NATIONAL AUTONOMOUS
UNIVERSITY OF MEXICO

FACULTY OF ENGINEERING

ELECTRICAL ENGINEERING

DIVISION COMPUTER
ENGINEERING COMPUTER

GRAPHICS and HUMAN-COMPUTER INTERACTION

Technical Manual

Account Number: 317335930

Group: 12

Deadline date: 26/05/2023

Semester: 2023-2

Table of Contents

Content

1. Objectives
2. Preliminary Research
   2.1- Flowchart
   2.2- Gantt Chart
3. Conceptual Design
   3.1- Created Models
   3.2- Project Scope
4. Tool Selection
5. Prototype Development
6. Limitations
7. Documentation
8. Conclusions
9. References

1-Objective

The main objective of this Prototype project is to create a 3D model of our proposal within the "South Park" video game setting, inspired by the original Nintendo 64 game. This involves modeling and texturing 7 objects, creating both complex and simple animations, building a scene environment, and implementing Open gl coding.

With this Prototype project, we aim to determine if the work was executed according to the given instructions, achieving a dynamic environment with a cartoon design that represents the television series. This prototype serves as a deliverable and reinforces the knowledge gained during our Computer Graphics course.

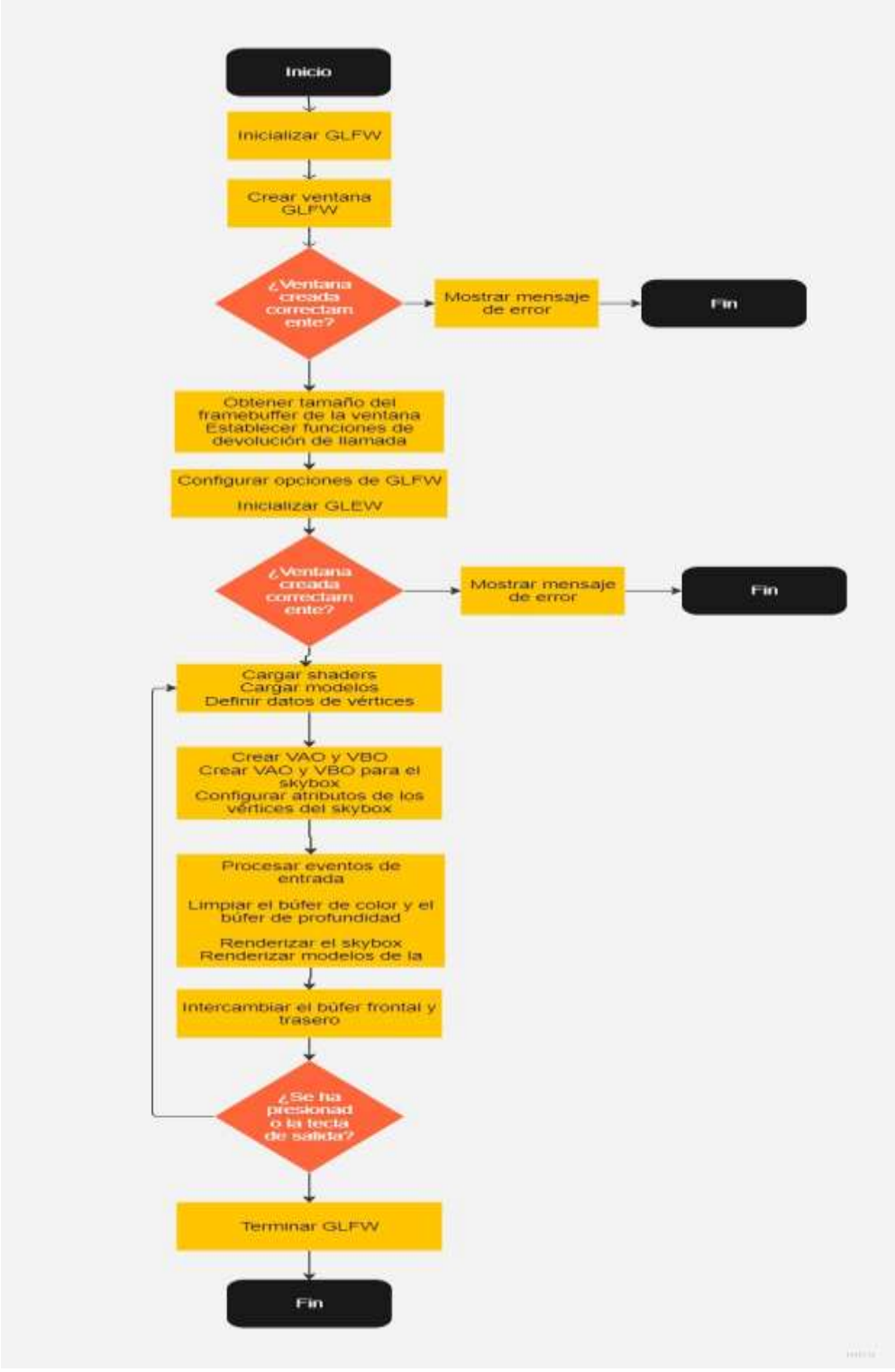
2-Preliminary Research

In our preliminary research for the project, we first needed to understand the modeling and environment we had to create. Our proposal involves creating a house from the "South Park: The Fractured but Whole" video game, drawing inspiration from its design in Nintendo 64 style. This approach adds charisma to the modeling and gives it a 2D-to-3D retrospective feel, using a different perspective and dynamic space.

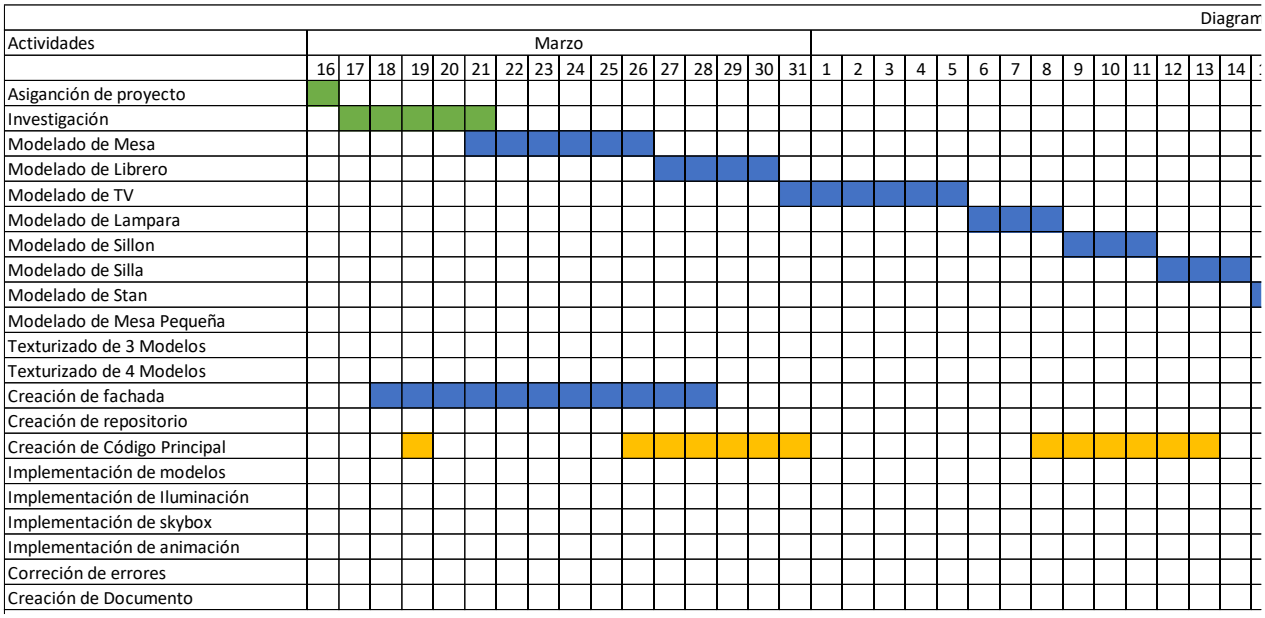
Initially, we decided to follow a prototype methodology because it is a small-scale project and we wanted to create a prototype before embarking on a larger-scale development. This approach allows us, as developers, to understand what needs to be done and how it should function, uncover any flaws or potential changes that can be implemented, and consider improvements for the final outcome.

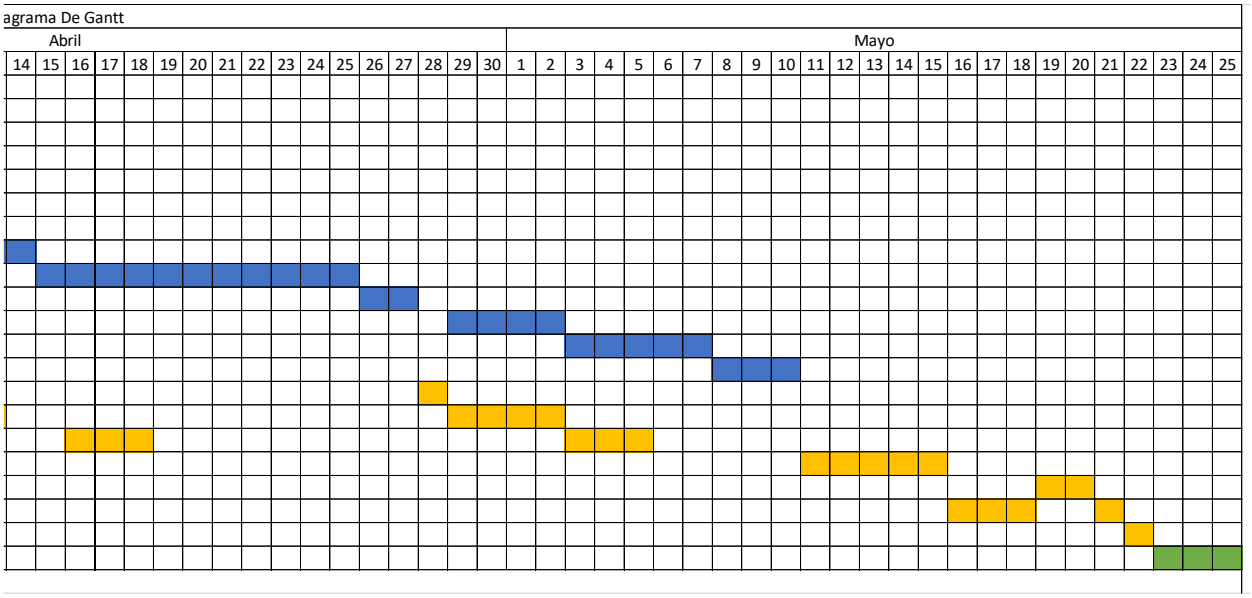
Starting with an investigation into the type of environment, we learned that we needed to create cartoon-like materials. The texturing had to be matte to achieve the desired style, and we aimed to incorporate soft lighting that would create the illusion of being inside the house. Lastly, we developed a basic code structure that would encompass all the models, lighting, skybox, simple animations, complex animations, and camera control.

2.1- Flowchart



2.2- Gantt Chart





3-Conceptual Design

The reference facade we took was Stan's house, which is representative of the series.

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

For the objects, we took them from the video game "South Park: The Stick of Truth" as the furniture will be themed and designed with the model of that time in the game. As we can see, the models are in a 2D image, so we will refer to the models from the Nintendo 64 game to create a similar style that captures the essence of video game graphics from that era.

“South park: La vara de la verdad“

We will show some images from the Nintendo 64 game to provide a reference for how we are guiding ourselves.

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

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

3.1- Models Created

We will show a reference image and an image of the completed models we proposed.

Chair:

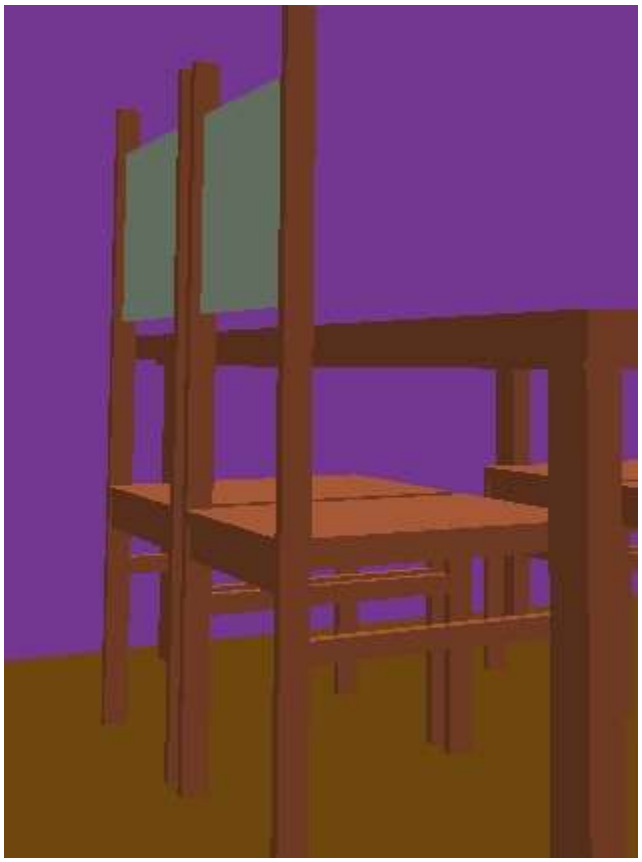
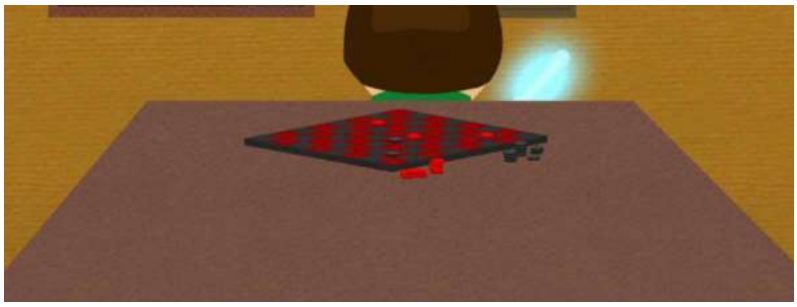
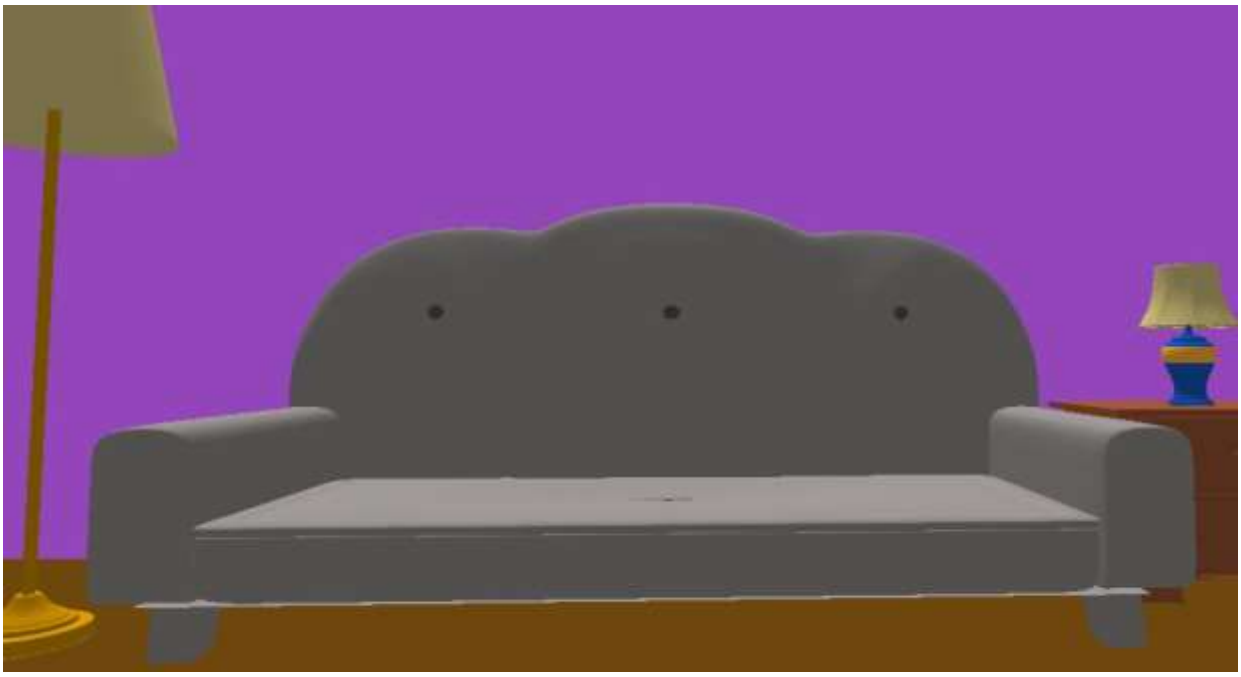


Table:

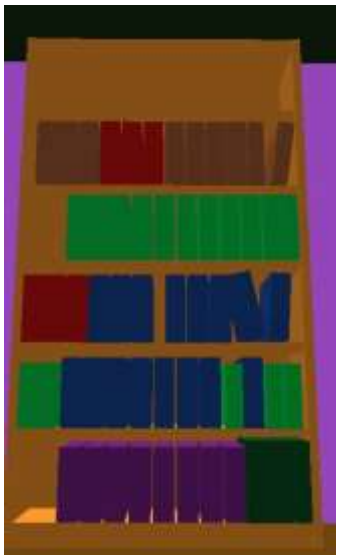


Sofa:



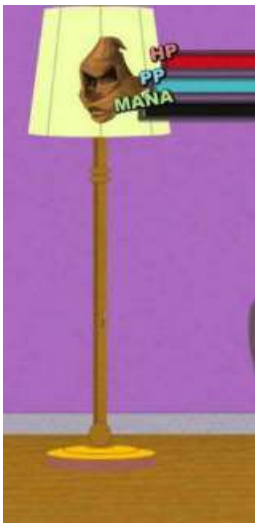
Small table:

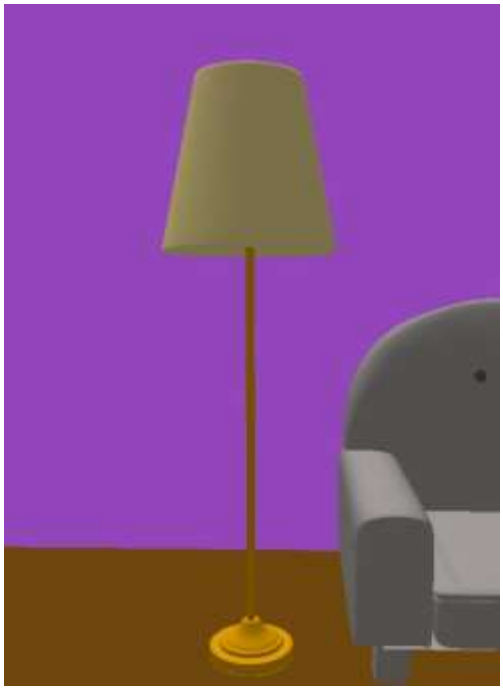
Bookshelf:



TV:

Lamp:





As a bonus, we also modeled a character from the series and the character who lives in the house:

Character:

For the facade, we created the proposed house inspired by the game, staying true to the proposed model and ensuring its similarity. Additionally, we added some bushes for an animation:

To conclude, we will take images of the final model with each object and facade:

3.2- Project Scope

With the completion of the models, we are extremely satisfied with the outcome. We have achieved a great similarity to the proposed designs and successfully captured the cartoon-style design we were aiming for through texturing. The modeling and texturing work have been exceptional.

For the house model, we had the vision of the in-game house but focused on the proposed house, paying attention to the details and ensuring the interior resembles a living room with accurately placed furniture, creating the desired atmosphere.

For the exterior skybox model, we managed to obtain the original 6 images from the video game. We had to align them correctly and convert the files. We also found a cartoon-like snow texture to maintain the style of the video game. Additionally, we modeled the bushes, which are essential for an animation.

Regarding lighting, we opted for an illumination that is not too bright but also not too dim. This was mainly done to maintain the cartoon aesthetic for both the outdoor environment and the interior of the house. We also took into consideration that the original game did not have intense lighting in the house interiors, as it affected the game's performance. Thus, we adopted a concept of using lighting that is neither too bright nor too dim, similar to the original game's approach.

4-Tool Selection

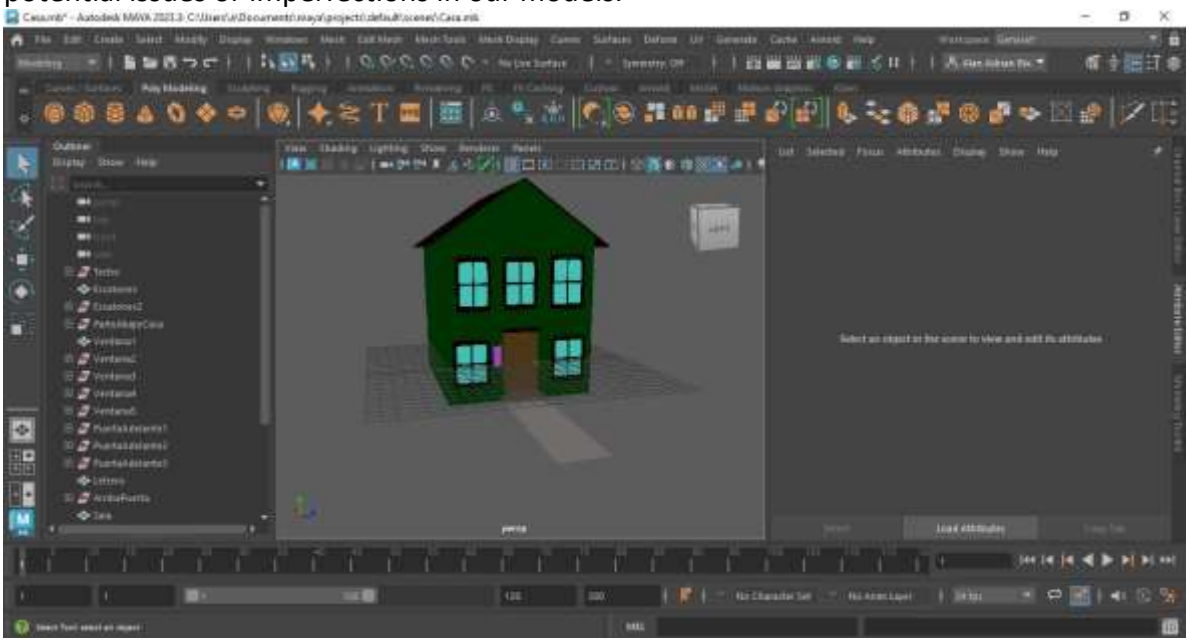
- Visual Studio: Primarily used for developing desktop, web, and mobile applications. It is one of the most popular tools in the software development industry and provides a comprehensive set of features and tools that facilitate the development process.
- Maya: A widely used 3D animation, modeling, and rendering software in the entertainment, design, and architectural visualization industries. Developed by Autodesk, Maya offers advanced tools that enable artists and designers to create high-quality digital content.
- GitHub: A web platform that provides hosting services for source code repositories and collaboration tools for software developers. It is widely used in the software development community and provides a centralized environment for storing, tracking, and managing versions of open-source and private projects.
- GitHub Desktop: A desktop application that provides a graphical user interface for interacting with repositories hosted on GitHub. It allows developers to clone repositories, make code changes, create branches, and push changes to the remote repository, all without needing to use command-line commands. It is a

useful tool for those who prefer an intuitive and simplified graphical interface for working with Git and GitHub.

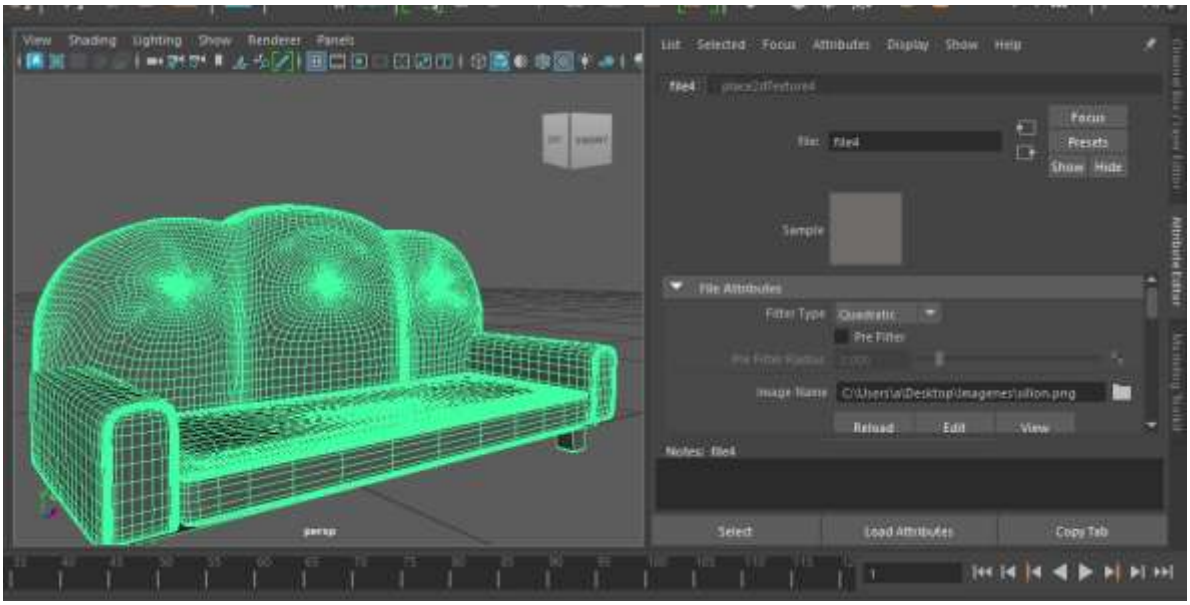
- C++: A general-purpose programming language widely used in software development. It was developed from the original C language with the goal of adding object-oriented programming features and facilitating the writing of more complex and efficient programs.
- GIMP: An open-source and free image editing software that offers a wide range of tools for creating and editing digital graphics. It is a popular and powerful alternative to commercial image editing programs like Adobe Photoshop.

5-Prototype Development

To begin with, we focused on researching and selecting a simple model that would be enjoyable to create without requiring excessive work. Starting with the modeling process, we first familiarized ourselves with Maya, exploring different commands and tools available to create our models. We aimed to create one model per week to identify any potential issues or imperfections in our models.



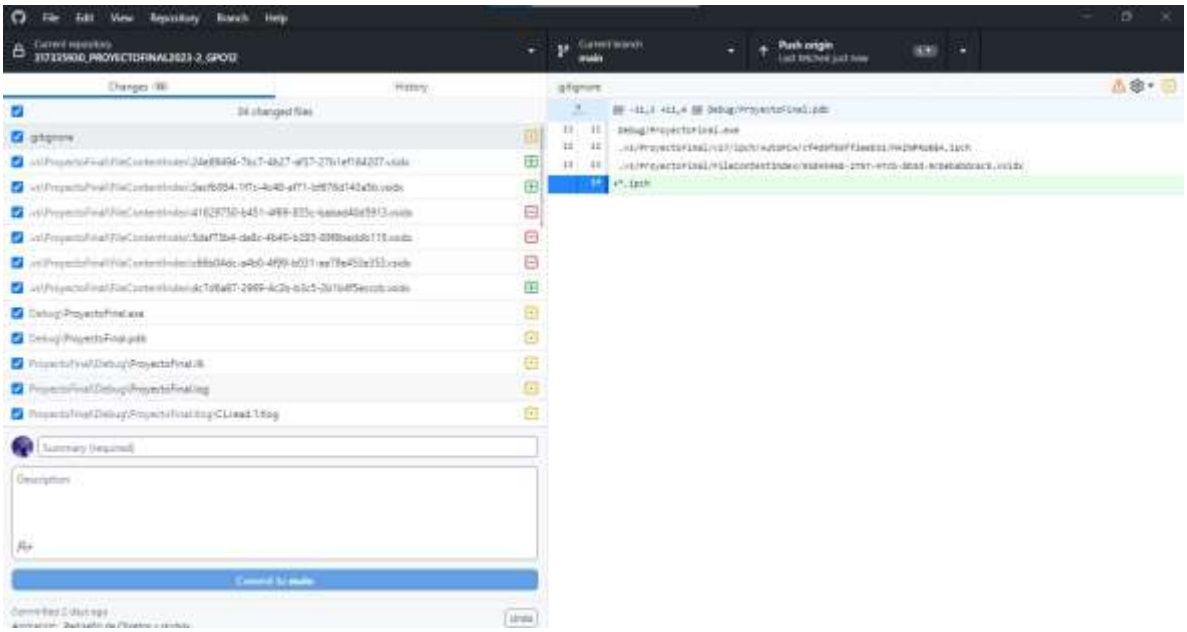
Maya was easy to understand after modeling several models, and for texturing, we used the Lambert shader and GIMP to apply textures by assigning different colors to each object. We aimed to create a cohesive color scheme for all objects to avoid generating too many similar images.



Regarding the creation of the facade, we started with the more challenging top part, which had a triangular-shaped roof. For the bottom part, we only needed to make a cutout for the door and add exterior elements such as windows with an animated tone and a designed door. In the interior, we added an extra square to differentiate the color from the exterior, and for the floor, we raised it slightly to incorporate a step-like design at the entrance, including a path.

For the repository implementation, we made a copy of the one implemented in the general class group. We created a fork to have our own copy and made the necessary modifications. Using GitHub Desktop, we performed different commits to track all the

changes made. Afterwards, we could push the changes and have them registered in our repository.

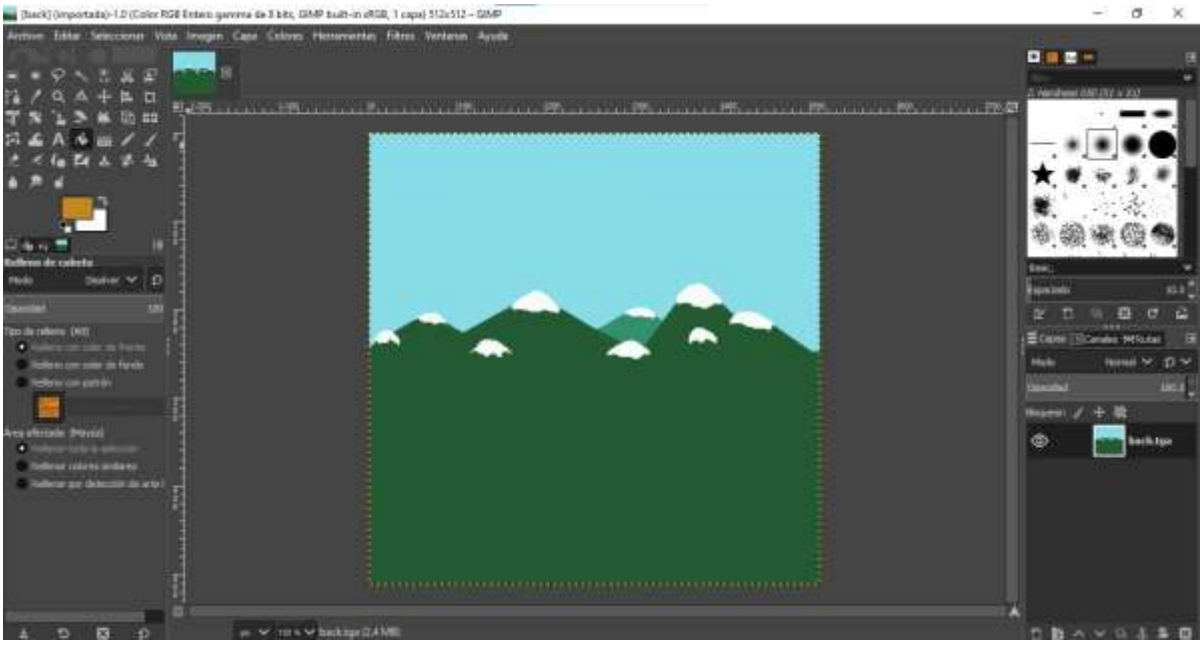


To implement the models, we first loaded them into the "models" folder, including the .obj file and the textures. Then, in our code implementation, we created the models and could visualize them

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	

For the skybox implementation, we searched different websites for the original skybox from the series. Once we found it, we had to convert the file type to JPG, rename the

image, and export it as .tga. Finally, we placed the code for the skybox and ensured that there were no issues.



6-Limitations

In general, we encountered some limitations in the project, mainly due to time constraints. We believe there is room for improvement in terms of lighting and animation, which were the areas where we faced the most challenges. Resolving these issues took a considerable amount of time.

Due to these limitations, we had to create a basic lighting setup that captured the desired atmosphere but lacked intricate details. Regarding animations, the original plan was to create five (three simple animations and two complex animations). However, we were only able to complete three simple animations (door, greeting, and drawer) and one complex animation (bush movement).

Despite these limitations, we are proud of the prototype we are delivering and believe that we have applied all the knowledge and skills acquired.

7-Documentation

Within the initial test document, we found that it was quite advanced and already contained many additions that will help us in the implementation of our own code, which we will provide. Therefore, in the documentation, we will explain what each part we added does and the purpose for which we implemented it.

```
#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"
```

This section includes several libraries and header files required for the code.

- The included libraries and header files are:
- iostream: Provides basic input/output operations.
- cmath: Provides mathematical functions such as sqrt, sin, cos, etc.
- GLEW: OpenGL Extension Wrangler Library for managing OpenGL extensions.
- GLFW: Library for creating and managing windows, OpenGL contexts, and handling user input.
- stb_image: Library for loading various image formats.
- glm: Mathematical library for OpenGL applications, including vector and matrix operations.
- SOIL2: Library for loading image files as textures.

- Shader: Custom header file for managing shaders.
- Camera: Custom header file for handling camera operations.
- Model: Custom header file for loading and rendering 3D models.
- Texture: Custom header file for managing textures.

```
// 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;
```

Function prototypes.

- KeyCallback: Keyboard callback function called when a key is pressed.
- MouseCallback: Mouse callback function called when the mouse is moved.
- DoMovement: Performs camera movements based on the keys pressed.

Window dimensions.

- The constants WIDTH and HEIGHT represent the dimensions of the window in pixels.

- SCREEN_WIDTH and SCREEN_HEIGHT are used to store the current screen dimensions.

Camera.

- The camera is initialized with an initial position and is used to control the view of the scene.
- lastX and lastY are used to store the coordinates of the last mouse position.
- keys is an array of booleans indicating which keys are currently pressed.
- firstMouse indicates if this is the first time the mouse is being moved.

Movements. Variables related to scene movements.

- rotStan: Rotation of the character Stan.
- rotPuerta: Rotation of the door.
- movCajon: Drawer displacement.
- anim, anim2, anim3: Animation variables.
- CajonAbierto, CajonAbrir, CajonCerrar: Variables related to opening and closing the drawer.
- tran: Transparency.
- Open: Opening variable.
- tiempo: Time.
- speed: Movement speed.

```
// 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
```

Positions of point lights.

- The pointLightPositions array contains the positions in 3D space of the point lights.
- Each element of the array represents a different point light.

Position of Light1.

- Light1 represents the position of a specific light in 3D space.

DeltaTime.

- deltaTime stores the elapsed time between the current frame and the last frame.
- lastFrame stores the time of the last frame.

```
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);
```

Main function of the program.

- The main function is the entry point of the program.
- In this function, GLFW is initialized, a window is created, and necessary options are configured.
- Callback functions for keyboard and mouse are set.
- Finally, the mouse input mode is set, and the dimensions of the window's framebuffer are obtained.
- 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");
```

Initialization of shaders.

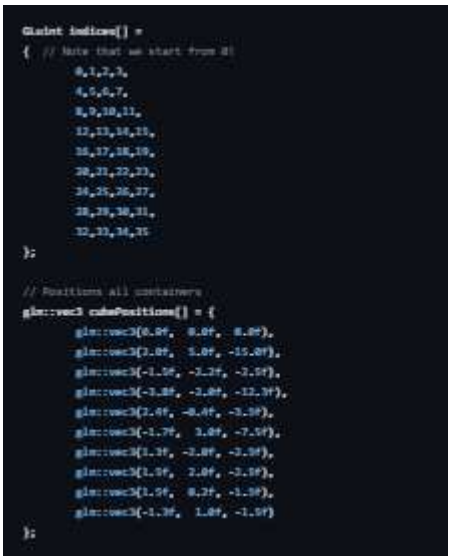
- Shader objects are created using shader files.
- Each Shader object represents a shader program that will be used in rendering.
- Shaders are loaded from the files "lighting.vs", "lighting.frag", "lamp.vs", "lamp.frag", "anim.vs", "anim.frag", "anim2.vs", "anim2.frag", "SkyBox.vs", and "SkyBox.frag".
- The Shader objects will be used later for rendering the objects in the scene.

```
//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");
```

Models used.

- Model objects are created using model files.
- Each Model object represents a 3D model that will be used in the scene.
- The models are loaded from the specified file paths.
- The Model objects will be used later for rendering the objects in the scene.

```
// 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
};


// 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
};
```



```
GLuint indices[] =
{  // Note that we start from 0!
        0,1,2,3,
        4,5,6,7,
        8,9,10,11,
        12,13,14,15,
        16,17,18,19,
        20,21,22,23,
        24,25,26,27,
        28,29,30,31,
        32,33,34,35
};

// Positions all containers
glm::vec3 cubePositions[] = {
        glm::vec3(0.0f,  0.0f,  0.0f),
        glm::vec3(2.0f,  5.0f, -15.0f),
        glm::vec3(-1.5f, -2.2f, -2.5f),
        glm::vec3(-3.8f, -2.0f, -12.3f),
        glm::vec3(2.4f, -0.4f, -3.5f),
        glm::vec3(-1.7f,  3.0f, -7.5f),
        glm::vec3(1.3f, -2.0f, -2.5f),
        glm::vec3(1.5f,  2.0f, -2.5f),
        glm::vec3(1.5f,  0.2f, -1.5f),
        glm::vec3(-1.3f,  1.0f, -1.5f)
};
```

Vertex Data Configuration:

- Positions: An array of coordinates that define the position of each vertex in 3D space.
- Normals: An array of vectors that specify the direction of the normal for each vertex.
- Texture Coordinates: An array of coordinates that maps a texture to each vertex.

For each vertex, the following values are provided:

- Position (x, y, z): The spatial coordinates of the vertex.
- Normal (nx, ny, nz): The components of the vertex's normal vector in 3D space.
- Texture Coordinates (tx, ty): The texture coordinates used to map a texture to the vertex.

These data are used to render a 3D object in a graphical context.

```
// 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);
```

Here's the breakdown of the steps involved in configuring the vertex data:

- First, we generate a Vertex Array Object (VAO) for the container and assign its identifier to the variable VAO.
- We also generate a Vertex Buffer Object (VBO) and an Element Buffer Object (EBO).
- We bind the current VAO for configuration.
- We bind the VBO as a GL Array Buffer and fill it with the vertex data.
- We bind the EBO as a GL Element Array Buffer and fill it with the element indices.
- We configure the position attribute of the vertices and enable it.
- We configure the normal attribute of the vertices and enable it.
- We configure the texture coordinate attribute of the vertices and enable it.
- We unbind the VAO to prevent accidental modifications.
- We set the texture units' values for shadow and materials used by the lighting shader.

```
// 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);
```

Here's the breakdown of the steps involved in configuring the vertex data for the light object and the skybox:

For the light object:

- Next, we configure the Vertex Array Object (VAO) for the light.
- We use the same Vertex Buffer Object (VBO) since the vertices are the same for the light object (also a 3D cube).
- We bind the VAO for the light for configuration.
- We only need to bind the VBO (to link it with glVertexAttribPointer), no need to fill it; the VBO data already contains everything we need.
- We configure the vertex attributes (only position data) for the light.
- We unbind the VAO to prevent accidental modifications.

For the skybox:

- We configure the Vertex Array Object (VAO) and Vertex Buffer Object (VBO) for the skybox.
- We generate the Vertex Array Object (VAO) and Vertex Buffer Object (VBO) for the skybox.
- We bind the Vertex Array Object (VAO) for the skybox for configuration.
- We bind the Vertex Buffer Object (VBO) for the skybox and fill it with the vertex data.
- We enable the position attribute of the skybox vertices.
- We configure the vertex attributes (only position data) for the 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);
```

Here's the breakdown of the steps involved in loading the skybox textures and creating the projection matrix:

- We load the textures for the skybox.
- We create a vector of const GLchar* to store the paths of the skybox textures.
- We add the paths of the textures to the vector.
- We load the cubemapTexture using the LoadCubemap function of the TextureLoading class.
- We pass the vector of texture paths as an argument to the function.
- We create the projection matrix using the perspective function of glm.
- The projection matrix is used to transform the vertices into projection coordinates.
- We pass the field of view (camera zoom), aspect ratio, near plane distance, and far plane distance as arguments to the perspective function.

```
// 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);
```

Here's a breakdown of the main game loop:

- The main game loop begins.
- We calculate the elapsed time between frames to obtain deltaTime.
- We update lastFrame with the value of currentFrame.
- We check for active events (key presses, mouse movement, etc.) and call the corresponding response functions.
- We use glfwPollEvents to process any pending events.
- We clear the color buffer and the depth buffer.
- We set the clear color with glClearColor and clear the buffers with glClear.
- We enable depth testing with glEnable(GL_DEPTH_TEST). This allows OpenGL to perform depth testing to determine which fragments are drawn in front and which are occluded.
- We use the "lightingShader" shader to set the uniforms and draw the objects.
- We get the camera position and set it as "viewPos" in the shader.
- We set the directional light direction and its properties (ambient, diffuse, specular) in the shader.
- We calculate the color of point light 1 using the time and the Light1 components.
- We set the position, color, and properties of point light 1 in the 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
```

- We configure the SpotLight in the shader.
- We set the position of the spot light as the camera position.
- We set the direction of the spot light as the front direction of the camera.
- We set the ambient, diffuse, and specular properties of the spot light to zero.
- We set the constant, linear, and quadratic properties of the spot light to zero.
- We configure the cut-off angles of the spot light using the glm::cos and glm::radians functions.
- We set the material properties in the shader, such as shininess.
- We create the camera transformations by obtaining the view matrix with camera.GetViewMatrix. We get the locations of the uniforms in the shader for model, view, and projection.
- We pass the matrices to the shader using glUniformMatrix4fv for the view and projection matrices.
- We bind the VAO and assign an identity matrix (temp) to the 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);
```

*- Loading the house model:

- We obtain the view matrix using camera.GetViewMatrix.
- We create a model matrix and initialize it as an identity matrix.
- We apply a translation transformation to the model matrix to position the house in the scene.
- We pass the model matrix to the shader using glUniformMatrix4fv.
- We draw the facade model of the house using the Draw method of the Fachada object.
- We repeat the same process for the following objects.

```
// 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);
```

To draw the lamp object:

- We use the lamp shader (lampShader).
- We obtain the uniform locations for the matrices in the lamp shader.
- We set the matrices (view, projection, and model) in the lamp shader using glUniformMatrix4fv.
- We apply a translation transformation to the model matrix to position the lamp in the scene.
- We pass the model matrix to the shader using glUniformMatrix4fv.

- We draw the lamp object using the Draw method of the SV object.

To draw the animation object using the Anim2 shader:

- We obtain the uniform locations for the matrices in the animation shader.
- We set the matrices (view, projection, and model) in the animation shader using glUniformMatrix4fv.
- We apply a rotation and translation transformation to the model matrix to position the bush in the scene.
- We pass the model matrix to the shader using glUniformMatrix4fv.
- We pass the current time to the shader using glUniform1f.
- We draw the bush object using the Draw method of the Arbusto object.

```
        // 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;
```

To draw the skybox at the end:

- We change the depth function (depthFunc) to pass the depth test when values are equal to the content of the depth buffer.
- We use the skybox shader (SkyBoxshader).
- We remove any translation component from the view matrix (view) so that the skybox is fixed at the camera's position.
- We set the matrices (view and projection) in the skybox shader using glUniformMatrix4fv.

- We bind the VAO (skyboxVAO) and activate the cubemap texture (cubemapTexture).
- We draw the skybox triangles using glDrawArrays.
- We restore the depth function to its default value.

We swap the screen buffers.

- We delete the used VAO, VBO, and EBO.
- We terminate GLFW, freeing any resources allocated by GLFW. The main function ends and returns 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()

- Move/alter the camera position based on user input. Camera Control:
- If the W key or up arrow is pressed, move the camera forward.
- If the S key or down arrow is pressed, move the camera backward.
- If the A key or left arrow is pressed, move the camera to the left.
- If the D key or right arrow is pressed, move the camera to the right. Animations:
- Greeting Animation:
- If the "anim" flag is true:
- If the "anim2" flag is false and "rotStan" is less than 20.0f, increment "rotStan" by 0.5f.
- If the "anim2" flag is true and "rotStan" is greater than 0.0f, decrement "rotStan" by 0.5f.
- If "rotStan" is greater than or equal to 20.0f, set the "anim2" flag to true.
- If "rotStan" is less than or equal to 0.0f, set the "anim2" flag to false. Door Closing Animation:
- If the "anim3" flag is true:
- If the "Open" flag is false and "rotPuerta" is greater than -90.0f, decrement "rotPuerta" by 0.5f.

- If the "Open" flag is true and "rotPuerta" is less than 0.0f, increment "rotPuerta" by 0.5f.
- If "rotPuerta" is less than or equal to -90.0f, set the "anim3" flag to false.
- If "rotPuerta" is greater than or equal to 0.0f, set the "anim3" flag to false. Drawer Animation:
- If the "CajonAbierto" flag is true:
- If the "CajonAbrir" flag is true, increment "movCajon" by 0.01.
- If "movCajon" is greater than or equal to 0.4f, set the "CajonAbrir" flag to false.
- If the "CajonCerrar" flag is true:
- If the "CajonCerrar" flag is true, decrement "movCajon" by 0.01.
- If "movCajon" is less than or equal to 0.02f, set the "CajonCerrar" flag to false.

```
// 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;

        }

}
```


```
}
//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)

- Called when a key is pressed/released via GLFW. Parameters:
- window: Pointer to the current GLFW window.
- key: Code of the pressed/released key.
- scancode: Platform-specific key code.
- action: Action performed on the key (pressed or released).
- mode: Mode of the key (Shift, Ctrl, Alt, Super). Escape:
- If the Escape key (GLFW_KEY_ESCAPE) is pressed and the action is GLFW_PRESS, the GLFW window is closed. Key Control:
- If the key has a valid code (key >= 0 and key < 1024):
- If the action is GLFW_PRESS, the corresponding key in the "keys" array is set to true.
- If the action is GLFW_RELEASE, the corresponding key in the "keys" array is set to false. Animation Activation:
- If the Z key (GLFW_KEY_Z) is pressed, Animation 1 is activated by setting the "anim" flag to true.
- If the X key (GLFW_KEY_X) is pressed, Animation 2 is activated by toggling the value of the "Open" variable and setting the "anim3" flag to true.
- If the C key (GLFW_KEY_C) is pressed, Animation 3 is activated by setting the "CajonAbierto" flag to true.
- If "movCajon" is greater than or equal to 0.4f, the "CajonCerrar" flag is set to true and the "CajonAbrir" flag is set to false.
- If "movCajon" is less than or equal to 0.02f, the "CajonCerrar" flag is set to false and the "CajonAbrir" flag is set to true.

```
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)

- Called when the mouse is moved inside the GLFW window. Parameters:
- window: Pointer to the current GLFW window.
- xPos: X position of the mouse cursor.
- yPos: Y position of the mouse cursor. Workflow: First Mouse Move:
- If it's the first time the mouse is moved, the current cursor position is stored in the "lastX" and "lastY" variables, and the "firstMouse" flag is set to false. Mouse Offset Calculation:
- The mouse offset in the X-axis is calculated and stored in the "xOffset" variable (difference between the current and previous positions).
- The mouse offset in the Y-axis is calculated and stored in the "yOffset" variable (difference between the previous and current positions, inverted because Y coordinates go from bottom to top).
- The "lastX" and "lastY" variables are updated with the current cursor positions. Processing Mouse Movement in the Camera:
- The "ProcessMouseMovement" function of the camera is called, passing the "xOffset" and "yOffset" as arguments.

8-Conclusions

During the execution of this project, we encountered some details and complications. We faced some challenges in improving certain aspects and implementing certain features. We struggled with time management and focused more on delivering a work that met the evaluation criteria. On the other hand, the experience gained from this project reflects the reality that developers often face: being self-taught, learning under time constraints, and following a strict schedule to meet all the requirements demanded by a client.

We acquired a substantial amount of knowledge, most of which we applied directly during the practical work. The modeling part was particularly enjoyable and dynamic, as we had the opportunity to represent various objects and create a favorite character. Texturing the models to achieve a cartoon-like appearance was not overly complicated, but it did require a significant amount of effort.

Regarding the limitations, we discussed the issues we encountered. While we consider them to be relatively minor, we recognize that they pale in comparison to the challenges faced in real-world projects. Nevertheless, we are satisfied with what we have achieved in this project and the overall practice. Ultimately, we successfully achieved our main objective and applied the knowledge we acquired throughout the project.

9-References

- 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&_sS archString=skybox