Universidad Nacional Autónoma de México
Facultad de Ingeniería
División de Ingeniería Eléctrica
Ingeniería en computación
Computación Gráfica e Interacción Humano Computadora

Manual Técnico

Número de cuenta: 317335930

Grupo: 12

Fecha de entrega: Jueves 25 de mayo del 2023

Semestre: 2023-2

Índice

Contenido

1-Objetivo

En la realización de este proyecto Prototipo, nuestro objetivo principal es realizar el modelado 3D de nuestra propuesta sobre un espacio del video juego de “south park” ambientado en el juego original de Nintendo 64, realizando el modelaje y texturizado de 7 objetos, realizando animaciones complejas y sencillas, creando un ambiente para el escenario y finalizando con la codificación de Open gl.

Con la realización de este proyecto Prototipo queremos saber si el trabajo realizado fue hecho con las indicaciones que se pidió, logrando un ambiente dinámico, con un diseño cartoon y que sea representativo de la serie de televisión, este será un prototipo de entrega y reafirmando los conocimientos que tuvimos en nuestro curso de Computación gráfica.

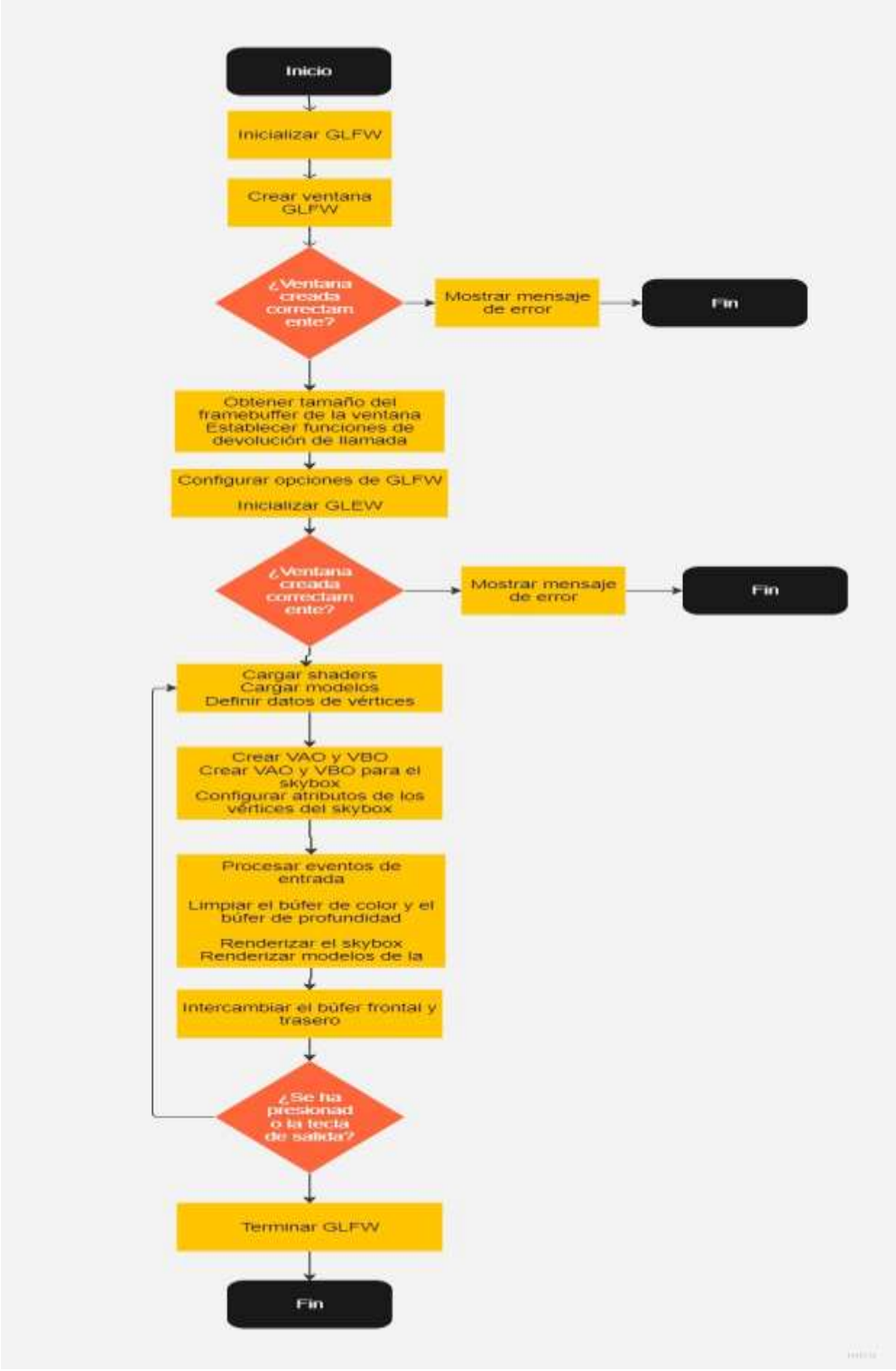
2-Investigación Preliminar

En nuestra investigación del proyecto, necesitamos primero saber sobre el modelado y ambiente que tenemos que realizar, nuestra propuesta fue hacer una casa del videojuego de “south park: Retaguardia en peligro” que estará inspirado su diseño de Nintendo 64, esto para que su modelado sea más carismático y sea parecido a una retrospectiva de cómo puede ser un juego 2D en 3D, utilizando una perspectiva diferente y un espacio dinámico.

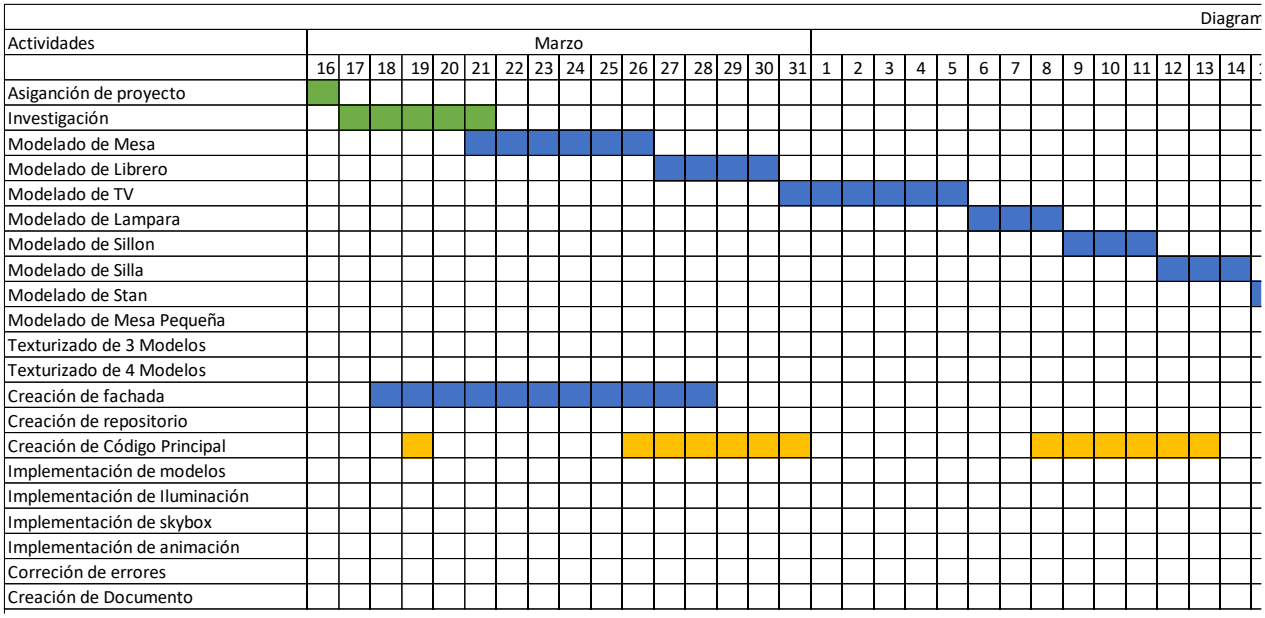
En primer inicio tomaremos el plan de utilizar una metodología de prototipo por el hecho de que es un proyecto pequeño y que queremos crear un prototipo antes de empezar un desarrollo a gran escala, esto nos permitirá como desarrolladoras comprender que se debe hacer y el funcionamiento, para descubrir fallos o cambios que se puedan implementar o consideraciones que también podrían cambiar, esto para permitir mejorar nuestro trabajo final.

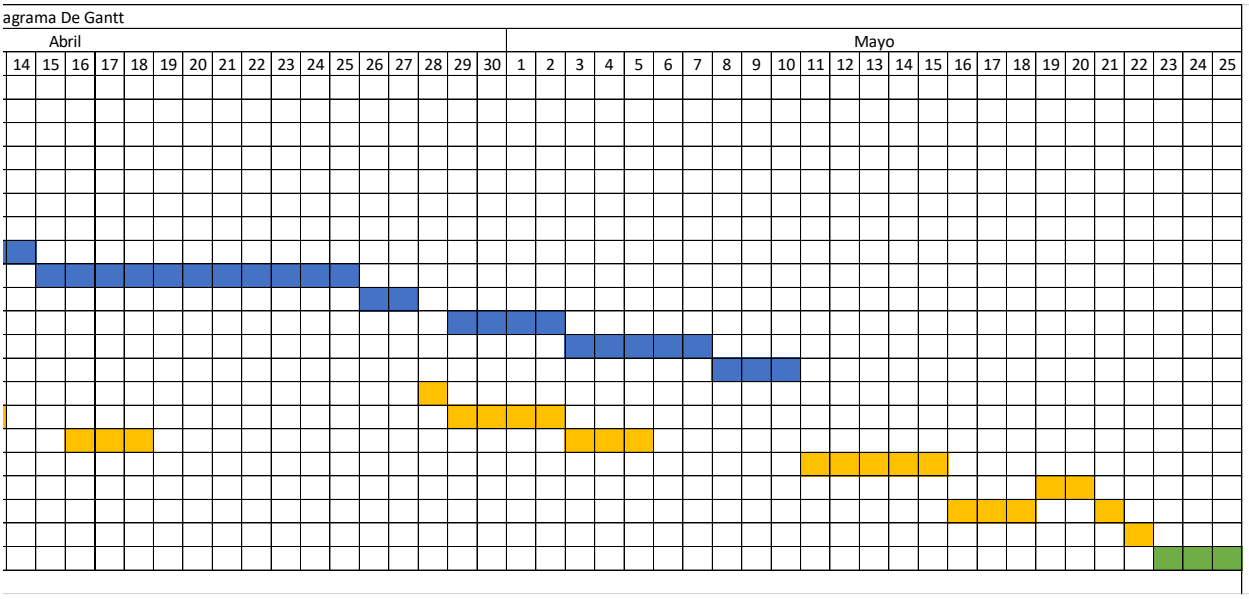
Iniciando con una investigación del tipo de ambiente, supimos que teníamos que hacer materiales cartoon, este tipo de texturizado tendría que ser mate para que pudiéramos tener ese estilo que buscamos y agregar una iluminación tenue que hiciera parecer que estamos dentro de la casa. Por último, crear nuestro código base para que este implementado todos los modelos, iluminación, skybox, animaciones sencillas, animaciones complejas y control de cámara.

2.1-Digrama de flujo



2.2- Diagrama de Gantt





3- Diseño Conceptual

La facha que tomamos de referencia fue la casa del personaje Stan la cual es representativa de la serie.

*https://southpark.fandom.com/wiki/Marsh_Residence

De los objetos los tomamos del videojuego “South park: La vara de la verdad“ ya que los muebles estaran ambientados y diseñados con el modelo de ese tiempo del videojuego y como podemos observar los modelos están en una imagen 2D, entonces tomaremos referencia a los modelos del juego de Nintendo 64 para modelar al tipo de ese ambiente de gráficos de videojuegos.

“South park: La vara de la verdad“

Mostraremos algunas imágenes del juego de Nintendo 64 para dar referencia de como nos guiamos.

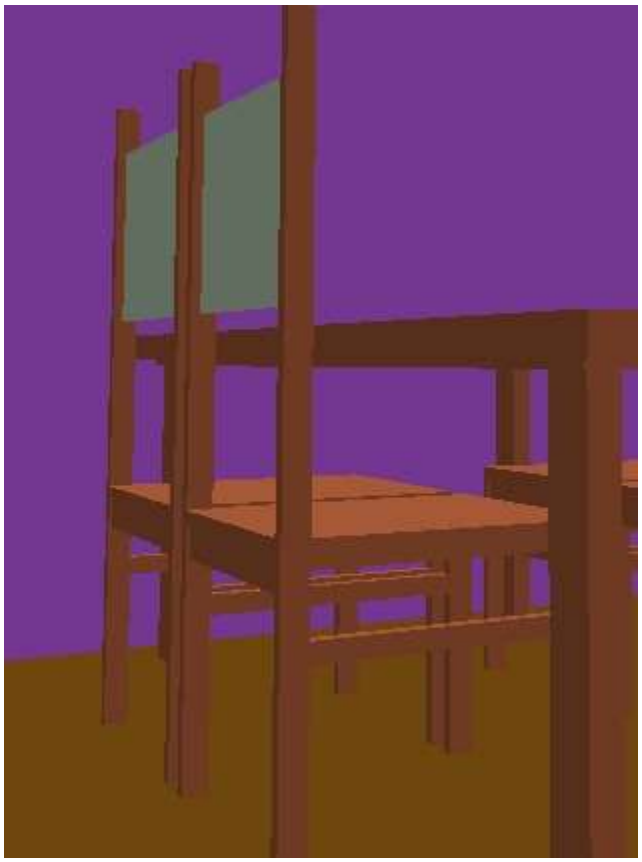
*https://xtremeretro.com/south-park/

*https://drew1440.com/2021/11/09/south-park/

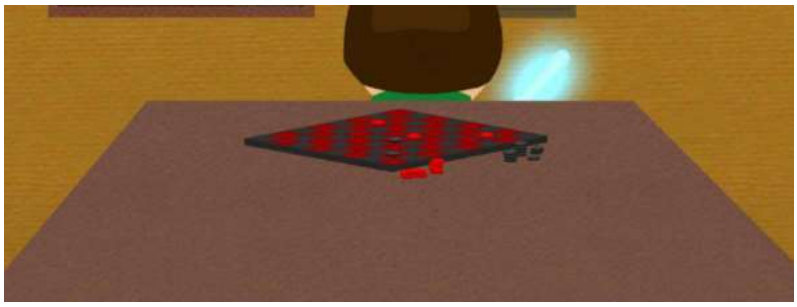
3.1-Modelos Realizados

Los modelos que propusimos mostraremos una imagen de referencia y una imagen de como quedaron.

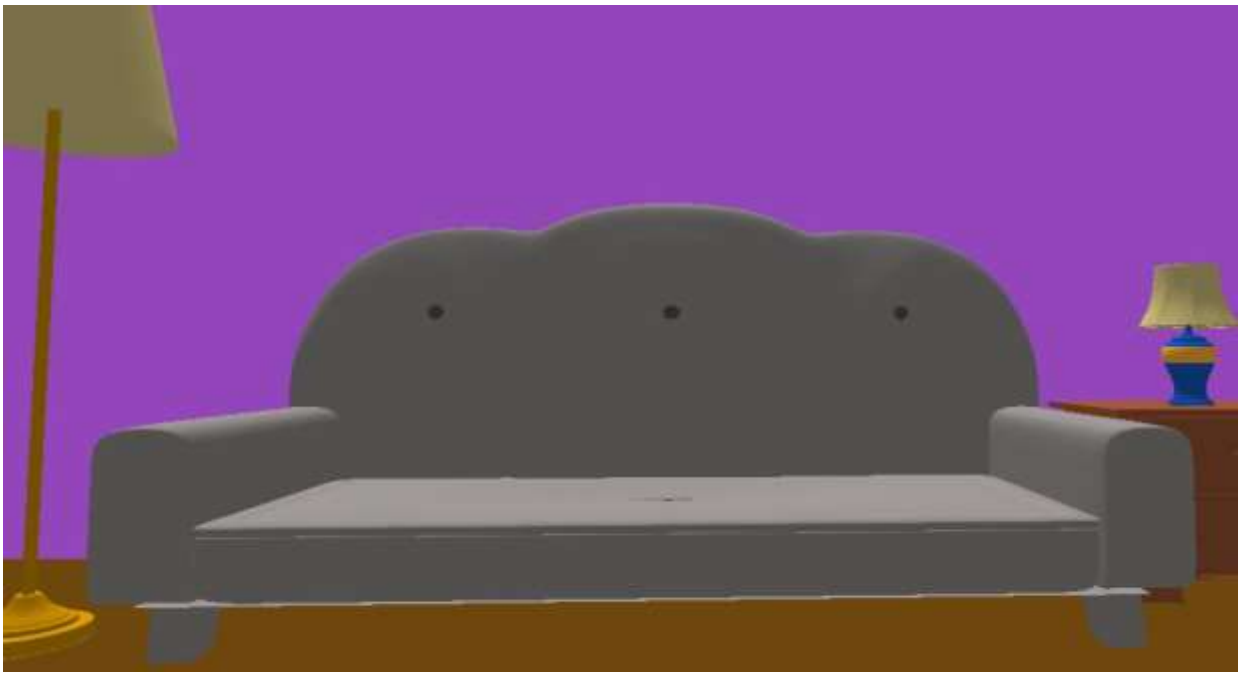
Silla:



Mesa:

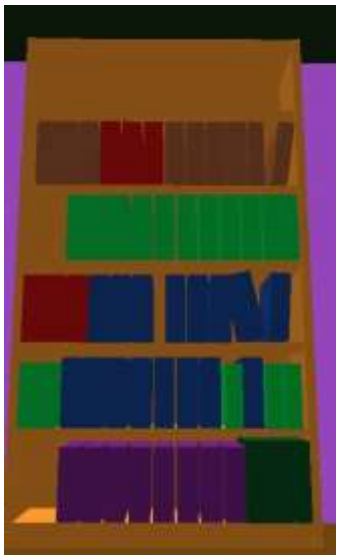


Sillón:



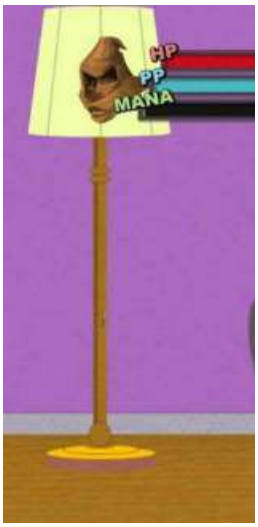
Mesa Pequeña:

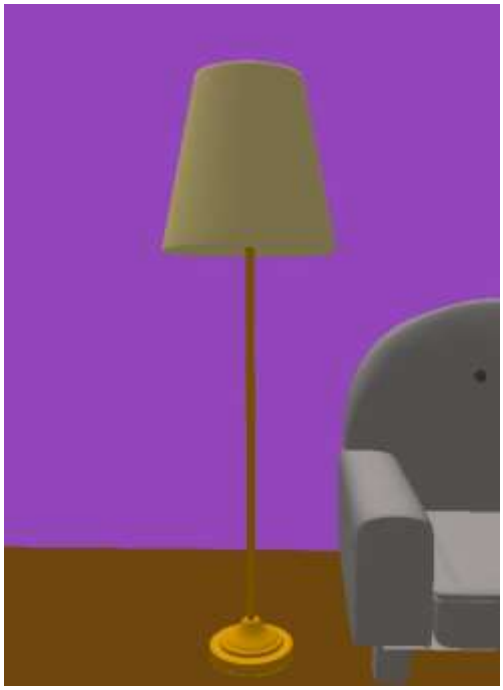
Librero:



TV:

Lampara:





Como bonus modelamos un personaje de la serie y el personaje que vive en la casa:

Para la fachada creamos la casa propuesta inspirada en el juego y sin salirnos del modelo propuesto y que tuviera la similitud con lo propuesto, además de agregar unos arbustos para la realización de una animación:

Para finalizar, tomaremos imágenes del modelo final con cada objeto y fachada:

3.2-Alcances del proyecto

Con la realización de los modelos empezaremos con una explicación de un satisfecho trabajo, ya que se logro una gran similitud con los diseños propuestos y logramos con los texturizados el diseño que estábamos buscando cartoon, logrando un trabajo de modelado y texturizado excepcional.

Para el modelado de la casa tuvimos visión de la casa del juego, pero enfocarnos en la casa que se propuso, guiándonos en los detalles que tienen y haciendo que interior de la casa sea lo mas parecido en el acodo de los muebles y además logrando ese ambiente de sala de estar.

Para el modelo de exterior del skybox, logramos conseguir las 6 imágenes originales del videojuego, solo tuvimos que lograr el acodo correcto y además de convertir los archivos, logramos encontrar también una textura para la nieve que sea un poco cartoon de misma forma, logrando que sea lo mas parecido a ese estilo de videojuego y también el modelado de los arbustos que son de ayuda para la realización de una animación.

Para el caso de la iluminación tomamos una iluminación que no sea tan brillante pero tampoco que sea tan apagada, esto principalmente para utilizarlo con el ambiente de afuera y el ambiente de la sala y que no se viera tan brillantes nuestros objetos y perdieran este toque de cartoon, además de que por lo visto en un documento que buscamos del videojuego, ellos no tomaron mucha iluminación en los interiores de la casa, ya que esto producía algo de bajo rendimiento al juego, entonces tomamos su idea y utilizamos el concepto de un punto que ayude a la luz ni tan brillan y ni tan apagado.

4-Selección de herramientas

-Visual studio: Utilizada principalmente para desarrollar aplicaciones, tanto de escritorio como web y móviles. Es una de las herramientas más populares en la industria del desarrollo de software y ofrece un conjunto completo de características y herramientas que facilitan el proceso de desarrollo.

-Maya: Es un software de animación, modelado y renderizado 3D ampliamente utilizado en la industria del entretenimiento, el diseño y la visualización arquitectónica. Desarrollado por Autodesk, Maya ofrece un conjunto de herramientas avanzadas que permiten a los artistas y diseñadores crear contenido digital de alta calidad.

-Github: Es una plataforma web que ofrece servicios de alojamiento de repositorios de código fuente y herramientas de colaboración para desarrolladores de software. Es

ampliamente utilizado en la comunidad de desarrollo de software y proporciona un entorno centralizado para el almacenamiento, seguimiento y gestión de versiones de proyectos de código abierto y privado

-Github Desktop: Es una aplicación de escritorio que proporciona una interfaz gráfica de usuario para interactuar con repositorios alojados en GitHub. Permite a los desarrolladores clonar repositorios, realizar cambios en el código, crear ramas (branches) y enviar cambios al repositorio remoto, todo ello sin tener que usar comandos de línea de comandos. Es una herramienta útil para aquellos que prefieren una interfaz gráfica intuitiva y simplificada para trabajar con Git y GitHub.

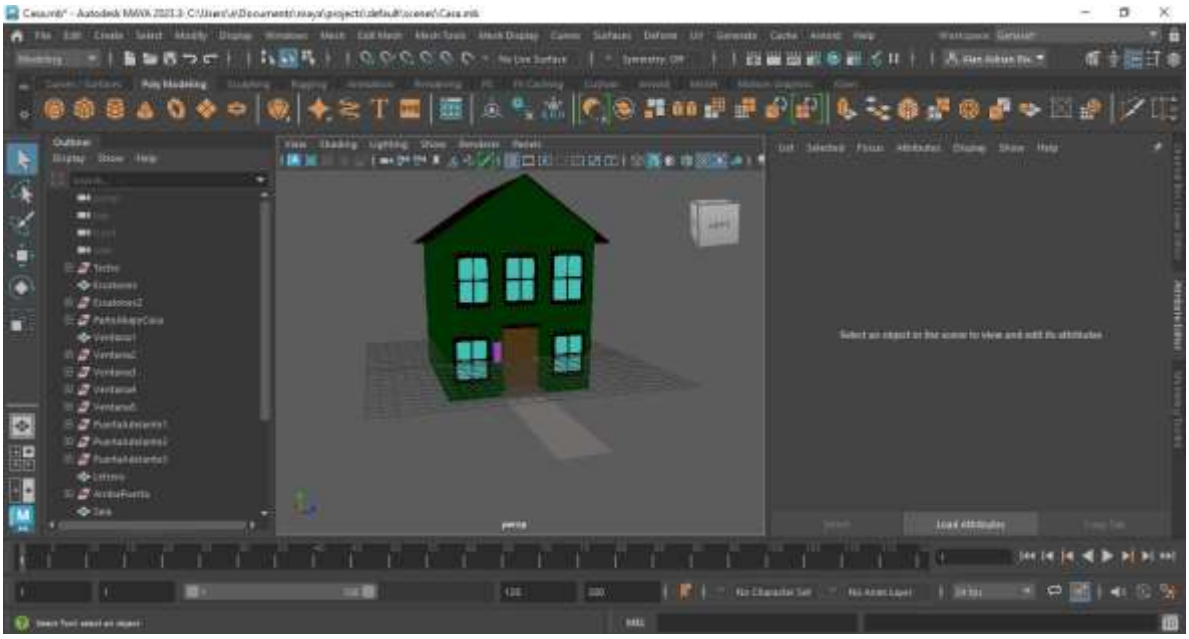
-C++: Es un lenguaje de programación de propósito general que se utiliza ampliamente en el desarrollo de software. Fue desarrollado a partir del lenguaje C original con el objetivo de agregar características de programación orientada a objetos y facilitar la escritura de programas más complejos y eficientes.

-GIMP: es un software de edición de imágenes de código abierto y gratuito que ofrece un amplio conjunto de herramientas para crear y editar gráficos digitales. Es una alternativa popular y potente a programas comerciales de edición de imágenes, como Adobe Photoshop.

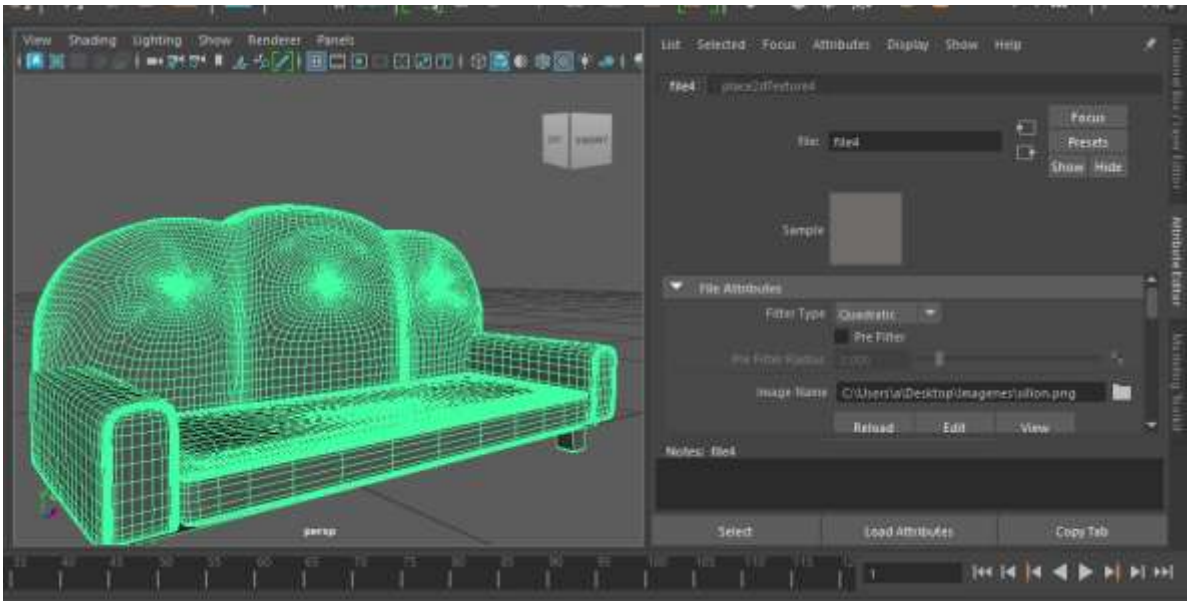
5-Desarrollo del prototipo

Al tener nuestro enfoque de que se tenía que realizar, tuvimos que poner en marcha primero en el enfoque de investigar primero un modelo que fuera sencillo y que fuera un gusto para crear y no llevar trabajo extra.

Realizando los modelados empezamos primero familiarizando con maya, buscando comandos y viendo las diferentes herramientas que se pueden crear, para empezar con nuestros modelos tuvimos que hacer un modelo por semana para ver si tuviera problema alguno o viendo imperfectos que tuviera nuestro modelo.



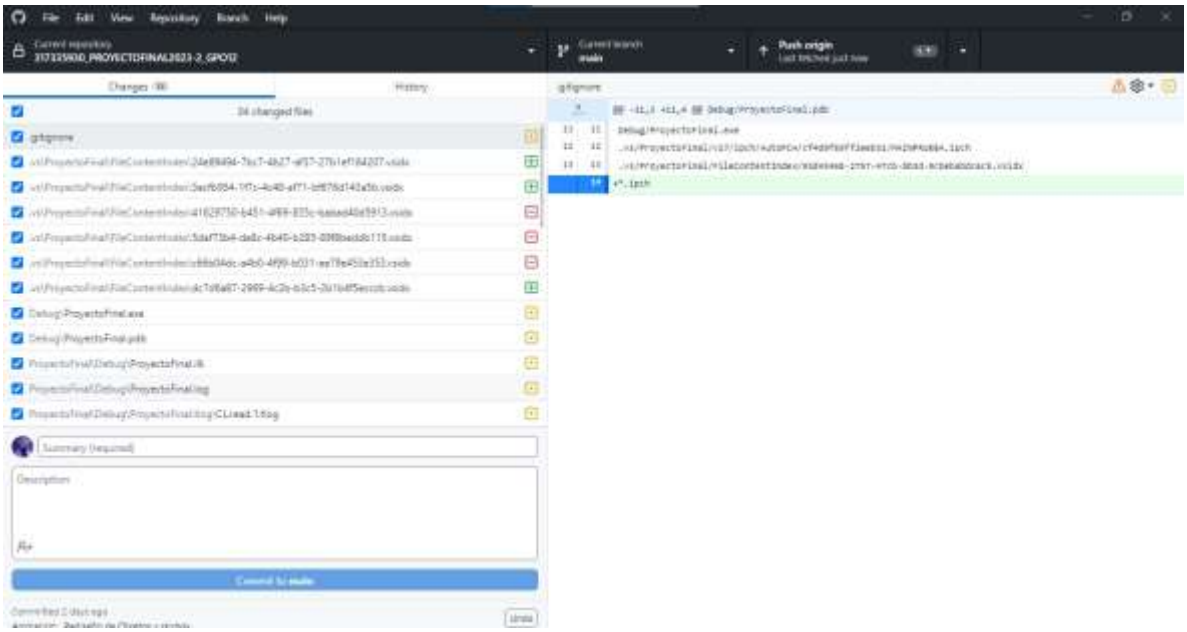
Maya es fácil de comprender después de modelar ya bastantes modelos, llevando las texturas tuvimos que realizar un texturizado en lambet y con la ayuda de GIMP realizamos el texturizado del color asignando diferentes colores para cada objeto y proponiendo unos colores generales para todo objeto y no generar muchas imágenes similares.



Con la creación de la fachada, realizamos diferentes formas, primero realizamos la parte de arriba ya que esta era la más difícil por el techo que tenia forma triangulas, para la parte de abajo solo tuvimos que hacer un recorte en la parte de la puerta y realizamos los aditamentos de la parte exterior como ejemplo fueron las ventanas que buscamos ese tono de animación y el diseño de la puerta de misma manera. En la parte interior se tuvo que agregar un cuadrado extra, ya que nuestra sala tendría un diferente color que el

exterior y para el suelo tuvimos que subirlo un poco mas para implementar ese diseño de escalones en la entrada agregando un camino.

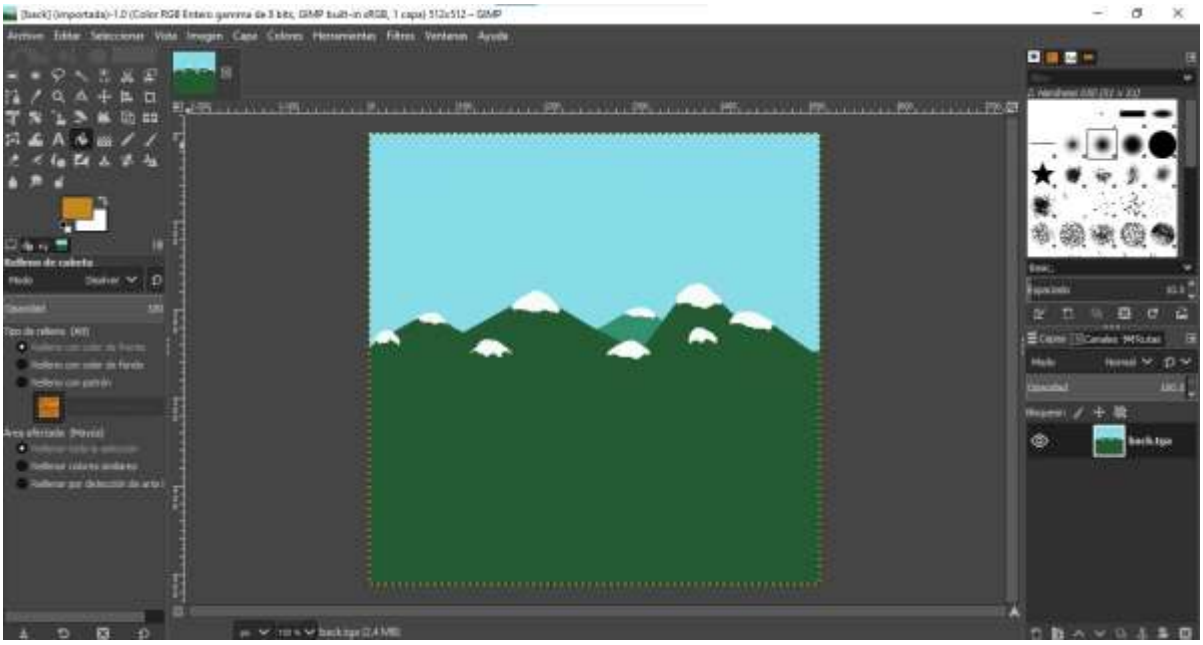
En la creación del repositorio se realizo una copia del que esta implementado en el grupo general de la clase, para ello tuvimos que crear un fork para que tuviéramos una copia y realizar las diferentes modificaciones, en GitHub Desktop realizaremos los diferentes commit para que esto registre todos los cambios que se hicieron, después podremos subirlos y que se registren en nuestro repositorio.



Para la implementación de los modelos tuvimos que cargar primero a la carpeta modelos, donde tenemos el .obj y las texturas, después en la implementación de nuestro código los crearemos y podremos visualizar.

Nombre	Fecha de modificación	Tipo	Tamaño
Arbusto	21/05/2023 04:51 p. m.	Carpeta de archivos	
Carro	17/05/2023 06:27 p. m.	Carpeta de archivos	
Casa	21/05/2023 03:37 a. m.	Carpeta de archivos	
Lampara	13/05/2023 02:07 p. m.	Carpeta de archivos	
Librero	21/05/2023 03:34 a. m.	Carpeta de archivos	
Mesa	13/05/2023 04:17 p. m.	Carpeta de archivos	
MesaPequeña	20/05/2023 07:11 p. m.	Carpeta de archivos	
New Folder	18/05/2023 10:19 p. m.	Carpeta de archivos	
Piso	21/05/2023 02:39 a. m.	Carpeta de archivos	
Silla	13/05/2023 05:18 p. m.	Carpeta de archivos	
Sillon	13/05/2023 04:11 p. m.	Carpeta de archivos	
Stan	21/05/2023 03:43 a. m.	Carpeta de archivos	
Televisor	20/05/2023 02:12 a. m.	Carpeta de archivos	

La implementación del skybox, tuvimos que buscar en diferentes paginas el skybox original de la serie, logrando encontrar tuvimos que convertir el tipo de archivo a jpg, cambiar el nombre de la imagen y exportar de tipo .tga, después solo colocar en el código en la parte de skybox y ver que no tuviera un problema.



6- Limitaciones

En general tuvimos algunas limitaciones en el proyecto que por cuestiones de tiempo creemos que podríamos mejorar en ese ambiente, teniendo en cuenta que la realización de la iluminación y de la animación es en donde más tuvimos problema y fue un problema que tuvimos que resolver, pero nos llevó mucho tiempo en resolver estas dudas y problemas.

Por estas limitaciones tuvimos que crear una iluminación básica que tuviera el ambienta que estamos buscando, pero solo logrando que la iluminación fuera básica y sin gran detalle, por otro lado, las animaciones se solicitaron que fueran 5 (3 animaciones sencillas y 2 animaciones complejas) por ello solo se pudo realizar 3 animaciones sencillas (puerta, saludo y cajón) y una animación compleja (movimiento de arbusto)

Pero con estas limitaciones, podemos decir que el prototipo que estamos entregando es un trabajo que estamos orgullos y que creemos que aportamos todos los conocimientos.

7-Documentación

Dentro del documento inicial de prueba tendremos que estaba bastante avanzado y que ya contiene muchos aditamentos que nos ayudara en la implementación de nuestros códigos que proporcionaremos nosotros, por ello en la documentación explicaremos sobre que hace cada parte que agregamos nosotros y con que propósito fue que lo implementamos.

```cpp
#include <iostream>
#include <cmath>

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

// GLFW
#include <GLFW/glfw3.h>

// Other Libs
#include "stb_image.h"

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

//Load Models
#include "SOIL2/SOIL2.h"


// Other includes
#include "Shader.h"
#include "Camera.h"
#include "Model.h"
#include "Texture.h"
```

Esta sección incluye varias bibliotecas y archivos de encabezado necesarios para el código.

* - Las bibliotecas y archivos de encabezado incluidos son:

* - iostream: Proporciona operaciones básicas de entrada/salida.

* - cmath: Proporciona funciones matemáticas como sqrt, sin, cos, etc.

* - GLEW: Biblioteca de envoltura de extensiones de OpenGL para gestionar extensiones de OpenGL.

* - GLFW: Biblioteca para crear y gestionar ventanas, contextos OpenGL y manejar la entrada del usuario.

* - stb_image: Biblioteca para cargar diversos formatos de imagen.

* - glm: Biblioteca matemática para aplicaciones de OpenGL, que incluye operaciones de vectores y matrices.

* - SOIL2: Biblioteca para cargar archivos de imagen como texturas.

* - Shader: Archivo de encabezado personalizado para gestionar shaders.

* - Camera: Archivo de encabezado personalizado para manejar operaciones de cámara.

* - Model: Archivo de encabezado personalizado para cargar y renderizar modelos 3D.

* - Texture: Archivo de encabezado personalizado para gestionar texturas.

```
// Function prototypes
void KeyCallback(GLFWwindow* window, int key, int scancode, int action, int mode);
void MouseCallback(GLFWwindow* window, double xPos, double yPos);
void DoMovement();

// Window dimensions
const GLuint WIDTH = 1350, HEIGHT = 900;
int SCREEN_WIDTH, SCREEN_HEIGHT;

// Camera
Camera  camera(glm::vec3(-13.0f, 5.0f, 0.0f));
GLfloat lastX = WIDTH / 2.0;
GLfloat lastY = HEIGHT / 2.0;
bool keys[1024];
bool firstMouse = true;

//Movimientos
float rotStan = 0;
float rotPuerta = 0;
float  movCajon = 0.0f;
bool anim = false, anim2 = false, anim3 = false;
bool CajonAbierto = false, CajonAbrir = true, CajonCerrar = false;
float tran = 0.0f;
bool Open = true;
float tiempo;
float speed = 0.0f;
```

Prototipos de funciones.

*- KeyCallback: Callback de teclado que se llama cuando se presiona una tecla.
*- MouseCallback: Callback de mouse que se llama cuando se mueve el mouse.
*- DoMovement: Realiza los movimientos de la cámara según las teclas presionadas.

Dimensiones de la ventana.

*- Las constantes WIDTH y HEIGHT representan las dimensiones de la ventana en píxeles.
*- SCREEN_WIDTH y SCREEN_HEIGHT se utilizan para almacenar las dimensiones actuales de la pantalla.

Cámara.

*- La cámara se inicializa con una posición inicial y se utiliza para controlar la vista de la escena.
*- lastX y lastY se utilizan para almacenar las coordenadas del último posicionamiento del mouse.
*- keys es un arreglo de booleanos que indica qué teclas están actualmente presionadas.
*- firstMouse indica si esta es la primera vez que se mueve el mouse.

Movimientos.
Variables relacionadas con los movimientos de la escena.

*-rotStan: Rotación del personaje Stan
*-rotPuerta: Rotación de la puerta.
*-movCajon: Desplazamiento del cajón.
*-anim, anim2, anim3: Variables de animación.
*-CajonAbierto, CajonAbrir, CajonCerrar: Variables relacionadas con la apertura y cierre del cajón.
*-tran: Transparencia.
*-Open: Variable de apertura.
*-tiempo: Tiempo.
*-speed: Velocidad de movimiento.

```
// Positions of the point lights
glm::vec3 pointLightPositions[] = {
        glm::vec3(0.0f,0.0f, 0.0f),
        glm::vec3(0.0f,0.0f, 0.0f),
        glm::vec3(0.0f,0.0f,  0.0f),
        glm::vec3(0.0f,0.0f, 0.0f)
};


glm::vec3 Light1 = glm::vec3(0);


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

Posiciones de las luces puntuales.

*-El arreglo pointLightPositions contiene las posiciones en el espacio 3D de las luces puntuales.
*-Cada elemento del arreglo representa una luz puntual distinta.

Posición de la Luz1.

*-Light1 representa la posición de una luz específica en el espacio 3D.

Deltatime.

*-deltaTime almacena el tiempo transcurrido entre el fotograma actual y el último fotograma.
*-lastFrame guarda el tiempo del último fotograma.

```
int main()
{
        // Init GLFW
        glfwInit();


        // Create a GLFWwindow object that we can use for GLFW's functions
        GLFWwindow* window = glfwCreateWindow(WIDTH, HEIGHT, "Iluminacion 2", nullptr, nullptr);

        if (nullptr == window)
        {
                std::cout << "Failed to create GLFW window" << std::endl;
                glfwTerminate();

                return EXIT_FAILURE;
        }

        glfwMakeContextCurrent(window);

        glfwGetFramebufferSize(window, &SCREEN_WIDTH, &SCREEN_HEIGHT);

        // Set the required callback functions
        glfwSetKeyCallback(window, KeyCallback);
        glfwSetCursorPosCallback(window, MouseCallback);

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

Función principal del programa.

*-La función main es el punto de entrada del programa.
*-En esta función se realiza la inicialización de GLFW, se crea una ventana y se configuran las opciones necesarias.
*-También se establecen las funciones de devolución de llamada para el teclado y el mouse.
*-Por último, se establece el modo de entrada del mouse y se obtienen las dimensiones del framebuffer de la ventana.

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

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

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");
Shader SkyBoxshader("Shaders/SkyBox.vs", "Shaders/SkyBox.frag");
```

Inicialización de shaders.

*-Se crean objetos Shader utilizando los archivos de los shaders.
*-Cada objeto Shader representa un programa de shaders que se utilizará en la renderización.
*-Los shaders se cargan desde los archivos "lighting.vs", "lighting.frag", "lamp.vs", "lamp.frag",
*-"anim.vs", "anim.frag", "anim2.vs", "anim2.frag", "SkyBox.vs" y "SkyBox.frag".
*-Los objetos Shader se utilizarán posteriormente para renderizar los objetos de la escena.

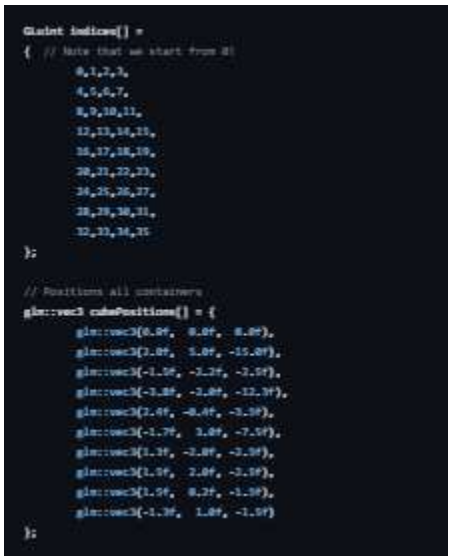
```
//Modelos utilizados
Model Casa((char*)"Models/Casa/Casa.obj");
Model Lampara((char*)"Models/Lampara/Lampara.obj");
Model Librero((char*)"Models/Librero/Librero.obj");
Model Mesa((char*)"Models/Mesa/Mesa.obj");
Model MesaPequeña((char*)"Models/MesaPequeña/Mesita.obj");
Model Cajon((char*)"Models/MesaPequeña/Cajon.obj");
Model Silla((char*)"Models/Silla/Silla.obj");
Model Sillon((char*)"Models/Sillon/Sillon.obj");
Model Stan((char*)"Models/Stan/Stan.obj");
Model Televisor((char*)"Models/Televisor/Televisor.obj");
Model Piso((char*)"Models/Piso/Piso.obj");
Model Brazo((char*)"Models/Stan/Brazo.obj");
Model Cuerpo((char*)"Models/Stan/Cuerpo.obj");
Model Fachada((char*)"Models/Casa/Fachada.obj");
Model Puerta((char*)"Models/Casa/Puerta.obj");
Model Arbusto((char*)"Models/Arbusto/Arbusto.obj");
Model Arbusto2((char*)"Models/Arbusto/Arbusto.obj");
```

Modelos utilizados.

*-Se crean objetos Model utilizando los archivos de los modelos.
*-Cada objeto Model representa un modelo 3D que se utilizará en la escena.
*-Los modelos se cargan desde los archivos especificados en las rutas.
*-Los objetos Model se utilizarán posteriormente para renderizar los objetos en la escena.

```
// Set up vertex data (and buffer(s)) and attribute pointers
GLfloat vertices[] =
{
        // Positions           // Normals            // Texture Coords
        -0.5f, -0.5f, -0.5f,   0.0f,  0.0f, -1.0f,   0.0f, 0.0f,
        0.5f, -0.5f, -0.5f,    0.0f,  0.0f, -1.0f,   1.0f, 0.0f,
        0.5f,  0.5f, -0.5f,    0.0f,  0.0f, -1.0f,   1.0f, 1.0f,
        0.5f,  0.5f, -0.5f,    0.0f,  0.0f, -1.0f,   1.0f, 1.0f,
        -0.5f,  0.5f, -0.5f,   0.0f,  0.0f, -1.0f,   0.0f, 1.0f,
        -0.5f, -0.5f, -0.5f,   0.0f,  0.0f, -1.0f,   0.0f, 0.0f,

        -0.5f, -0.5f,  0.5f,   0.0f,  0.0f,  1.0f,   0.0f, 0.0f,
        0.5f, -0.5f,  0.5f,    0.0f,  0.0f,  1.0f,   1.0f, 0.0f,
        0.5f,  0.5f,  0.5f,    0.0f,  0.0f,  1.0f,   1.0f, 1.0f,
        0.5f,  0.5f,  0.5f,    0.0f,  0.0f,  1.0f,   1.0f, 1.0f,
        -0.5f,  0.5f,  0.5f,   0.0f,  0.0f,  1.0f,   0.0f, 1.0f,
        -0.5f, -0.5f,  0.5f,   0.0f,  0.0f,  1.0f,   0.0f, 0.0f,

        -0.5f,  0.5f,  0.5f,   -1.0f,  0.0f,  0.0f,   1.0f, 0.0f,
        -0.5f,  0.5f, -0.5f,   -1.0f,  0.0f,  0.0f,   1.0f, 1.0f,
        -0.5f, -0.5f, -0.5f,   -1.0f,  0.0f,  0.0f,   0.0f, 1.0f,
        -0.5f, -0.5f, -0.5f,   -1.0f,  0.0f,  0.0f,   0.0f, 1.0f,
        -0.5f, -0.5f,  0.5f,   -1.0f,  0.0f,  0.0f,   0.0f, 0.0f,
        -0.5f,  0.5f,  0.5f,   -1.0f,  0.0f,  0.0f,   1.0f, 0.0f,

        0.5f,  0.5f,  0.5f,    1.0f,  0.0f,  0.0f,   1.0f, 0.0f,
        0.5f,  0.5f, -0.5f,    1.0f,  0.0f,  0.0f,   1.0f, 1.0f,
        0.5f, -0.5f, -0.5f,    1.0f,  0.0f,  0.0f,   0.0f, 1.0f,
        0.5f, -0.5f, -0.5f,    1.0f,  0.0f,  0.0f,   0.0f, 1.0f,
        0.5f, -0.5f,  0.5f,    1.0f,  0.0f,  0.0f,   0.0f, 0.0f,
        0.5f,  0.5f,  0.5f,    1.0f,  0.0f,  0.0f,   1.0f, 0.0f,

        -0.5f, -0.5f, -0.5f,   0.0f, -1.0f,  0.0f,   0.0f, 1.0f,
        0.5f, -0.5f, -0.5f,    0.0f, -1.0f,  0.0f,   1.0f, 1.0f,
        0.5f, -0.5f,  0.5f,    0.0f, -1.0f,  0.0f,   1.0f, 0.0f,
        0.5f, -0.5f,  0.5f,    0.0f, -1.0f,  0.0f,   1.0f, 0.0f,
        -0.5f, -0.5f,  0.5f,   0.0f, -1.0f,  0.0f,   0.0f, 0.0f,
        -0.5f, -0.5f, -0.5f,   0.0f, -1.0f,  0.0f,   0.0f, 1.0f,

        -0.5f,  0.5f, -0.5f,   0.0f,  1.0f,  0.0f,   0.0f, 1.0f,
        0.5f,  0.5f, -0.5f,    0.0f,  1.0f,  0.0f,   1.0f, 1.0f,
        0.5f,  0.5f,  0.5f,    0.0f,  1.0f,  0.0f,   1.0f, 0.0f,
        0.5f,  0.5f,  0.5f,    0.0f,  1.0f,  0.0f,   1.0f, 0.0f,
        -0.5f,  0.5f,  0.5f,   0.0f,  1.0f,  0.0f,   0.0f, 0.0f,
        -0.5f,  0.5f, -0.5f,   0.0f,  1.0f,  0.0f,   0.0f, 1.0f
};
```

```
GLfloat skyboxVertices[] = {
        // Positions
        -1.0f,  1.0f, -1.0f,
        -1.0f, -1.0f, -1.0f,
        1.0f, -1.0f, -1.0f,
        1.0f, -1.0f, -1.0f,
        1.0f,  1.0f, -1.0f,
        -1.0f,  1.0f, -1.0f,

        -1.0f, -1.0f,  1.0f,
        -1.0f, -1.0f, -1.0f,
        -1.0f,  1.0f, -1.0f,
        -1.0f,  1.0f, -1.0f,
        -1.0f,  1.0f,  1.0f,
        -1.0f, -1.0f,  1.0f,

        1.0f, -1.0f, -1.0f,
        1.0f, -1.0f,  1.0f,
        1.0f,  1.0f,  1.0f,
        1.0f,  1.0f,  1.0f,
        1.0f,  1.0f, -1.0f,
        1.0f, -1.0f, -1.0f,

        -1.0f, -1.0f,  1.0f,
        -1.0f,  1.0f,  1.0f,
        1.0f,  1.0f,  1.0f,
        1.0f,  1.0f,  1.0f,
        1.0f, -1.0f,  1.0f,
        -1.0f, -1.0f,  1.0f,

        -1.0f,  1.0f, -1.0f,
        1.0f,  1.0f, -1.0f,
        1.0f,  1.0f,  1.0f,
        1.0f,  1.0f,  1.0f,
        -1.0f,  1.0f,  1.0f,
        -1.0f,  1.0f, -1.0f,

        -1.0f, -1.0f, -1.0f,
        -1.0f, -1.0f,  1.0f,
        1.0f, -1.0f, -1.0f,
        1.0f, -1.0f, -1.0f,
        -1.0f, -1.0f,  1.0f,
        1.0f, -1.0f,  1.0f
};
```



Configuración de datos de vértices:

- Posiciones: Un arreglo de coordenadas que define la posición de cada vértice en el espacio tridimensional.
- Normales: Un arreglo de vectores que especifica la dirección de la normal de cada vértice.
- Coordenadas de textura: Un arreglo de coordenadas que asigna una textura a cada vértice.

Para cada vértice, se proporcionan los siguientes valores:

- Posición (x, y, z): Las coordenadas espaciales del vértice.
- Normal (nx, ny, nz): Las componentes del vector normal del vértice en el espacio tridimensional.
- Coordenadas de textura (tx, ty): Las coordenadas de textura que se utilizan para mapear una textura al vértice.

Estos datos se utilizan para renderizar un objeto tridimensional en un contexto gráfico.

```
// First, set the container's VAO (and VBO)
GLuint VBO, VAO, EBO;
glGenVertexArrays(1, &VAO);
glGenBuffers(1, &VBO);
glGenBuffers(1, &EBO);

glBindVertexArray(VAO);
glBindBuffer(GL_ARRAY_BUFFER, VBO);
glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW);

glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EBO);
glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(indices), indices, GL_STATIC_DRAW);

// Position attribute
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (GLvoid*)0);
glEnableVertexAttribArray(0);
// normal attribute
glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (GLvoid*)(3 * sizeof(GLfloat)));
glEnableVertexAttribArray(1);
// Texture Coordinate attribute
glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (GLvoid*)(6 * sizeof(GLfloat)));
glEnableVertexAttribArray(2);
glBindVertexArray(0);
// Set texture units
lightingShader.Use();
glUniform1i(glGetUniformLocation(lightingShader.Program, "material.diffuse"), 0);
glUniform1i(glGetUniformLocation(lightingShader.Program, "material.specular"), 1);
```

*-Primero, generamos un Vertex Array Object (VAO) para el contenedor y asignamos su identificador a la variable VAO.
*-También generamos un Vertex Buffer Object (VBO) y un Element Buffer Object (EBO).
*-Enlazamos el VAO actual para su configuración.

*-Enlazamos el VBO como un Buffer de Arreglos de GL y lo llenamos con los datos de los vértices.
*-Enlazamos el EBO como un Buffer de Arreglos de GL y lo llenamos con los índices de los elementos.
*-Configuramos el atributo de posición de los vértices y lo habilitamos.
*-Configuramos el atributo de normal de los vértices y lo habilitamos.
*-Configuramos el atributo de coordenadas de textura de los vértices y lo habilitamos.
*-Desenlazamos el VAO para evitar modificaciones accidentales.
*-Configuramos los valores de las unidades de textura
*-para la sombra y los materiales utilizados por el shader de iluminación.

```
// Then, we set the light's VAO (VBO stays the same. After all, the vertices are the same for the light object (also a 3D cube))
GLuint lightVAO;
glGenVertexArrays(1, &lightVAO);
glBindVertexArray(lightVAO);
// We only need to bind to the VBO (to link it with glVertexAttribPointer), no need to fill it; the VBO's data already contains all we need.
glBindBuffer(GL_ARRAY_BUFFER, VBO);
// Set the vertex attributes (only position data for the lamp))
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(GLfloat), (GLvoid*)0); // Note that we skip over the other data in our buffer object (we
glEnableVertexAttribArray(0);
glBindVertexArray(0);


//SkyBox
glEnable(GL_BLEND);

GLuint skyboxVBO, skyboxVAO;
glGenVertexArrays(1, &skyboxVAO);
glGenBuffers(1, &skyboxVBO);
glBindVertexArray(skyboxVAO);
glBindBuffer(GL_ARRAY_BUFFER, skyboxVBO);
glBufferData(GL_ARRAY_BUFFER, sizeof(skyboxVertices), &skyboxVertices, GL_STATIC_DRAW);
glEnableVertexAttribArray(0);
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(GLfloat), (GLvoid*)0);
```

*-A continuación, configuramos el Vertex Array Object (VAO) para la luz.

*-Utilizamos el mismo Vertex Buffer Object (VBO) ya que los vértices son los mismos para el objeto de luz (también un cubo en 3D).

*-Enlazamos el VAO de la luz para su configuración.

*-Solo necesitamos enlazar el VBO (para vincularlo con glVertexAttribPointer), no es necesario llenarlo; los datos del VBO ya contienen todo lo que necesitamos.

*-Configuramos los atributos de vértice (solo datos de posición para la lámpara).

*-Desenlazamos el VAO para evitar modificaciones accidentales.

*-Configuramos el Vertex Array Object (VAO) y Vertex Buffer Object (VBO) para el Skybox.

*-Generamos el Vertex Array Object (VAO) y Vertex Buffer Object (VBO) para el Skybox.

*-Enlazamos el Vertex Array Object (VAO) del Skybox para su configuración.

*-Enlazamos el Vertex Buffer Object (VBO) del Skybox y llenamos los datos de los vértices.

*-Habilitamos el atributo de posición de los vértices del Skybox.

*-Configuramos los atributos de vértice (solo datos de posición) para el Skybox.

```
// Load textures
vector<const GLchar*> faces;
faces.push_back("SkyBox/right.tga");
faces.push_back("SkyBox/left.tga");
faces.push_back("SkyBox/top.tga");
faces.push_back("SkyBox/bottom.tga");
faces.push_back("SkyBox/back.tga");
faces.push_back("SkyBox/front.tga");

GLuint cubemapTexture = TextureLoading::LoadCubemap(faces);

glm::mat4 projection = glm::perspective(camera.GetZoom(), (GLfloat)SCREEN_WIDTH / (GLfloat)SCREEN_HEIGHT, 0.1f, 100.0f);
```

*-Cargamos las texturas del Skybox.

-Creamos un vector de const GLchar para almacenar las rutas de las texturas del Skybox.

*-Añadimos las rutas de las texturas al vector.

*-Cargamos el cubemapTexture utilizando la función LoadCubemap de la clase TextureLoading.

*-Pasamos el vector de rutas de texturas como argumento.

*-Creamos la matriz de proyección utilizando la función perspective de glm.

*-La matriz de proyección se utiliza para transformar los vértices en coordenadas de proyección.

*-Pasamos el campo de visión (zoom de la cámara), la relación de aspecto, la distancia del plano cercano y el plano lejano como argumentos.

```
// Directional light
glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.direction"), -0.2f, -1.0f, -0.3f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.ambient"), 0.6f, 0.6f, 0.6f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.diffuse"), 0.6f, 0.6f, 0.6f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.specular"), 1.0f, 1.0f, 1.0f);


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


glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].position"), pointLightPositions[0].x, pointLightPositions[0].y, po
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].ambient"), lightColor.x, lightColor.y, lightColor.z);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].diffuse"), lightColor.x, lightColor.y, lightColor.z);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].specular"), 0.0f, 0.0f, 0.0f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[0].constant"), 1.0f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[0].linear"), 0.7f); //Determinan que tan brillante y que tanto se va
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[0].quadratic"), 1.8f);
```

*-Comienza el bucle principal del juego.

*-Calculamos el tiempo transcurrido entre fotogramas para obtener el deltaTime.

*-Actualizamos el lastFrame con el valor del currentFrame.

*-Comprobamos si hay eventos activos (teclas presionadas, movimiento del ratón, etc.) y llamamos a las funciones de respuesta correspondientes.

*-Utilizamos glfwPollEvents para procesar los eventos pendientes.

*-Limpiamos el búfer de color y el búfer de profundidad.

*-Establecemos el color de borrado con glClearColor y borramos los búferes con glClear.

*-Habilitamos la prueba de profundidad con glEnable(GL_DEPTH_TEST).Esto permite que OpenGL realice pruebas de profundidad para determinar qué fragmentos se dibujan delante y cuáles se ocultan.

*-Utilizamos el shader "lightingShader" para establecer las uniformes y dibujar los objetos.

*-Obtenemos la posición de la cámara y la establecemos como "viewPos" en el shader.

*-Establecemos la dirección de la luz direccional y sus propiedades (ambient, diffuse, specular) en el shader.

*-Calculamos el color de la luz puntual 1 utilizando el tiempo y las componentes Light1.

*-Establecemos la posición, el color y las propiedades de la luz puntual 1 en el shader.

```
// SpotLight
glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight.position"), camera.GetPosition().x, camera.GetPosition().y, camera.GetP
glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight.direction"), camera.GetFront().x, camera.GetFront().y, camera.GetFront(
glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight.ambient"), 0.0f, 0.0f, 0.0f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight.diffuse"), 0.0f, 0.0f, 0.0f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "spotLight.specular"), 0.0f, 0.0f, 0.0f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight.constant"), 1.0f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight.linear"), 0.0f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight.quadratic"), 0.0f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight.cutOff"), glm::cos(glm::radians(12.5f)));
glUniform1f(glGetUniformLocation(lightingShader.Program, "spotLight.outerCutOff"), glm::cos(glm::radians(15.0f)));

// Set material properties
glUniform1f(glGetUniformLocation(lightingShader.Program, "material.shininess"), 32.0f);

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

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

// Pass the matrices to the shader
glUniformMatrix4fv(viewLoc, 1, GL_FALSE, glm::value_ptr(view));
glUniformMatrix4fv(projLoc, 1, GL_FALSE, glm::value_ptr(projection));
glBindVertexArray(VAO);
glm::mat4 tmp = glm::mat4(1.0f); //Temp
```

*-Configuramos la luz de tipo SpotLight en el shader.

*-Establecemos la posición de la luz puntual como la posición de la cámara.

*-Establecemos la dirección de la luz puntual como la dirección frontal de la cámara.

*-Configuramos las propiedades ambient, diffuse y specular de la luz puntual como cero.

*-Configuramos las propiedades constant, linear y quadratic de la luz puntual como cero.

*-Configuramos los ángulos de corte de la luz puntual utilizando la función glm::cos y glm::radians.

*-Establecemos las propiedades del material en el shader, como el brillo (shininess).

*-Creamos las transformaciones de la cámara, obteniendo la matriz de vista con camera.GetViewMatrix. Obtenemos las ubicaciones de las uniformes en el shader para model, view y projection.

*-Pasamos las matrices al shader utilizando glUniformMatrix4fv para las matrices de vista y proyección.

*-Enlazamos el VAO y asignamos una matriz de identidad (temp) a la variable tmp.

```
//Carga de modelo
//Casa
view = camera.GetViewMatrix();
glm::mat4 model(1);
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-8.0f, 3.39f, -1.2f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Fachada.Draw(lightingShader);
```

*-Carga del modelo de la casa:

*-Obtenemos la matriz de vista utilizando camera.GetViewMatrix.

*-Creamos una matriz de modelo y la inicializamos como una matriz de identidad.

*-Aplicamos una transformación de traslación a la matriz de modelo para posicionar la casa en la escena.Pasamos la matriz de modelo al shader utilizando glUniformMatrix4fv.

*-Dibujamos el modelo de la fachada de la casa utilizando el método Draw del objeto Fachada.

*-Realizaremos lo mismo para los siguientes objetos

```
// Also draw the lamp object, again binding the appropriate shader
lampShader.Use();
// Get location objects for the matrices on the lamp shader (these could be different on a different shader)
modelLoc = glGetUniformLocation(lampShader.Program, "model");
viewLoc = glGetUniformLocation(lampShader.Program, "view");
projLoc = glGetUniformLocation(lampShader.Program, "projection");

///* Set matrices*/
glUniformMatrix4fv(viewLoc, 1, GL_FALSE, glm::value_ptr(view));
glUniformMatrix4fv(projLoc, 1, GL_FALSE, glm::value_ptr(projection));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(0.0f, 0.1f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
//SV.Draw(lampShader);
glBindVertexArray(0);
speed = 0.5f;

Anim2.Use();
tiempo = glfwGetTime() * speed;
// 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);
```

Dibujar el objeto de la lámpara:

*-Usamos el shader de la lámpara (lampShader).

*- Obtenemos las ubicaciones de los objetos para las matrices en el shader de la lámpara.

*- Establecemos las matrices (view, projection y model) en el shader de la lámpara utilizando glUniformMatrix4fv.

*- Aplicamos una transformación de traslación a la matriz de modelo para posicionar la lámpara en la escena.

*- Pasamos la matriz de modelo al shader utilizando glUniformMatrix4fv.

*- Dibujamos el objeto de la lámpara utilizando el método Draw del objeto SV.

Usamos el shader de la animación (Anim2).

*- Obtenemos las ubicaciones de los objetos para las matrices en el shader de la animación.

*- Establecemos las matrices (view, projection y model) en el shader de la animación utilizando glUniformMatrix4fv.

*- Aplicamos una transformación de rotación y traslación a la matriz de modelo para posicionar el arbusto en la escena.

*- Pasamos la matriz de modelo al shader utilizando glUniformMatrix4fv.

*- Pasamos el tiempo actual al shader utilizando glUniform1f.

*- Dibujamos el objeto del arbusto utilizando el método Draw del objeto Arbusto.

```
            // Draw skybox as last
            glDepthFunc(GL_LEQUAL);  // Change depth function so depth test passes when values are equal to depth buffer's content
            SkyBoxshader.Use();
            view = glm::mat4(glm::mat3(camera.GetViewMatrix()));    // Remove any translation component of the view matrix
            glUniformMatrix4fv(glGetUniformLocation(SkyBoxshader.Program, "view"), 1, GL_FALSE, glm::value_ptr(view));
            glUniformMatrix4fv(glGetUniformLocation(SkyBoxshader.Program, "projection"), 1, GL_FALSE, glm::value_ptr(projection));

            // skybox cube
            glBindVertexArray(skyboxVAO);
            glActiveTexture(GL_TEXTURE1);
            glBindTexture(GL_TEXTURE_CUBE_MAP, cubemapTexture);
            glDrawArrays(GL_TRIANGLES, 0, 36);
            glBindVertexArray(0);
            glDepthFunc(GL_LESS); // Set depth function back to default


            // Swap the screen buffers
            glfwSwapBuffers(window);
        }
        glDeleteVertexArrays(1, &VAO);
        glDeleteVertexArrays(1, &lightVAO);
        glDeleteBuffers(1, &VBO);
        glDeleteBuffers(1, &EBO);
        glDeleteVertexArrays(1, &skyboxVAO);
        glDeleteBuffers(1, &skyboxVBO);
        // Terminate GLFW, clearing any resources allocated by GLFW.
        glfwTerminate();

        return 0;
```

Dibujar el skybox al final:

*- Cambiamos la función de profundidad (depthFunc) para que la prueba de profundidad pase cuando los valores sean iguales al contenido del búfer de profundidad.

*- Usamos el shader del skybox (SkyBoxshader).

*- Eliminamos cualquier componente de traslación de la matriz de vista (view) para que el skybox esté fijo en la posición de la cámara.

*- Establecemos las matrices (view y projection) en el shader del skybox utilizando glUniformMatrix4fv.

*- Enlazamos el VAO (skyboxVAO) y activamos la textura del cubemap (cubemapTexture).

*- Dibujamos los triángulos del skybox utilizando glDrawArrays.

*- Restauramos la función de profundidad a su valor predeterminado.

Intercambiamos los búferes de pantalla.

*- Eliminamos los VAO, VBO y EBO utilizados.

*- Terminamos GLFW, liberando cualquier recurso asignado por GLFW.

Finaliza la función main y retorna 0.

```
// Moves/alters the camera positions based on user input
void DoMovement()
{

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

        }

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


        }

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


        }
```

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


    }

    //Primera Animacion de saludo
    if (anim)
    {
            if (!anim2 && rotStan < 20.0f)
                    rotStan += 0.5f;
            else if (anim2 && rotStan > 0.0f)
                    rotStan -= 0.5f;
            if (rotStan >= 20.0f)
                    anim2 = true;
            else if (rotStan <= 0.0f)
                    anim2 = false;
    }
```

```
//Segunda Animacion de cerrar puerta
if (anim3) {
        if (!Open) {
                if (rotPuerta > -90.0f) {
                        rotPuerta -= 0.5f;
                }
                else {
                        anim3 = false;
                }
        }
        else {
                if (rotPuerta < 0.0f) {
                        rotPuerta += 0.5f;
                }
                else {
                        anim3 = false;
                }
        }
}
```

```
        //Tercera animacion de cajon
        if (CajonAbierto) {

                if (CajonAbrir)
                {
                        movCajon += 0.01;
                        if (movCajon >= 0.4f)
                        {
                                CajonAbrir = false;
                        }

                }
                if (CajonCerrar)
                {
                        movCajon -= 0.01;
                        if (movCajon <= 0.02f)
                        {
                                CajonCerrar = false;

                        }
                }

        }


}
```

Función: DoMovement()

*-Mueve/altera la posición de la cámara según la entrada del usuario.

Control de la cámara:

* - Si se presiona la tecla W o la flecha arriba, mueve la cámara hacia adelante.

*- Si se presiona la tecla S o la flecha abajo, mueve la cámara hacia atrás.

*- Si se presiona la tecla A o la flecha izquierda, mueve la cámara hacia la izquierda.

* - Si se presiona la tecla D o la flecha derecha, mueve la cámara hacia la derecha.

Animaciones:

* - Animación de saludo:

* - Si la bandera "anim" es verdadera:

* - Si la bandera "anim2" es falsa y "rotStan" es menor a 20.0f, incrementa "rotStan" en 0.5f.

* - Si la bandera "anim2" es verdadera y "rotStan" es mayor a 0.0f, decrementa "rotStan" en 0.5f.

*- Si "rotStan" es mayor o igual a 20.0f, establece la bandera "anim2" como verdadera.

*- Si "rotStan" es menor o igual a 0.0f, establece la bandera "anim2" como falsa.

Animación de cerrar puerta:

*- Si la bandera "anim3" es verdadera:

*- Si la bandera "Open" es falsa y "rotPuerta" es mayor a -90.0f, decrementa "rotPuerta" en 0.5f.

*- Si la bandera "Open" es verdadera y "rotPuerta" es menor a 0.0f, incrementa "rotPuerta" en 0.5f.

*- Si "rotPuerta" es menor o igual a -90.0f, establece la bandera "anim3" como falsa.

*- Si "rotPuerta" es mayor o igual a 0.0f, establece la bandera "anim3" como falsa.

Animación de cajón:

*- Si la bandera "CajonAbierto" es verdadera:

*- Si la bandera "CajonAbrir" es verdadera, incrementa "movCajon" en 0.01.

*- Si "movCajon" es mayor o igual a 0.4f, establece la bandera "CajonAbrir" como falsa.

*- Si la bandera "CajonCerrar" es verdadera:

*- Si la bandera "CajonCerrar" es verdadera, decrementa "movCajon" en 0.01.

*- Si "movCajon" es menor o igual a 0.02f, establece la bandera "CajonCerrar" como falsa.

```
// Is called whenever a key is pressed/released via GLFW
void KeyCallback(GLFWwindow* window, int key, int scancode, int action, int mode)
{
	if (GLFW_KEY_ESCAPE == key && GLFW_PRESS == action)
	{
		glfwSetWindowShouldClose(window, GL_TRUE);
	}

	if (key >= 0 && key < 1024)
	{
		if (action == GLFW_PRESS)
		{
			keys[key] = true;
		}
		else if (action == GLFW_RELEASE)
		{
			keys[key] = false;
		}
	}
	//Boton para activar animacion 1
	if (keys[GLFW_KEY_Z])
	{
		anim = true;
	}
```

```
	//Boton para activar animacion 2
	if (keys[GLFW_KEY_X]) {
		Open = !Open;
		anim3 = true;

	}
	//Boton para activar animacion 3
	if (keys[GLFW_KEY_C])
	{
		CajonAbierto = true;
		if (movCajon >= 0.4f) {
			CajonCerrar = true;
			CajonAbrir = false;
		}
		else if (movCajon <= 0.02f) {
			CajonCerrar = false;
			CajonAbrir = true;

		}

	}
```

Función: KeyCallback(GLFWwindow* window, int key, int scancode, int action, int mode)

*-Se llama cuando se presiona/libera una tecla a través de GLFW.

Parámetros:

*- window: Puntero a la ventana GLFW actual.

*- key: Código de la tecla presionada/liberada.

*- scancode: Código específico de la plataforma para la tecla.

*- action: Acción realizada en la tecla (presionada o liberada).

*- mode: Modo de la tecla (Shift, Ctrl, Alt, Super).

Escape:

*- Si se presiona la tecla Escape (GLFW_KEY_ESCAPE) y la acción es GLFW_PRESS, se cierra la ventana GLFW.

Control de teclas:

*- Si la tecla tiene un código válido (key >= 0 y key < 1024):

*- Si la acción es GLFW_PRESS, se establece la tecla correspondiente en el arreglo "keys" como verdadera.

*- Si la acción es GLFW_RELEASE, se establece la tecla correspondiente en el arreglo "keys" como falsa.

Activación de animaciones:

*- Si se presiona la tecla Z (GLFW_KEY_Z), se activa la animación 1 estableciendo la bandera "anim" como verdadera.

*- Si se presiona la tecla X (GLFW_KEY_X), se activa la animación 2 invirtiendo el valor de la variable "Open" y estableciendo la bandera "anim3" como verdadera.

*- Si se presiona la tecla C (GLFW_KEY_C), se activa la animación 3 estableciendo la bandera "CajonAbierto" como verdadera.

*- Si "movCajon" es mayor o igual a 0.4f, se establece la bandera "CajonCerrar" como verdadera y "CajonAbrir" como falsa.

*- Si "movCajon" es menor o igual a 0.02f, se establece la bandera "CajonCerrar" como falsa y "CajonAbrir" como verdadera.

```
void MouseCallback(GLFWwindow* window, double xPos, double yPos)
{
        if (firstMouse)
        {
                lastX = xPos;
                lastY = yPos;
                firstMouse = false;
        }

        GLfloat xOffset = xPos - lastX;
        GLfloat yOffset = lastY - yPos;  // Reversed since y-coordinates go from bottom to left

        lastX = xPos;
        lastY = yPos;

        camera.ProcessMouseMovement(xOffset, yOffset);
}
```

-Función: MouseCallback(GLFWwindow window, double xPos, double yPos)

*-Se llama cuando se mueve el mouse dentro de la ventana GLFW.

Parámetros:

*- window: Puntero a la ventana GLFW actual.

*- xPos: Posición X del cursor del mouse.

*- yPos: Posición Y del cursor del mouse.

*-Flujo de trabajo:

Primera vez que se mueve el mouse:

*- Si es la primera vez que se mueve el mouse, se guarda la posición actual del cursor en las variables "lastX" y "lastY", y se establece la bandera "firstMouse" como falsa.

Cálculo de desplazamiento del mouse:

*- Se calcula el desplazamiento del mouse en el eje X y se almacena en la variable "xOffset" (diferencia entre la posición actual y la anterior).

*- Se calcula el desplazamiento del mouse en el eje Y y se almacena en la variable "yOffset" (diferencia entre la posición anterior y la actual, invertida debido a que las coordenadas Y van de abajo hacia arriba).

*- Se actualizan las variables "lastX" y "lastY" con las posiciones actuales del cursor.

Procesamiento del movimiento del mouse en la cámara:

*- Se llama a la función "ProcessMouseMovement" de la cámara, pasando como argumentos los desplazamientos "xOffset" y "yOffset".

8-Conclusiones

En la realización de este proyecto, tuvimos algunos detalles y algunas complicaciones en su realización con algunos problemas que si por intentar si se podían mejorar o si se podían implementar tuvimos problema en la organización del tiempo y nos empeñamos más en la realización de un trabajo que cumpliera con los criterios a evaluar. Por otro lado, la experiencia que nos llevamos podemos que decir que es lo más realista a que un desarrollador tiene que realizar, empleando ser autodidacta, aprendiendo contra reloj y siguiendo un estricto cronograma que pueda cumplir a la perfección todos los requerimientos que un cliente pida y requiera.
En los conocimientos adquiridos podemos ser sinceros en que fueron bastantes y la mayoría los empleamos en su realización de la práctica, algunos más que otros, en el apartado de modelado podemos ver que fue la parte mas divertida y fue muy dinámica el tratar de representar diferentes objetos y representar un personaje favorito, en otro punto la textura y que se pudieran ver los mas cartoon no fue complicado, pero si fue muy laborioso.

En el apartado de limitaciones pudimos hablar de los problemas que tuvimos y en realizad el pensar que son pocos, nos hace pensar que eso no es nada comparado a un proyecto de verdad, pero nos vamos satisfecho con lo logrado en este proyecto y las prácticas en general. Para finalizar, se logró cumplir con el objetivo principal y logramos implementar los conocimientos adquiridos en nuestro proyecto.

9-Bibliografia

- U-Tad. (2022, 22 noviembre). Autodesk Maya: el software de modelado 3D que debes conocer | U-tad. *U-tad*. https://u-tad.com/autodesk-maya-el-software-de-modelado-3d-que debes-conocer/

-Meneses, N. (2022b, julio 6). *CÓMO USAR GITHUB: Guía para Principiantes – Coding Dojo Latam*. Coding Dojo Latam. https://www.codingdojo.la/2022/07/06/como-usar github-guia-para-principiantes/

-*South Park | South Park Archives | Fandom*. (s. f.). South Park Archives.

https://southpark.fandom.com/wiki/South_Park

-colaboradores de Wikipedia. (2022). South Park (videojuego). *Wikipedia, la enciclopedia libre*. https://es.wikipedia.org/wiki/South_Park_(videojuego)

-*OpenGL: Primitivas, Proyección e Iluminación de objetos 3D*. (s. f.). https://academiaandroid.com/opengl-primitivas-proyeccion-iluminacion-de-objetos-3d/

. GameBanana. (s. f.). *Search | GameBanana*. https://gamebanana.com/search?_nPage=1&_sOrder=best_match&_idGameRow=35&_sSearchString=skybox