudimos haber checo mejor, nos sentimos satisfecho con lo logrado ya que fue nuestro esfuerzo y dedicación los que permitieron finalizar este proyecto.

Referencias

Requisitos del sistema de Visual Studio 2022. (2023, Julio 26). Microsoft Learn. Retrieved Mayo 9, 2024, obtenido de: https://learn.microsoft.com/es-es/visualstudio/releases/2022/system-requirements

Modelo algodon azucar (2024, Mayo 9), obtenido de: https://lumalabs.ai/genie?view=create

Modelo algodon azucar, obtenido de: https://lumalabs.ai/genie?view=create