TÉCNICAL MANUAL
FINAL PROJECT

Graphic Computing and Human-Computer Interaction Laboratory

National Autonomous University of Mexico

Faculty of Engineering

- 31803532-7
- 16/Mayo/2024

DevDream Solutions

Table of Contents

Objectives

- **Application of Knowledge:** Demonstrate the practical application of acquired knowledge during the course."

- **3D Recreation:** Select a real or fictional facade and space, and recreate them in 3D using OpenGL, ensuring that virtual objects closely resemble their reference images.

Project Scope

- **Space Selection:** The project will involve the selection and recreation of a facade and a space with at least 7 distinct objects.

- **Ambiance:** The recreation should capture the essence of the environment from the reference images.

- **Bilingual Delivery:** Project documentation will be delivered in Spanish and English, avoiding fully automatic translation.

Limitations

- **Individuality:** The project must be carried out and delivered individually.

- **Evaluation:** Projects with assessments below 5 will be considered deficient and will affect the final course grade.

- **Object Repetition:** Recreated objects that are repeated will be counted as a single object in the evaluation.

- **Mandatory Elements:** Doors, windows, chimneys, and stairs are mandatory elements and are not counted as objects for evaluation.

- **Base Code:** The provided base code must be used throughout the project.

Constraints

Delivery Constraints

- **Official Repository:** Project delivery must be done exclusively through a repository on GitHub. Any delivery outside of this platform will result in project nullification.

Creative Constraints

- **Prohibited Themes:** Recreation of spaces belonging to UNAM or specific themes such as The Simpsons, Rick and Morty, SpongeBob SquarePants, Kame House from Dragon Ball, Courage the Cowardly Dog's house, Minecraft content, and The Powerpuff Girls' house are not allowed.

- **Graphic Quality:** Creating 3D environments with low graphic characteristics, similar to those of games developed for consoles like PS1, PS2, Xbox, Nintendo 64, etc., is prohibited.

Animation Considerations

- **Animation Context:** Animations must have a purpose and be in harmony with the context of the recreated space. Simply rotating an object without reason is not sufficient.

- **Animation Complexity:** A complex animation should not be merely linear or consist solely of basic transformations of the object.

Resources

For efficiently and effectively developing this project, I've utilized various applications that have allowed me to manage all necessary tasks and aspects. Below, I present a detailed list of the tools I've used and how each has contributed to the success of this project.

OpenGL

OpenGL (Open Graphics Library) is a cross-platform, open-source API (Application Programming Interface) for rendering 2D and 3D graphics. It's widely used in the development of graphic applications such as video games, simulators, scientific visualization applications, among others. It's the tool that will allow us to render our project.

Maya 2023

Autodesk Maya is a 3D modeling, animation, rendering, and simulation software widely used in the entertainment industry, especially in film production, animation, video games, and visual effects. It offers a wide range of tools and features for creating complex and high-quality digital content. It was one of the most used tools throughout the development of the generated models.

Visual Studio Community 2022

The Visual Studio IDE is a creative starting point that can be used to edit, debug, and compile code, and then publish an application. It was the most used tool in the development of the project. Blocks of code were created and modified based on the necessary requirements requested, using the C++ language.

Paint 3D.

Paint 3D is a Microsoft application preinstalled in Windows 10 and later versions. While based on the classic Paint program, Paint 3D is a modernized version that allows users to create two and three-dimensional images easily and accessibly. It was used to develop some textures that needed to be created manually.

GIMP

GIMP (GNU Image Manipulation Program) is a powerful open-source and free image editing software available for various platforms, including Windows, macOS, and Linux. It allowed us to properly adjust the size of our textures and make modifications to play with transparencies.

Luma AI Genie

It's a revolutionary tool that transforms text into 3D models. This 3D generative model by Luma AI has the capability to create interactive three-dimensional models from instructions given in natural language. It was used to create some objects that weren't easy to recreate.

Ganttpro

It's an online project management tool based on Gantt charts. It's used by over 350,000 project managers, team leaders, executives, and other managers across various fields. It was used for the development of the Gantt chart.

Trello

Trello

It's a project management tool that utilizes the Kanban system. Known for its visual and intuitive interface, it allows teams to efficiently organize and monitor tasks and projects. It was used to implement the software methodology.

Genshin Impact

GENSHIN IMPACT

The Serenitea Pot is a special feature within the game Genshin Impact that allows players to create and customize their own living space, similar to a housing system. It was introduced in version 1.5 of the game. It was used as a creative source for the project.

GitHub

GitHub

GitHub is a collaborative development platform for storing, sharing, and working on code. It's widely used by programmers and companies for project management and software version control; it utilizes Git to store code repositories in the cloud. It was used for the purpose of delivering the official repository.

Git

git

It's a distributed version control system that allows developers to track and manage changes to source code over time. It works locally on the user's computer, allowing work on projects even without an internet connection.

Software Methodology

"Kanban" is a combination of two Japanese words: 看 (Kàn), which means "sign" or "visual signal," and 板 (Bǎn), which means "board." In Toyota, Kanban cards were paper cards indicating the need for a new product, a new part, or inventory, triggering the production process for that item.

In Kanban implemented for software development, teams start with a list of pending tasks. Work is "pulled" from the pending tasks based on the workload and capacity of each team member. Then, team members can visually track the work as it progresses through the task lifecycle, represented by stages on a Kanban board, until completion. In its current configuration, Kanban operates as a visual project management method that allows teams to find a balance between workload demand and team resource availability.

A Kanban board is a management tool that helps visualize work, limit work in progress (WIP), and maximize efficiency. Teams use the board to structure their workload and manage progress in real-time.

The concept of a Kanban board is simple. Each column represents a different stage of your workflow or project. As the task progresses, it moves to the corresponding column.

List of Activities First Phase.

Número de ED	Nombre de tarea / Título	Asignado a	Fecha de inici	Fecha de final	Fecha límite
1	Primera Fase Proyecto		22/02/2024	10/05/2024	10/05/2024
2	Búsqueda de Imagen de Referencia	Jimenez Cerv	22/02/2024	29/02/2024	29/02/2024
3	Búsqueda de Texturas	Jimenez Cerv	01/03/2024	07/03/2024	
4	Desarrollo de la Fachada	Jimenez Cerv	07/03/2024	22/04/2024	
5	Desarrollo de Objeto 01	Jimenez Cerv	07/03/2024	25/03/2024	
6	Desarrollo Objeto 02	Jimenez Cerv	14/03/2024	19/04/2024	
7	Desarrollo Objeto 03	Jimenez Cerv	21/03/2024	19/04/2024	
8	Desarrollo Objeto 04	Jimenez Cerv	28/03/2024	19/04/2024	
9	Desarrollo Objeto 05	Jimenez Cerv	28/03/2024	19/04/2024	
10	Desarrollo Objeto 06	Jimenez Cerv	28/03/2024	19/04/2024	
11	Desarrollo Objeto 07	Jimenez Cerv	28/03/2024	19/04/2024	
12	Desarrollo Objetos Extra 01	Jimenez Cerv	10/04/2024	22/04/2024	
13	Desarrollo Objeto Extra 02	Jimenez Cerv	10/04/2024	22/04/2024	
14	Desarrollo Objeto Extra 03	Jimenez Cerv	10/04/2024	22/04/2024	
15	Desarrollo Objeto Extra 04	Jimenez Cerv	10/04/2024	22/04/2024	
16	Desarrollo Objeto Extra 05	Jimenez Cerv	10/04/2024	22/04/2024	
17	Desarrollo Objeto Extra 06	Jimenez Cerv	10/04/2024	22/04/2024	
18	Animación Sencilla 01	Jimenez Cerv	22/04/2024	23/04/2024	
19	Animación Sencilla 02	Jimenez Cerv	22/04/2024	23/04/2024	
20	Animación Sencilla 03	Jimenez Cerv	23/04/2024	24/04/2024	
21	Animaciones sencillas extras	Jimenez Cerv	24/04/2024	29/04/2024	
22	Animación Compleja 01	Jimenez Cerv	29/04/2024	02/05/2024	
23	Animación Compleja 02	Jimenez Cerv	29/04/2024	02/05/2024	
24	Manual Técnico ES	Jimenez Cerv	29/04/2024	06/05/2024	
25	Manual Técnico EN	Jimenez Cerv	07/05/2024	07/05/2024	
26	Manual Usuario ES	Jimenez Cerv	29/04/2024	06/05/2024	
27	Manual Usuario EN	Jimenez Cerv	07/05/2024	07/05/2024	
28	Skybox	Jimenez Cerv	02/05/2024	06/05/2024	
29	Agregando Audio	Jimenez Cerv	06/05/2024	06/05/2024	
30	Ejecutable Final	Jimenez Cerv	06/05/2024	06/05/2024	08/05/2024

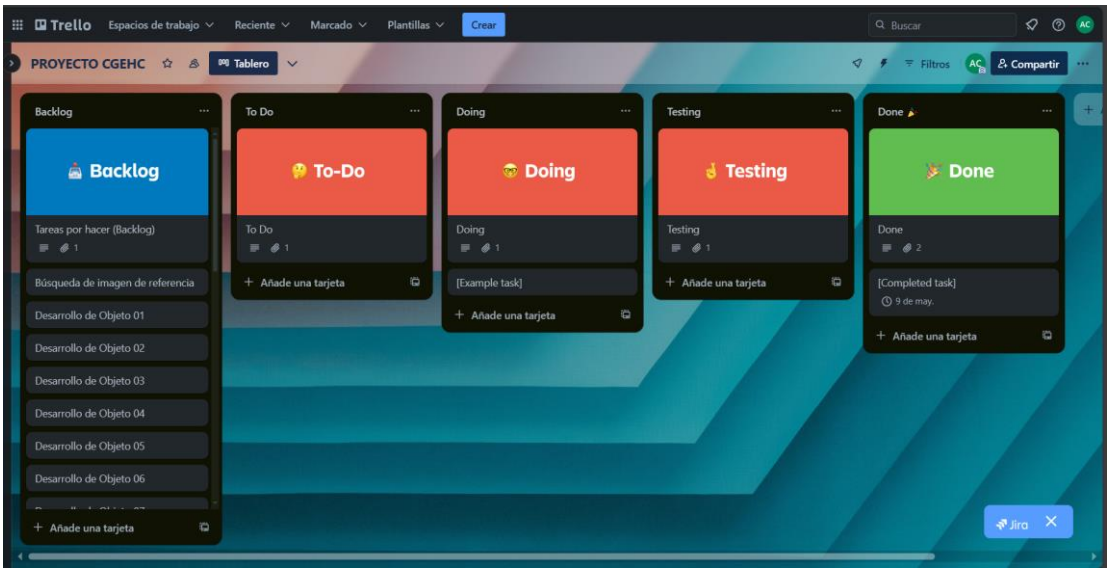
List of Activities Second Phase.

Cc	Número de ED	Nombre de tarea / Título	Asignado a	Fecha de inici	Fecha de final
	1	Tiempo Limite de Entrega	Jimenez Cerv	07/05/2024	20/05/2024
	2	Modificación Fachada Segundo Piso	Jimenez Cerv	07/05/2024	09/05/2024
	3	Modelo 01	Jimenez Cerv	08/05/2024	10/05/2024
	4	Modelo 02	Jimenez Cerv	08/05/2024	10/05/2024
	5	Modelo 03	Jimenez Cerv	08/05/2024	10/05/2024
	6	Modelo 04	Jimenez Cerv	08/05/2024	10/05/2024
	7	Modelo 05	Jimenez Cerv	08/05/2024	10/05/2024
	8	Modelo Extra 01	Jimenez Cerv	08/05/2024	10/05/2024
	9	Modelo Extra 02	Jimenez Cerv	08/05/2024	13/05/2024
	10	Modelo Extra 03	Jimenez Cerv	08/05/2024	13/05/2024
	11	Modelo Extra 04	Jimenez Cerv	08/05/2024	13/05/2024
	12	Modelo Extra 05	Jimenez Cerv	08/05/2024	15/05/2024
	13	Modelo Extra 06	Jimenez Cerv	08/05/2024	15/05/2024
	14	Animación Sencilla Extra	Jimenez Cerv	09/05/2024	10/05/2024
	15	Animación Sencilla Extra e iluminació	Jimenez Cerv	13/05/2024	15/05/2024
	16	Manual de Usuario ES-EN	Jimenez Cerv	13/05/2024	15/05/2024
	17	Manual Técnico ES-EN	Jimenez Cerv	15/05/2024	16/05/2024
	18	Ejecutable Final	Jimenez Cerv	15/05/2024	15/05/2024

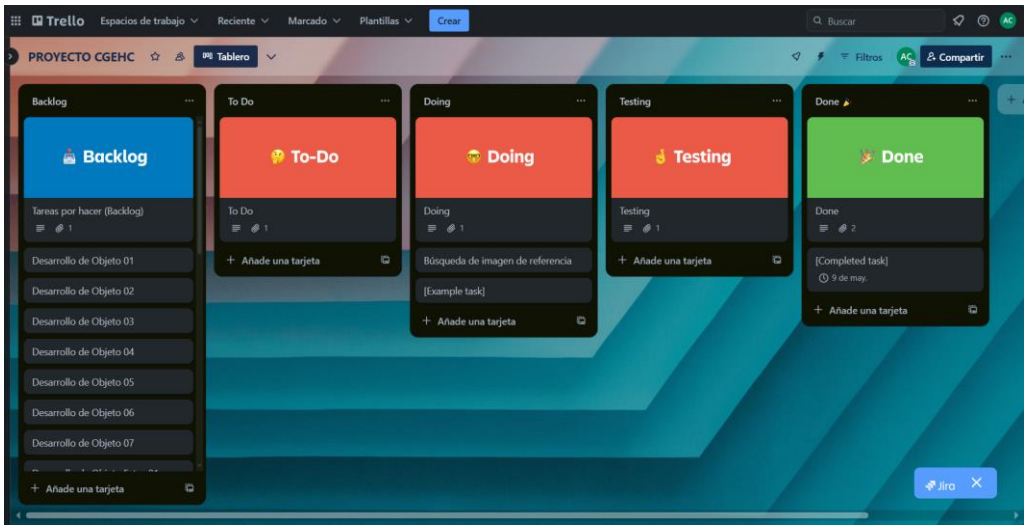
Kanban Board Progress

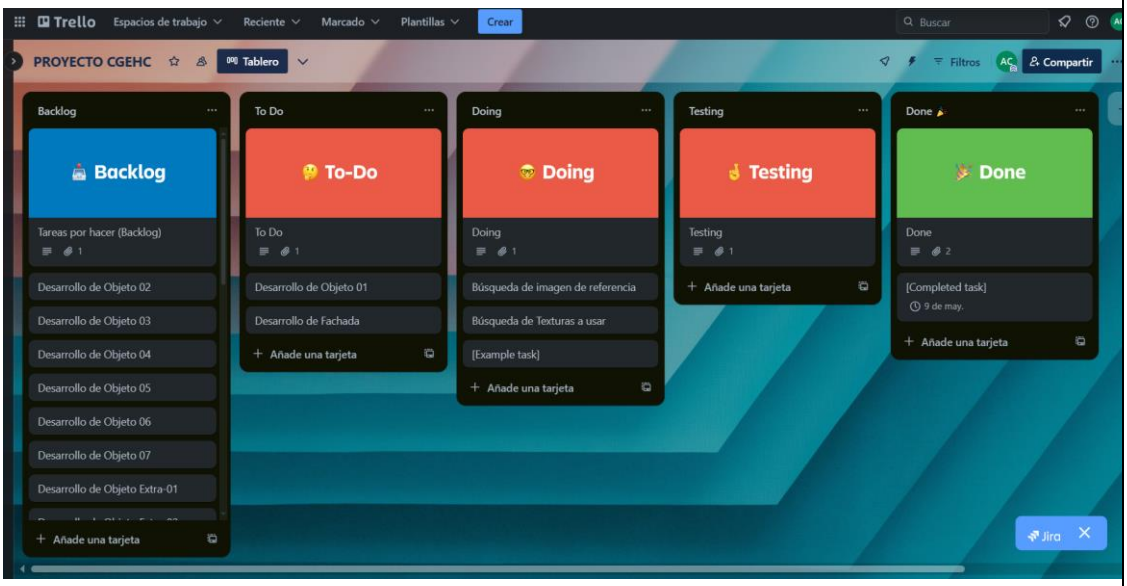
To avoid getting stuck in an ambiguous state of 'work in progress' (WIP), Thursday of each week was defined as the delivery day for the activity as demanded. Below, the progress of the Kanban board will be shown every Thursday since the publication of the reference document.

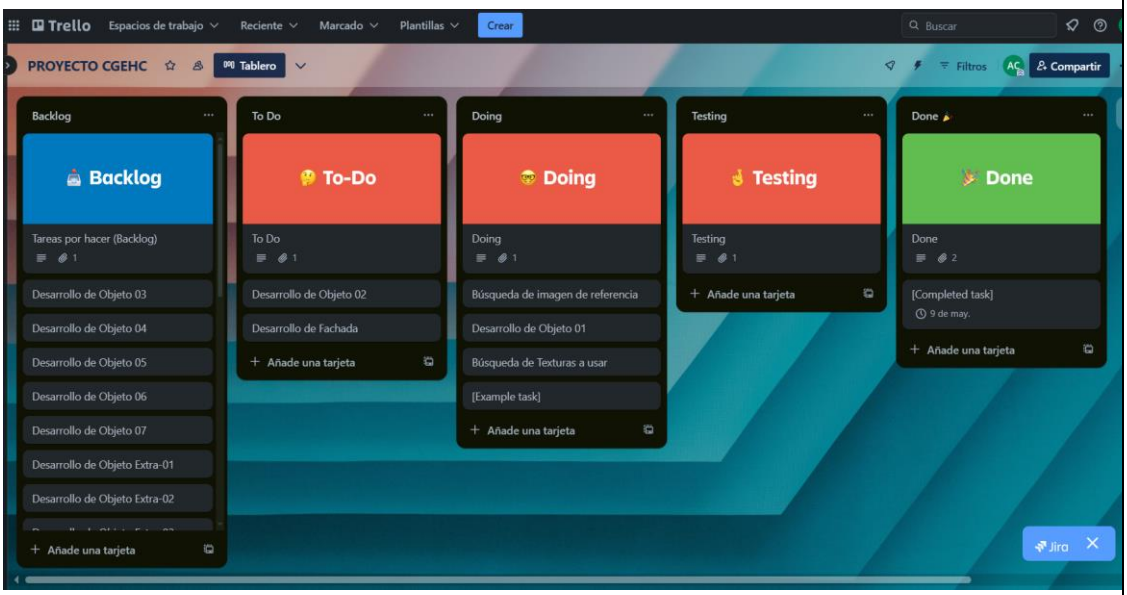
Jueves 22-febrero-2024
We begin to define the tasks to be carried out throughout the project.



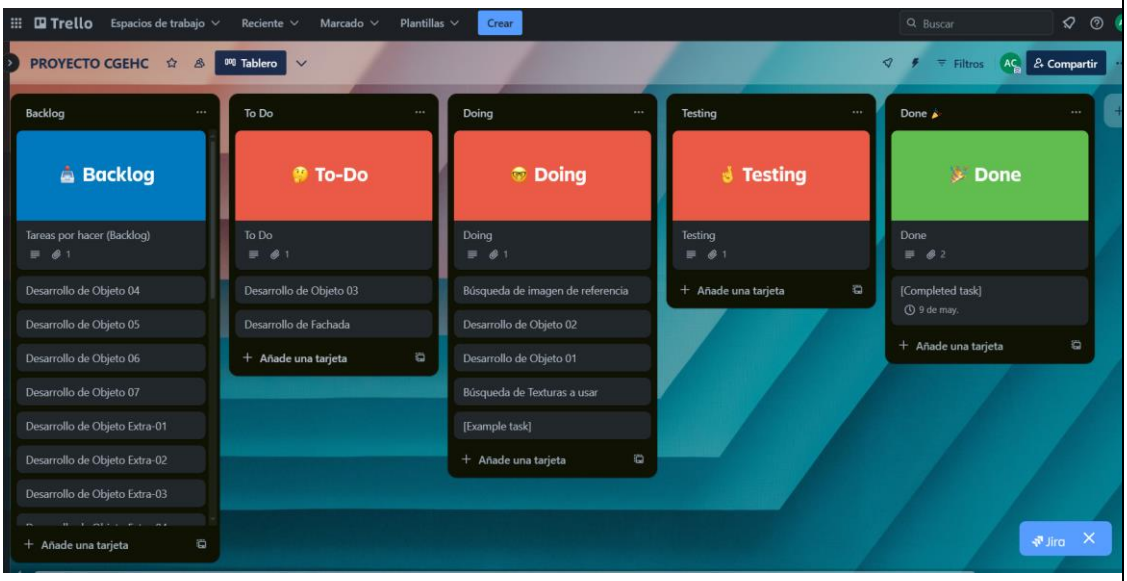
Jueves 29-febrero-2024
The first task was marked as completed, and the search for textures began.



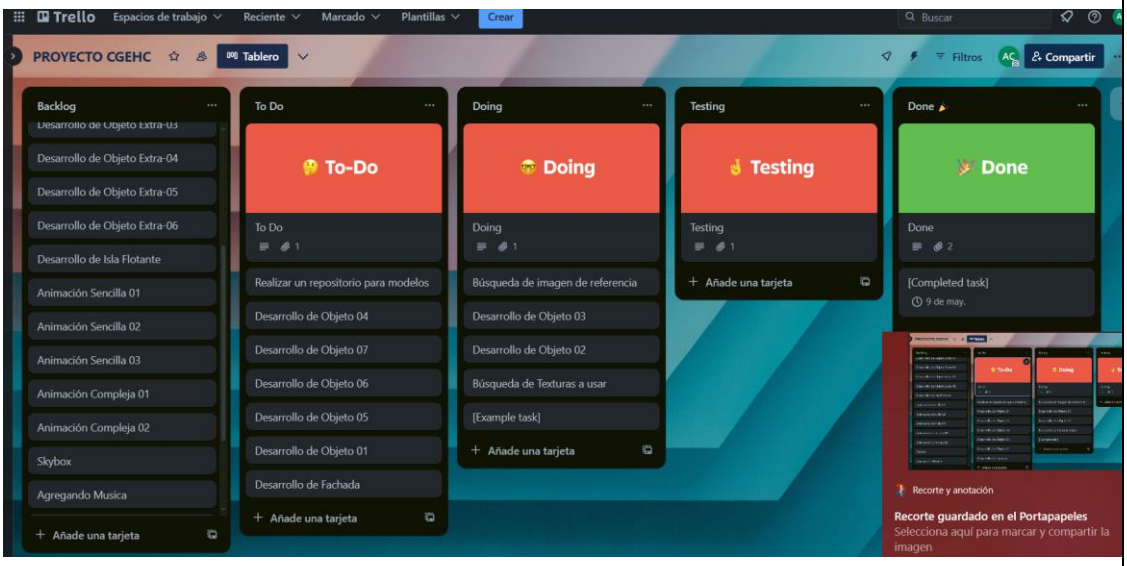
Jueves 07-marzo-2024
The texture search was marked as completed, and progress began on the facade and first object.


Jueves 14-marzo-2024
The progress of the first object was marked as completed, the second object was added, and work continues on the facade.


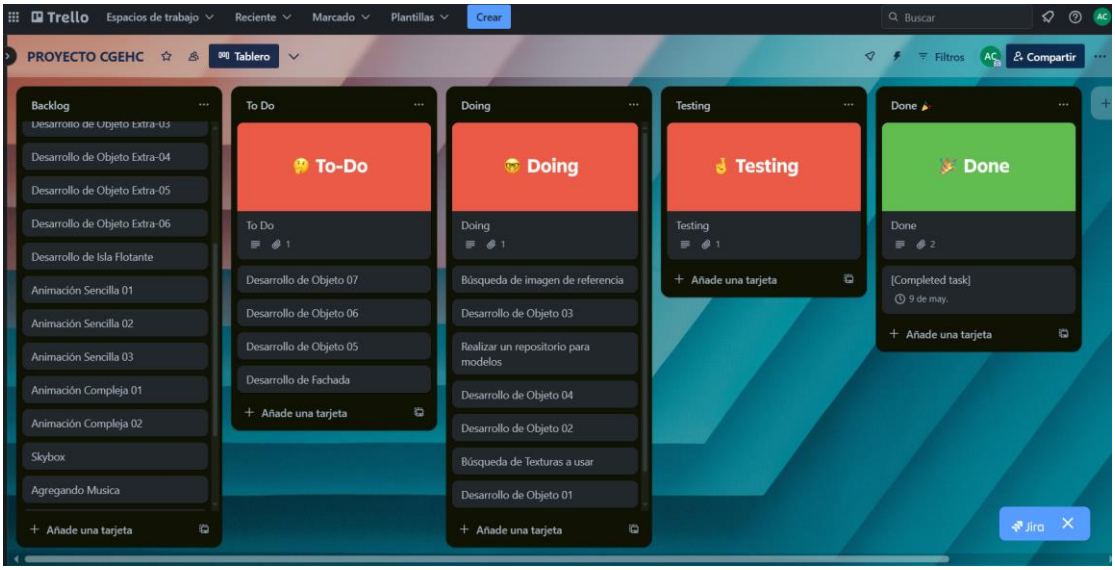
Jueves 21-marzo-2024
The progress of the second object was marked as completed, the third object was added, and work continues on the facade.



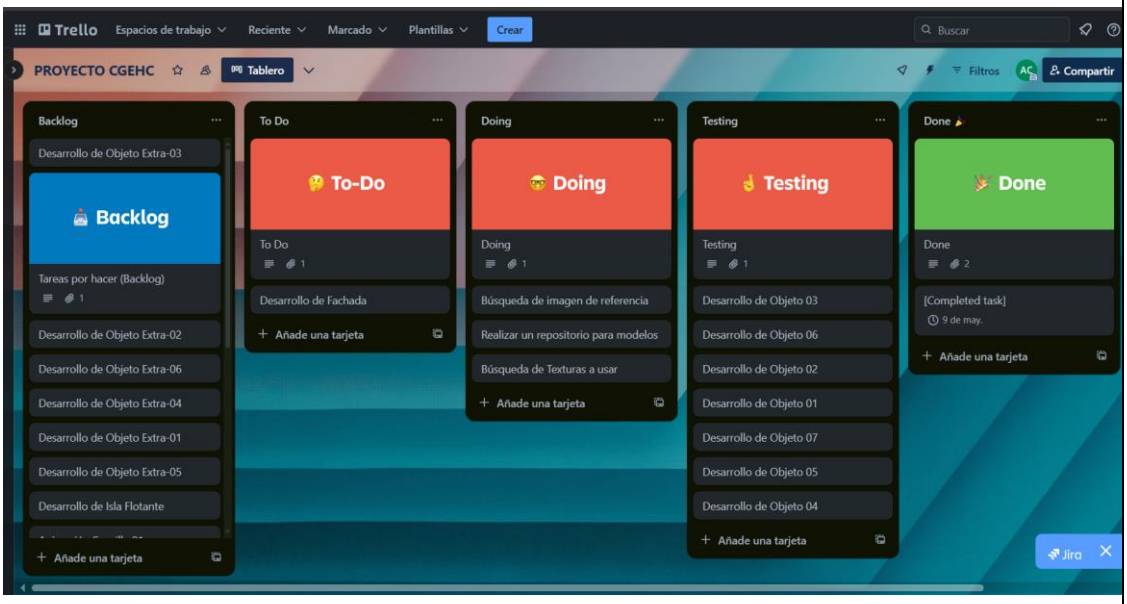
Jueves 28-marzo-2024
The progress of the third object was marked as completed. Work is ongoing on model 01 and subsequent models, and work continues on the facade. A repository is created for the progress of the models.



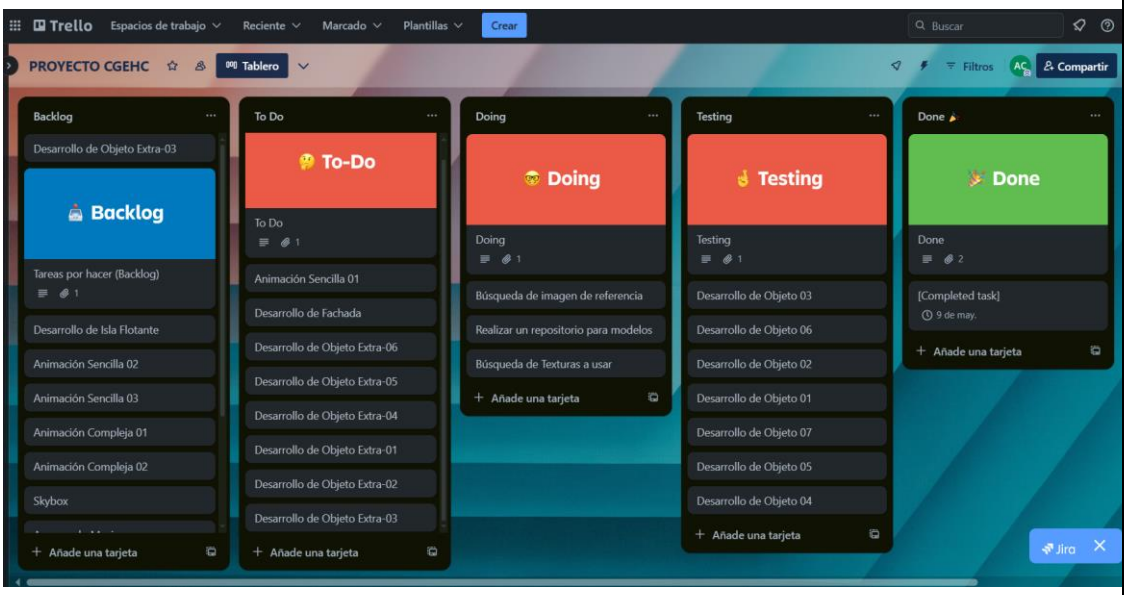
Jueves 04-abril -2024
The progress of the first models was marked as completed, and work continues on the facade.



Jueves 11-abril -2024
The progress of model 05 was marked as completed, and work continues on the facade. The finished models are sent to OpenGL.

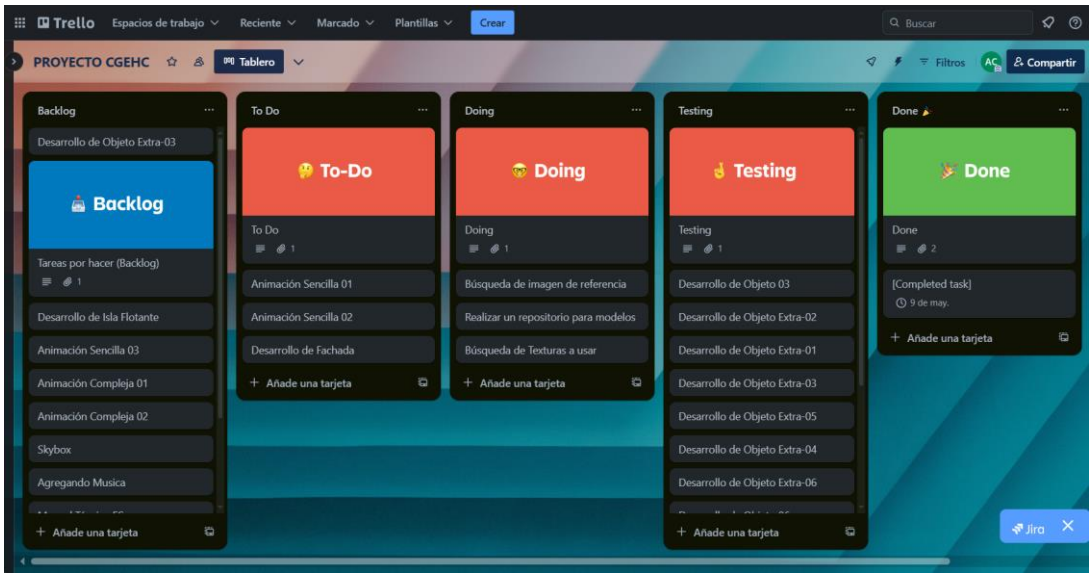


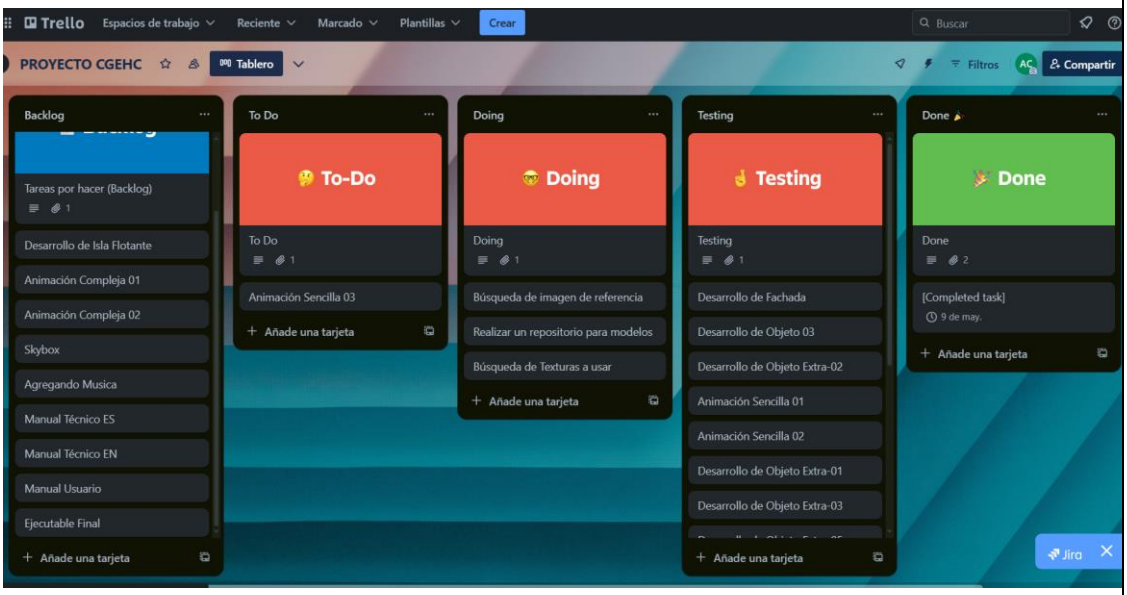
Jueves 18-abril -2024
The mandatory models were marked as completed, work continues on the facade and on the additional models. Work is also underway on the first simple animation.

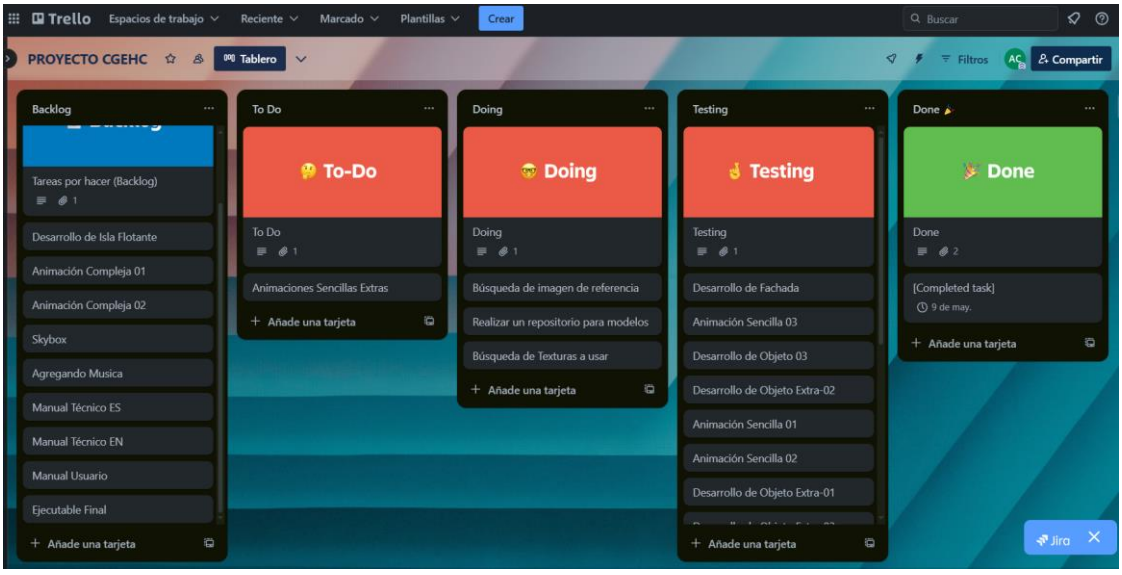


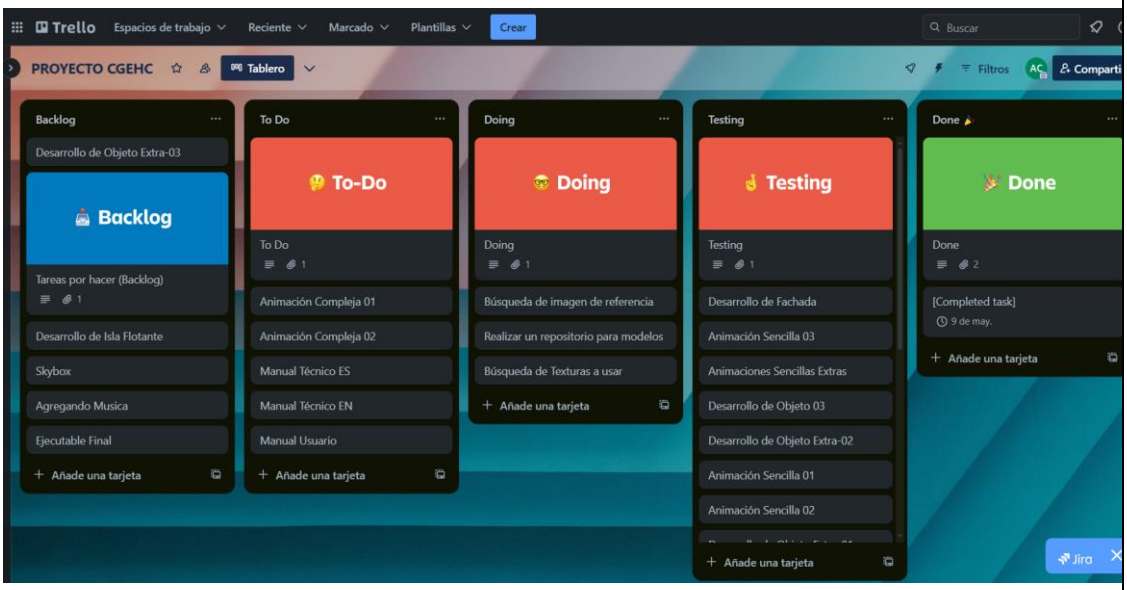
At this point, there was already a final repository for code delivery.

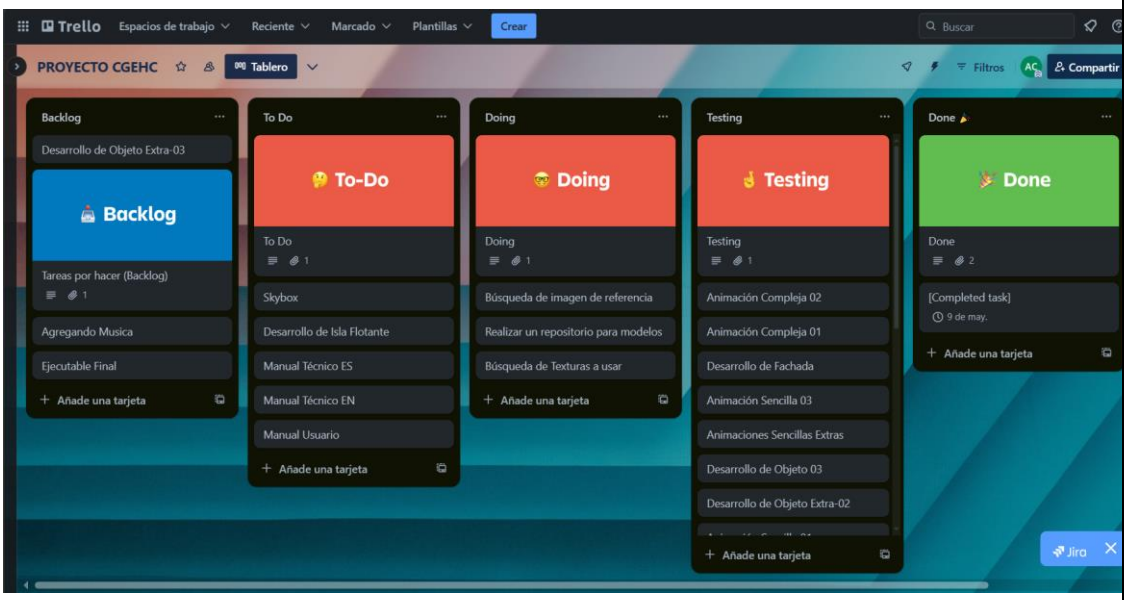
Domingo 21-abril -2024
All models were marked as completed, the facade was finally marked as completed, and work began on the first and second simple animations.



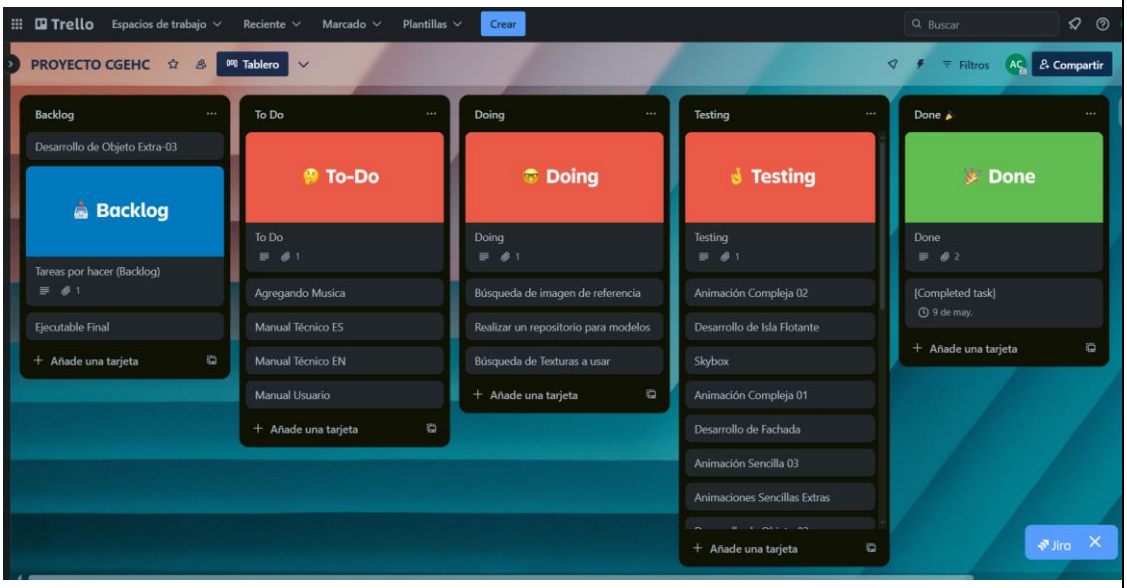
Lunes 22-abril -2024
The first and second simple animations are completed, and work begins on the third animation.


Martes 23-abril -2024
The third animation is completed, and work begins on an additional simple animation.


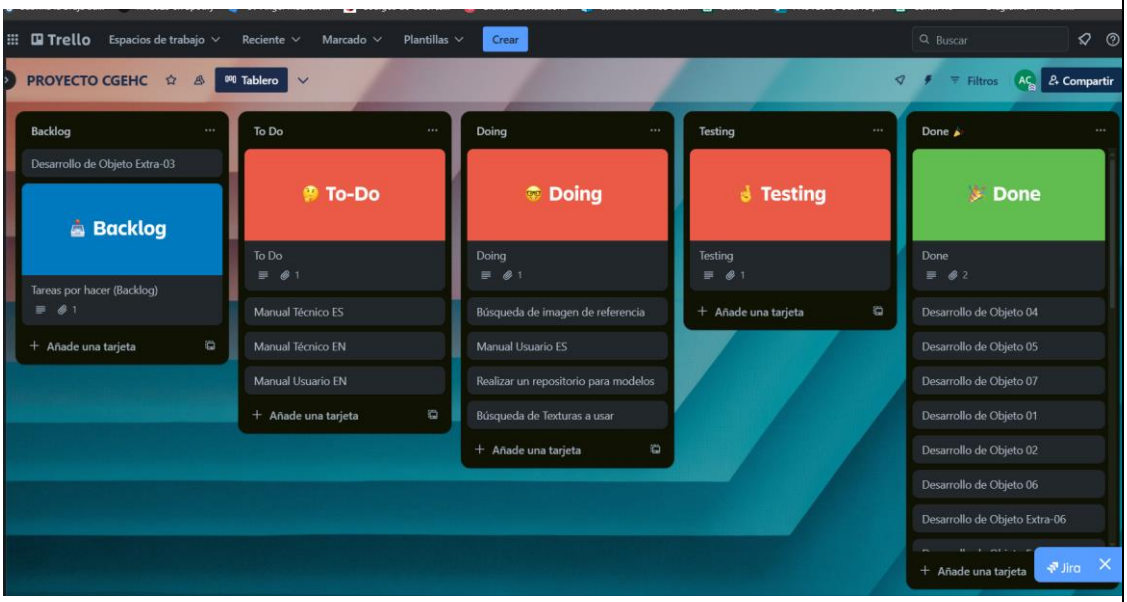
Domingo 28-abril -2024
The additional simple animation is completed, work begins on the complex animations. Work is also underway on the technical and user manuals.


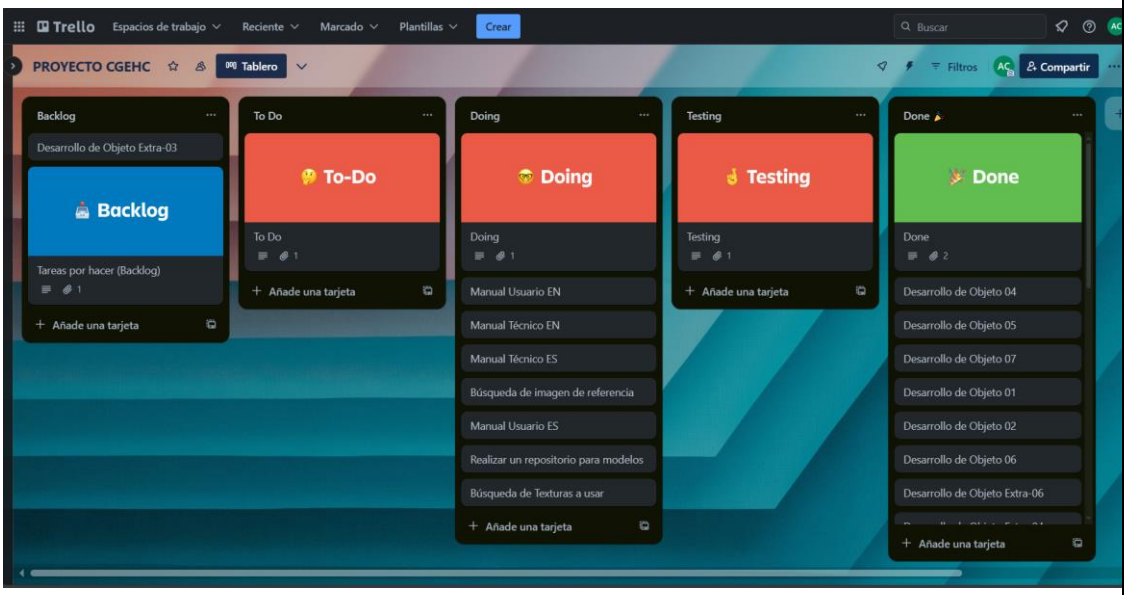
Jueves 02-mayo -2024
The additional simple animation, as well as the complex animations, are completed. Work begins on the skybox.


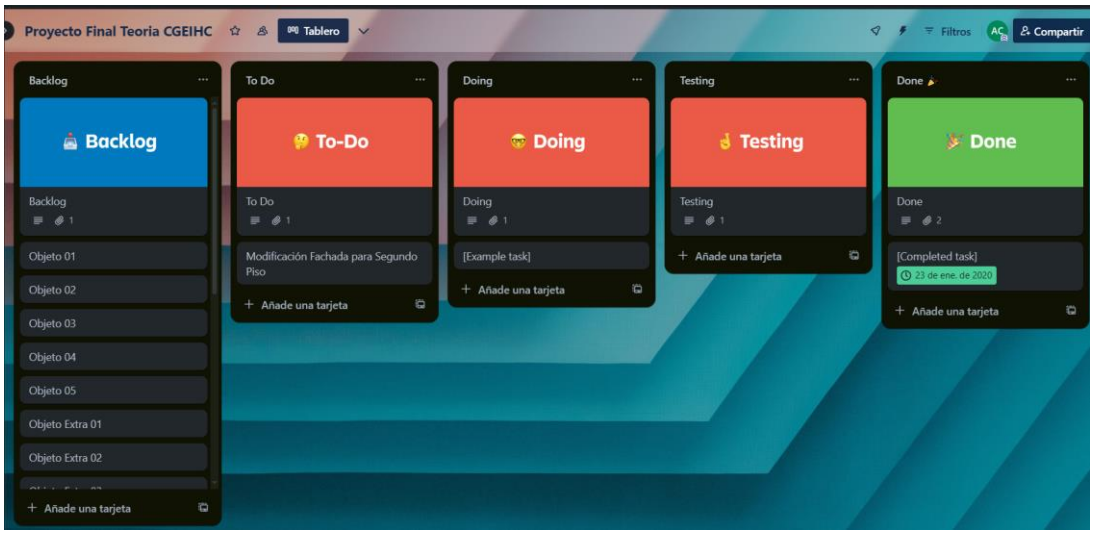
Domingo 05-mayo -2024
The skybox is completed, and work begins on the project's audio.



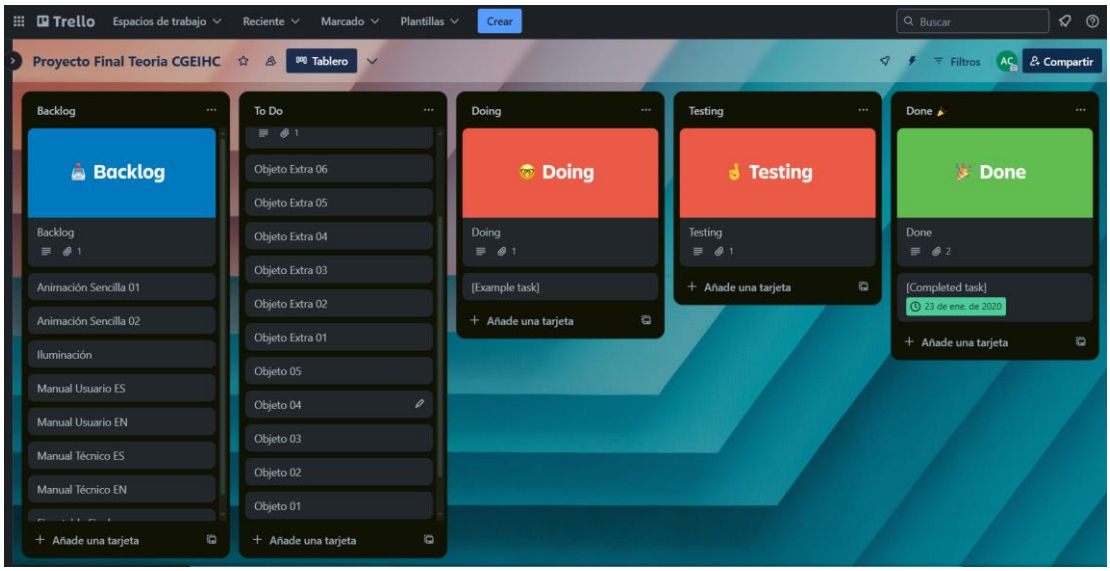
Lunes 06-mayo -2024
The audio insertion, obtaining the executable, and the Spanish user manual are completed.



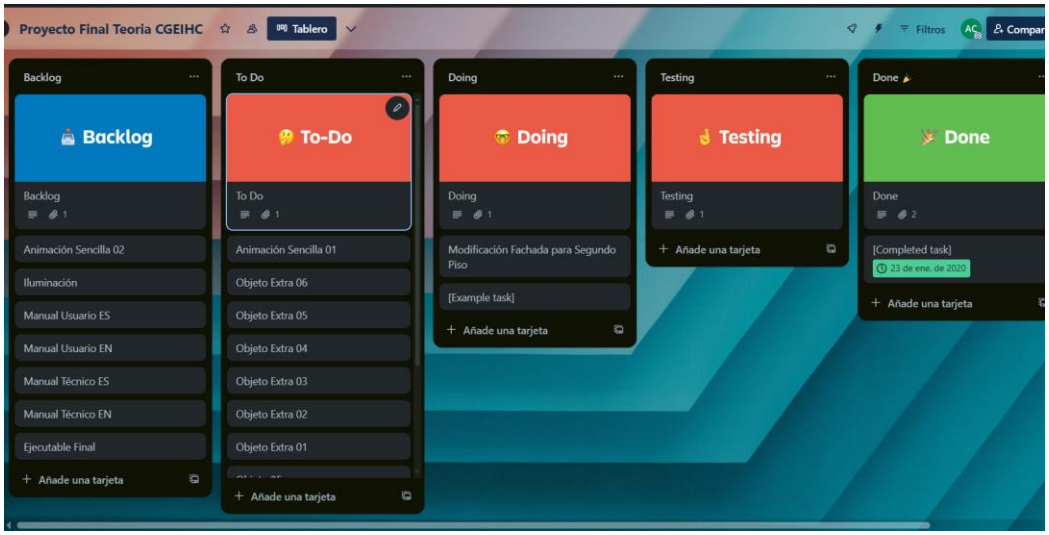
Martes 07-mayo -2024
The technical manual in Spanish and English, as well as the user manual in English, are completed.


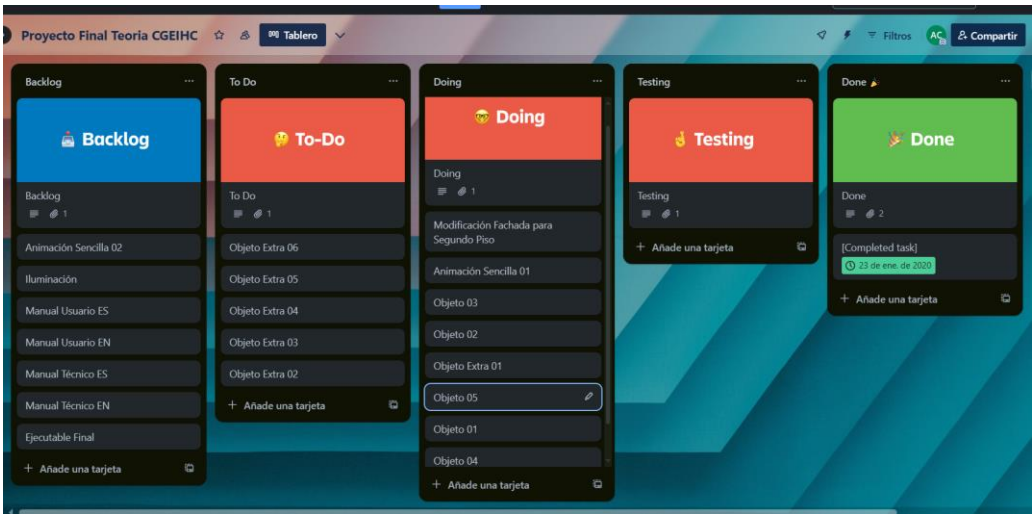
Martes 07-mayo -2024
Second phase activities are defined and work is done on the modification of the façade


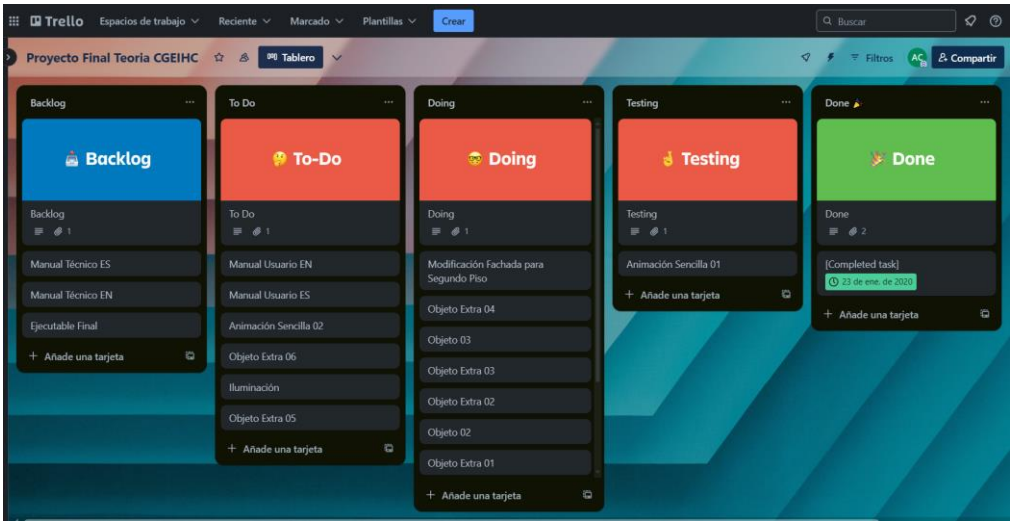
Miércoles 08-mayo -2024
Work continues on the façade and the other objects to be modelled are begun



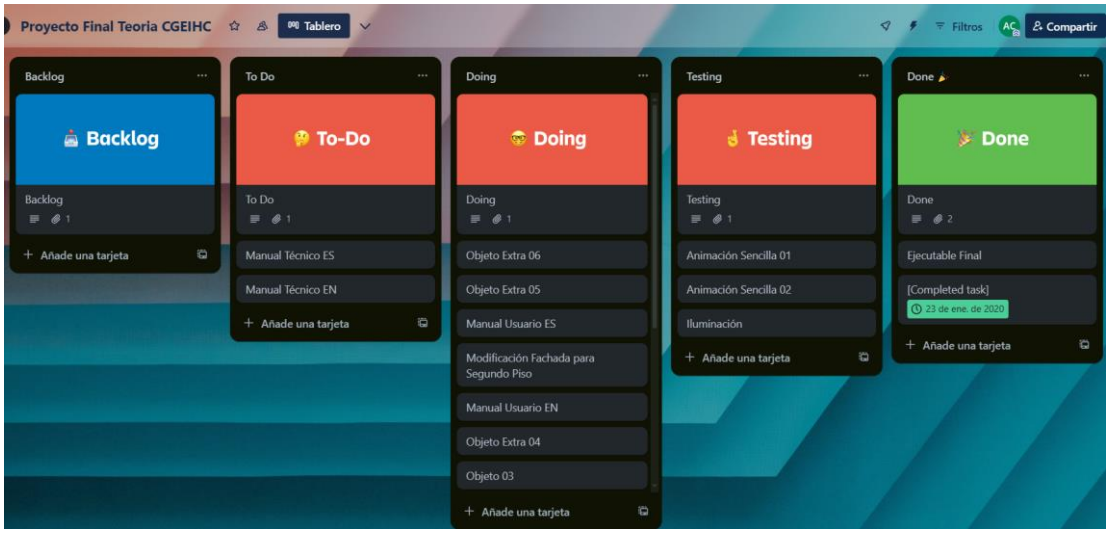
Jueves 09-mayo -2024
Work is done on the simple extra-01 animation and the modified façade is finished.



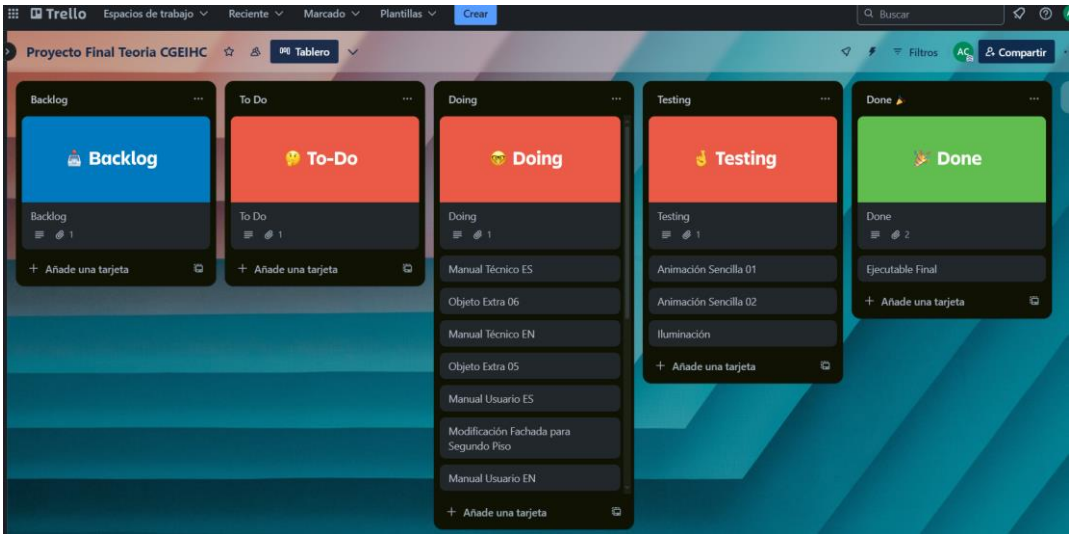
Viernes 10-mayo -2024
Simple Animation 01 and First Models Completed.


Lunes 13-mayo -2024
3 models are finished, work is done on the user manual, another simple animation and lighting is made.


Miercoles15-mayo -2024
The remaining models are finished, lighting and extra animation are finished; You work on the technical manual, and you get the executable.

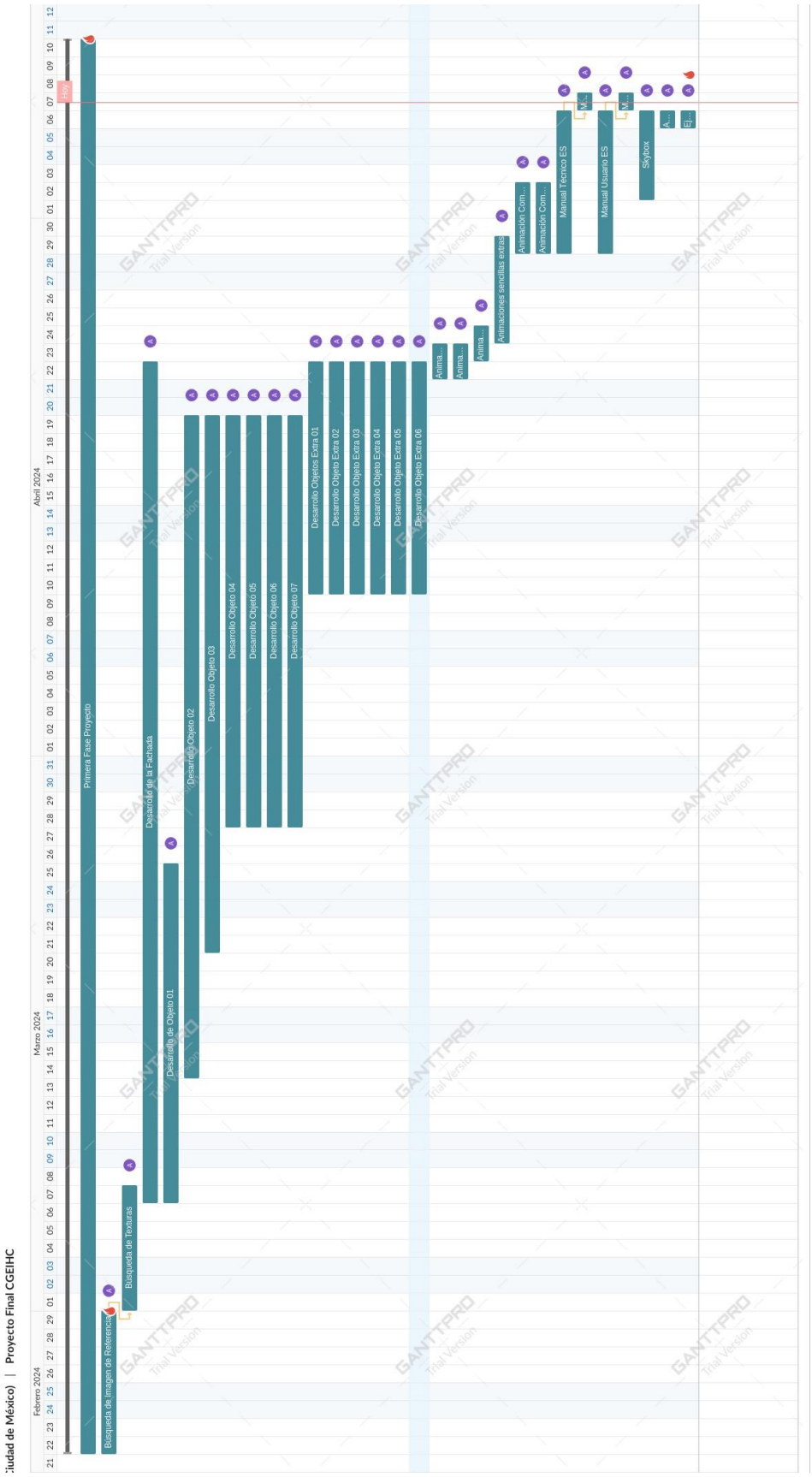


Jueves 16-mayo -2024
The technical manual and the project are completed

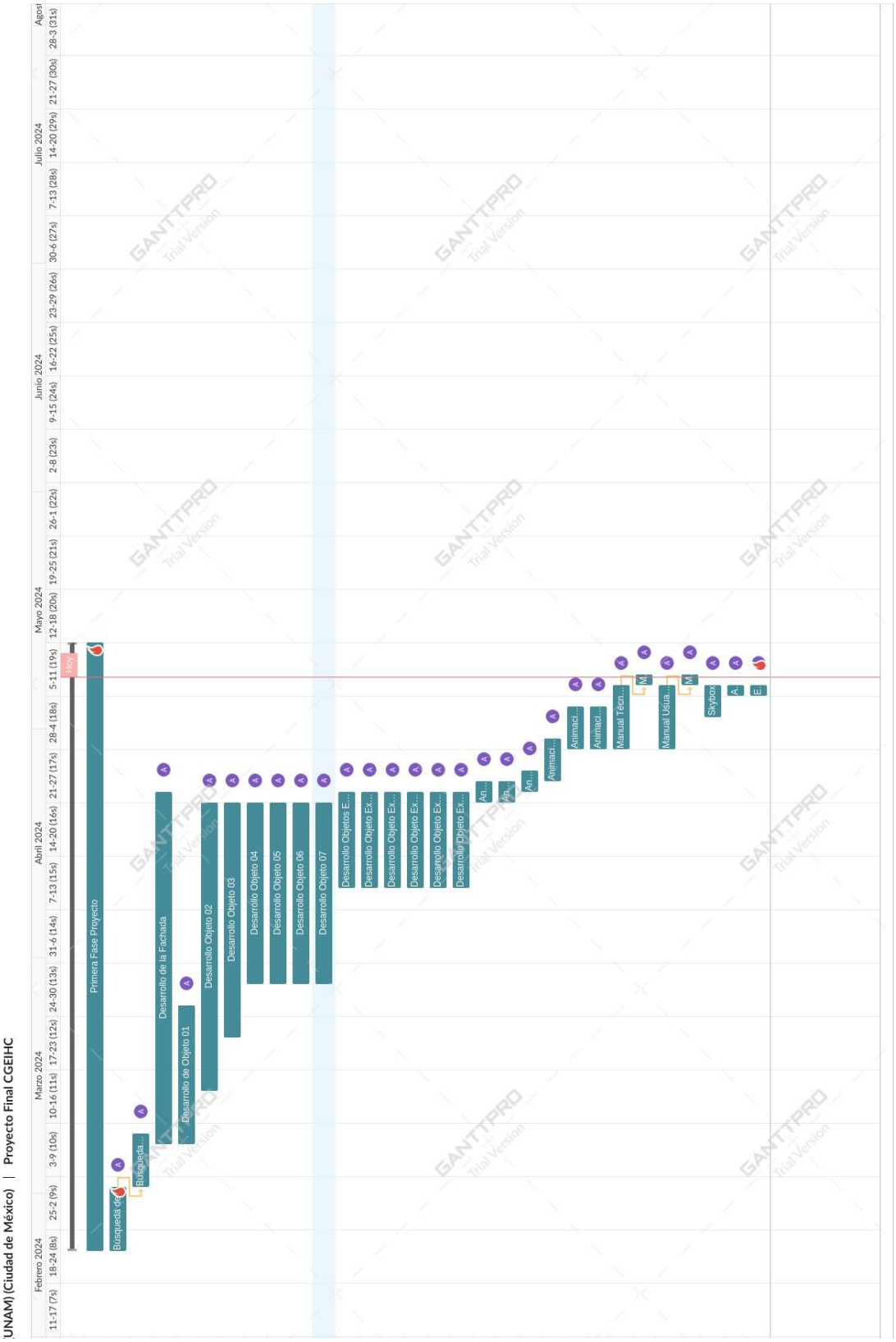


Gantt Chart First Phase

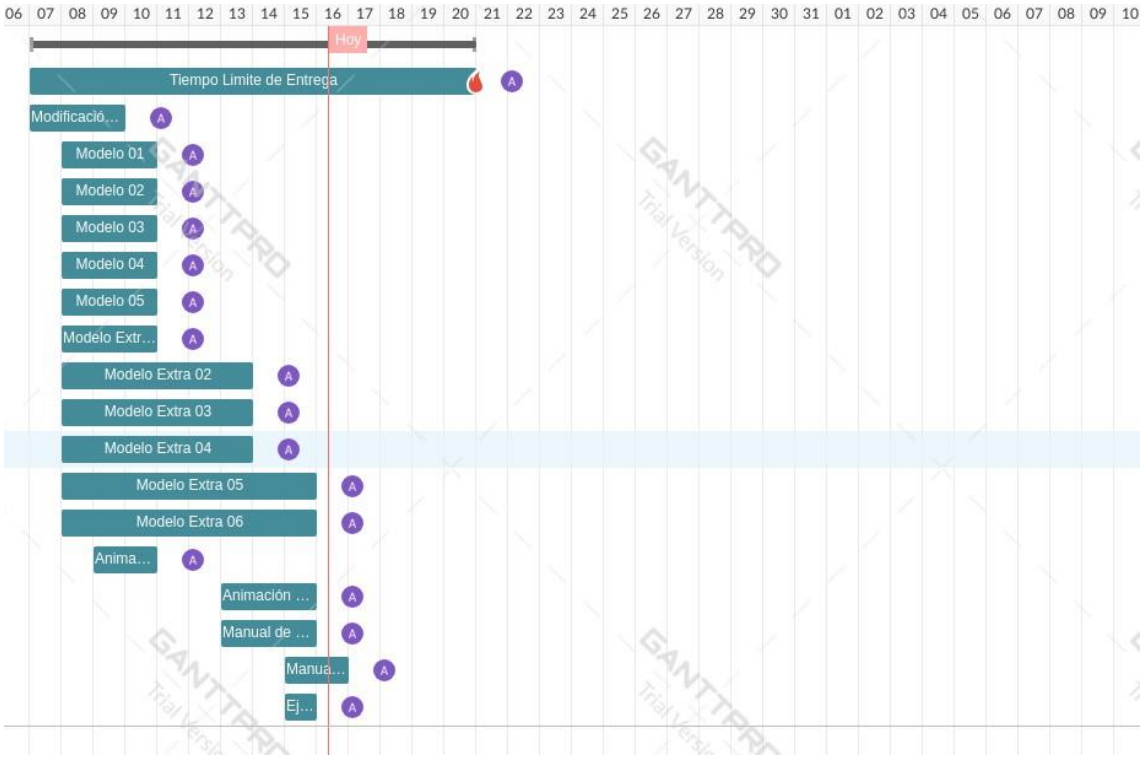
Gantt chart by days.



Gantt chart by weeks.



Gantt Chart Second Phase

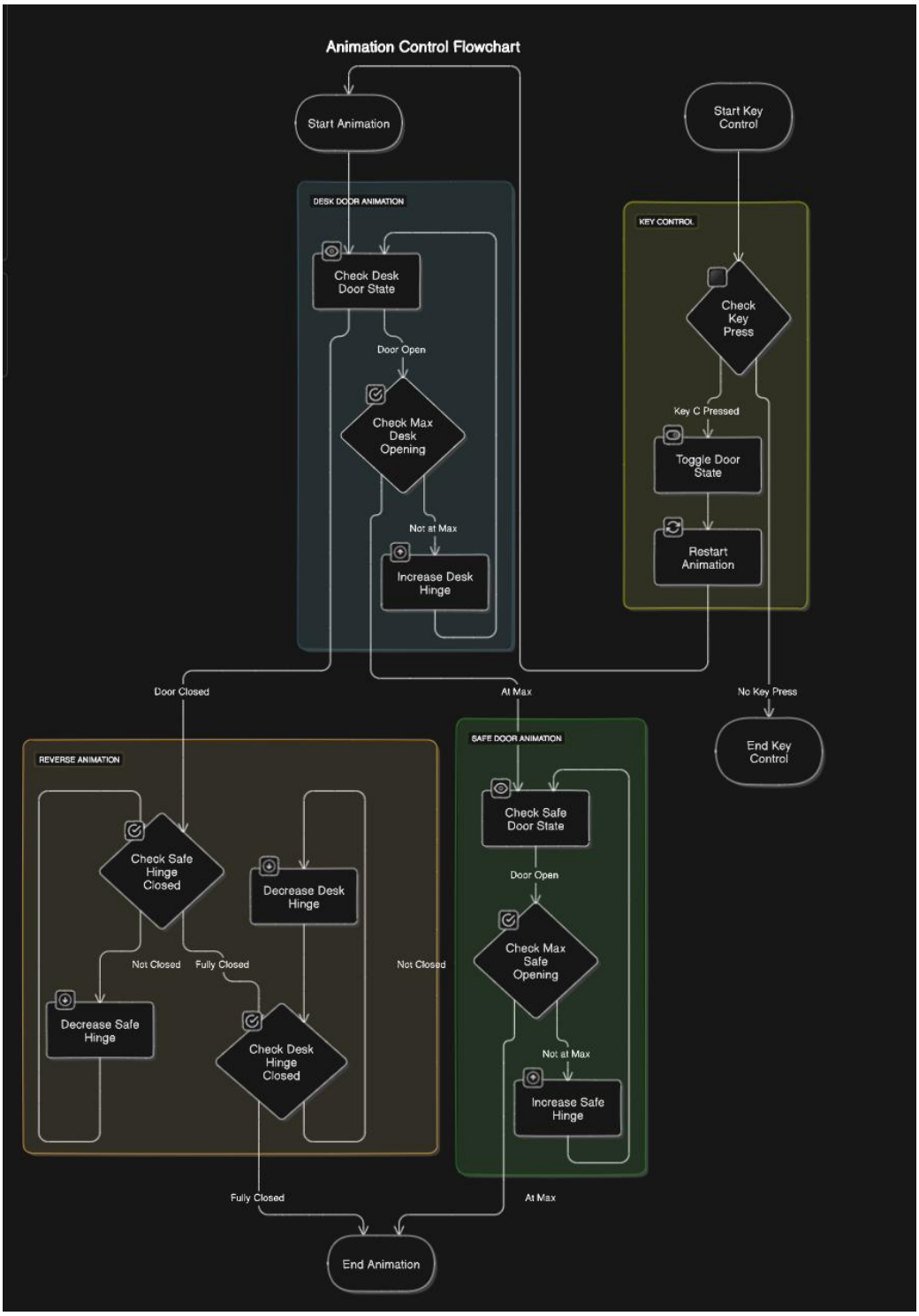


Software Flowchart

Simple Animation 01

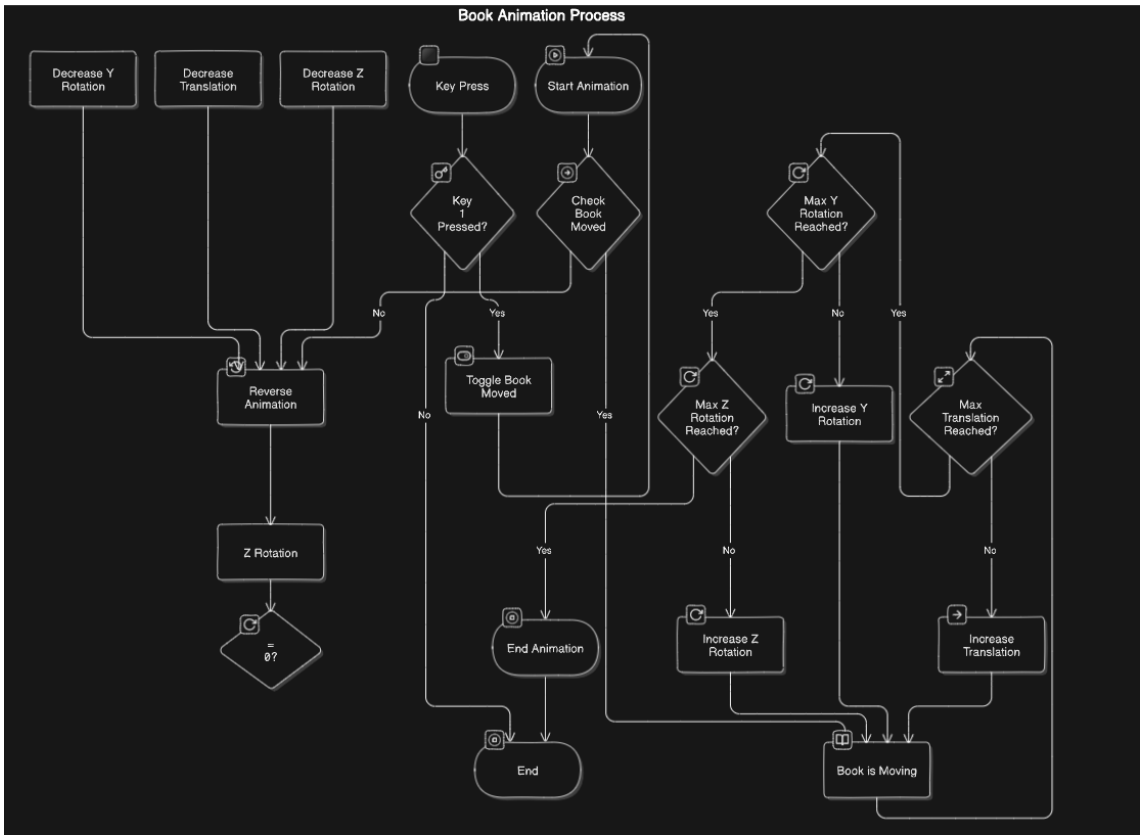
Here are grouped 3 animations based on the same logic, which consist of:

- Opening the front doors of the house
- Opening the lower doors of the bookcase
- Opening the desk door followed by the safe door.



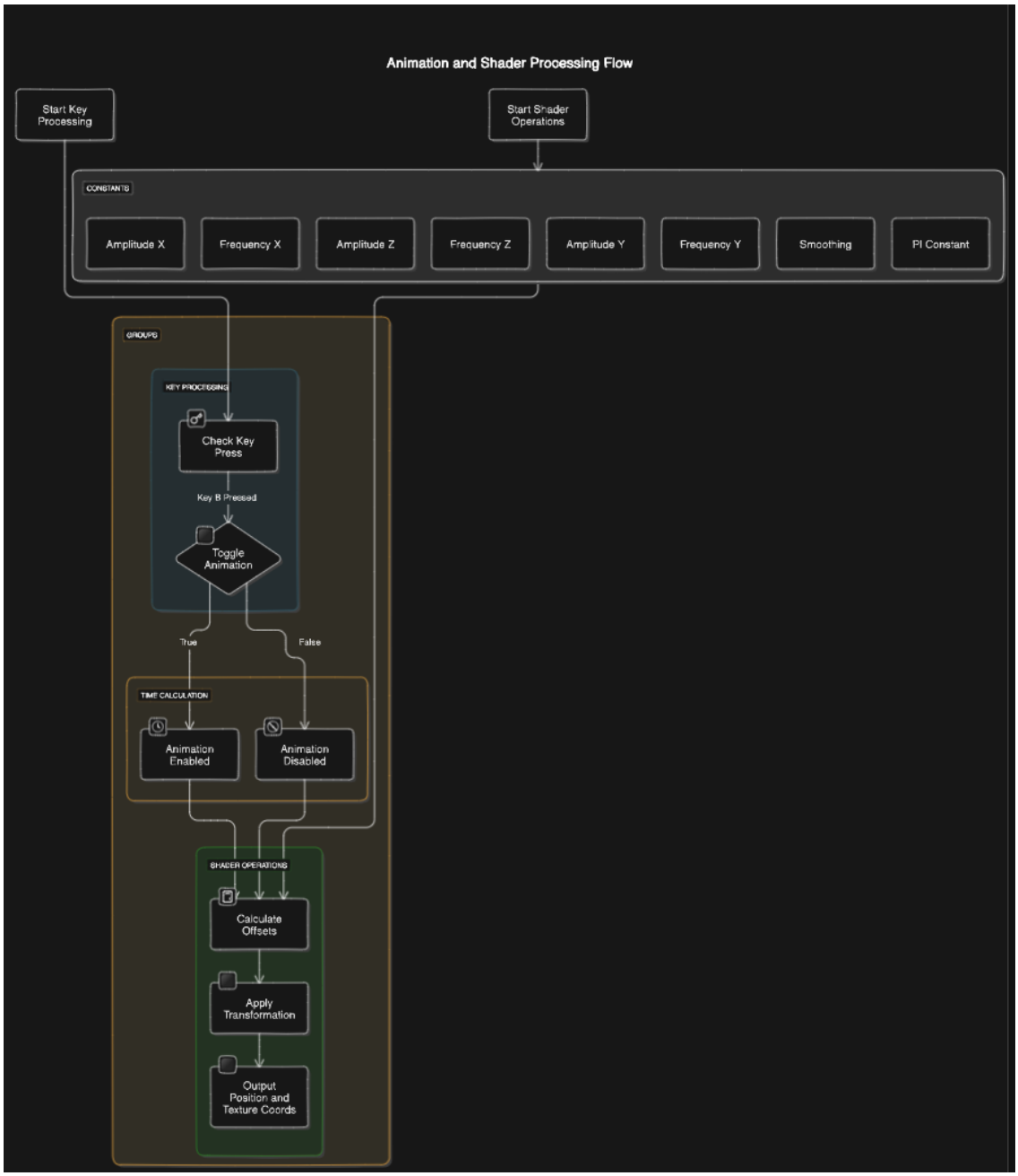
Simple Animation 02

This animation consists of picking up a book, rotating it to view the cover, and tilting it for easier observation.



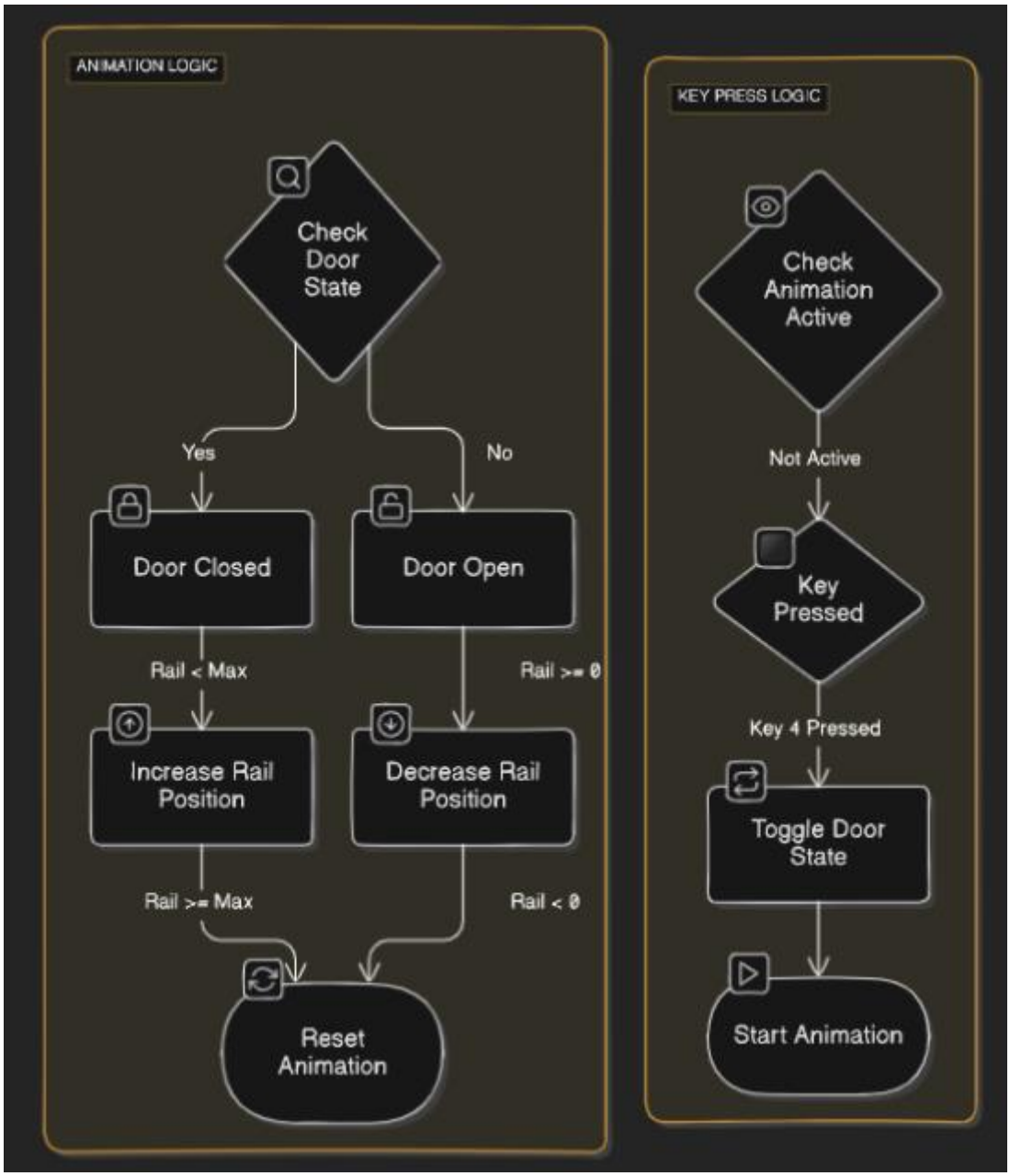
Simple Animation 03

This animation consists of simulating that the prototype (special item in the context of Genshin Impact) is floating.



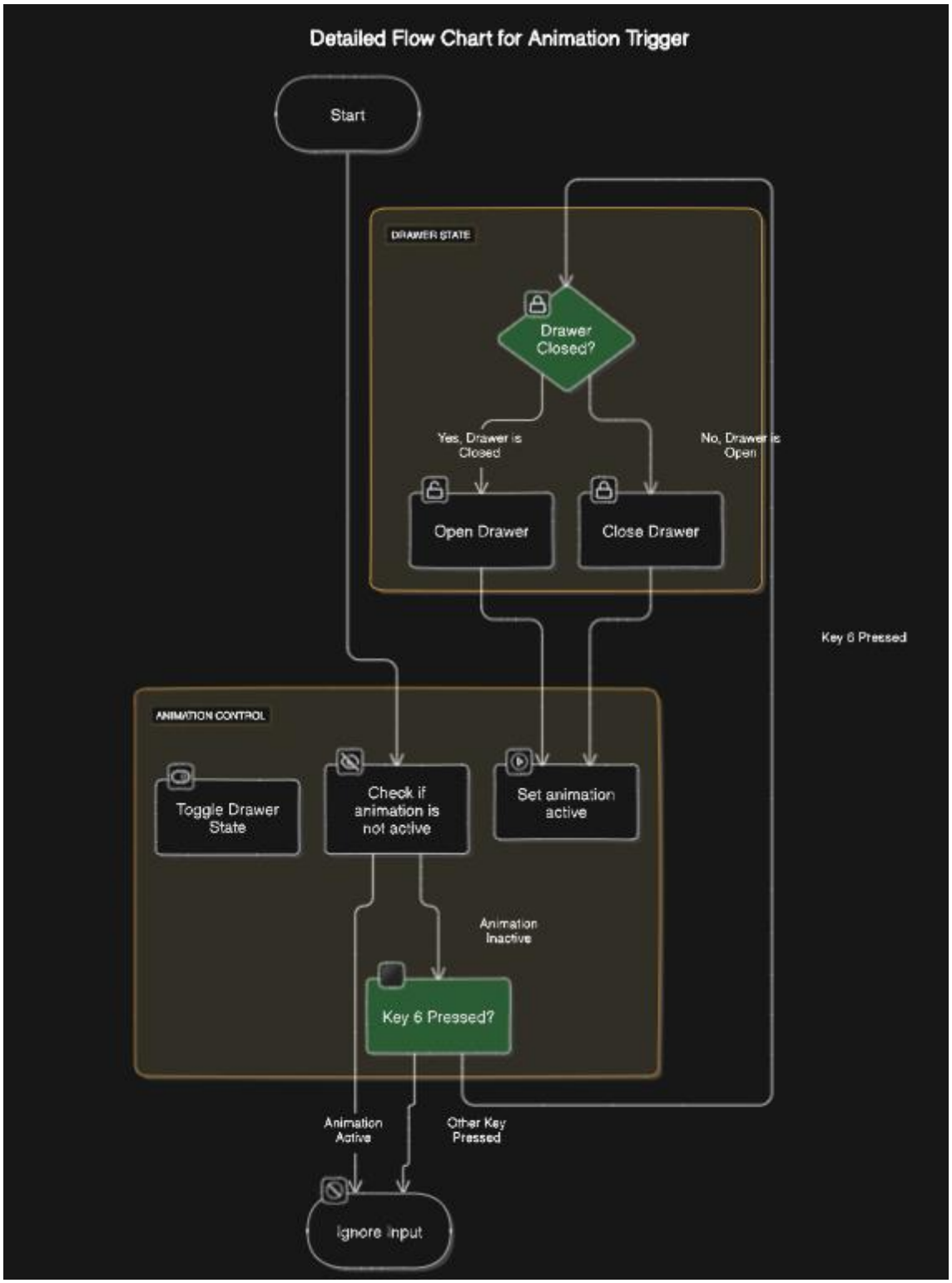
Simple Animation 04

This animation consists of opening the door to the second room, which has an oriental style.



Simple Animation 05

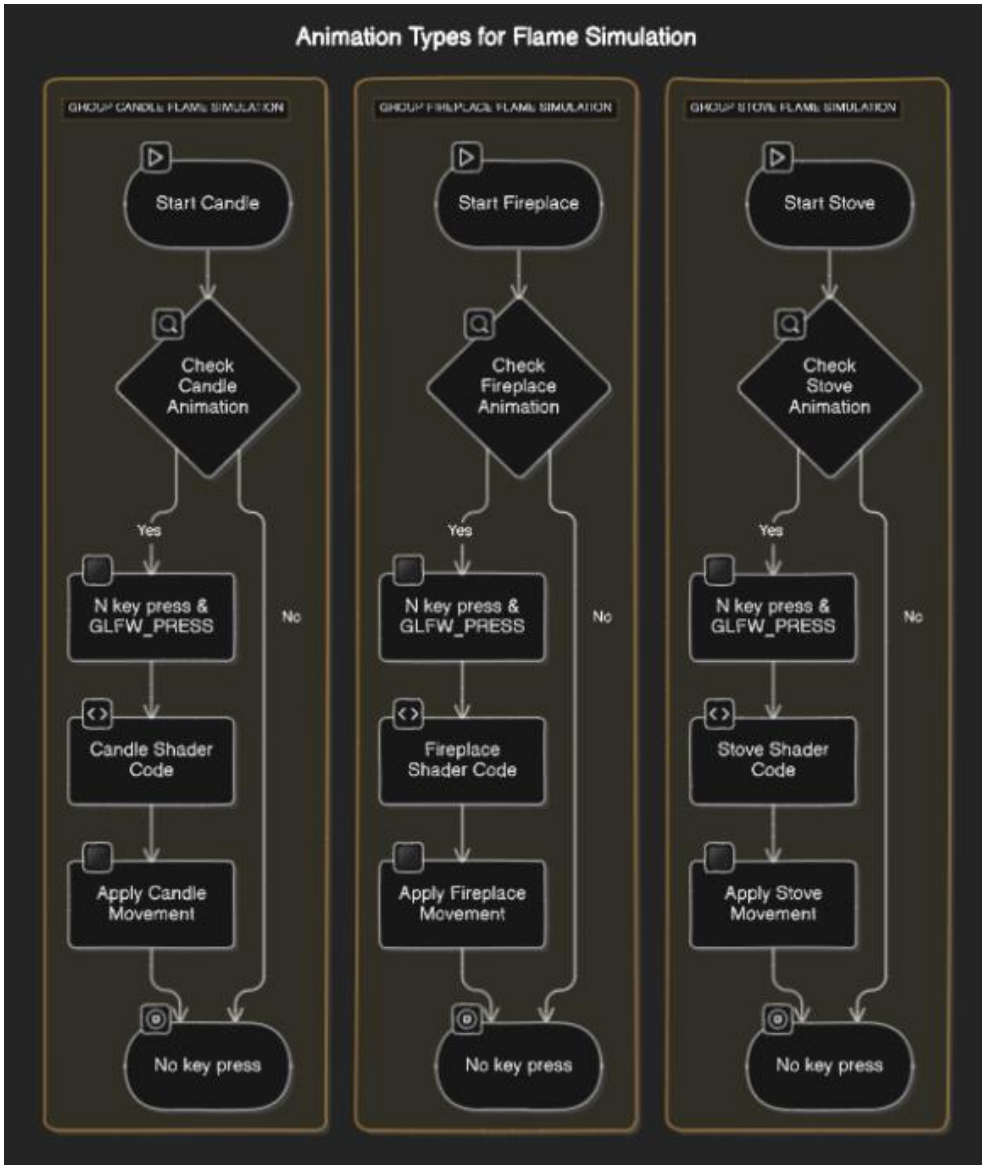
This animation is responsible for opening the drawers of the furniture next to the bed. There are a total of 2 drawers.



Complex Animation 01

This animation groups 3 types:

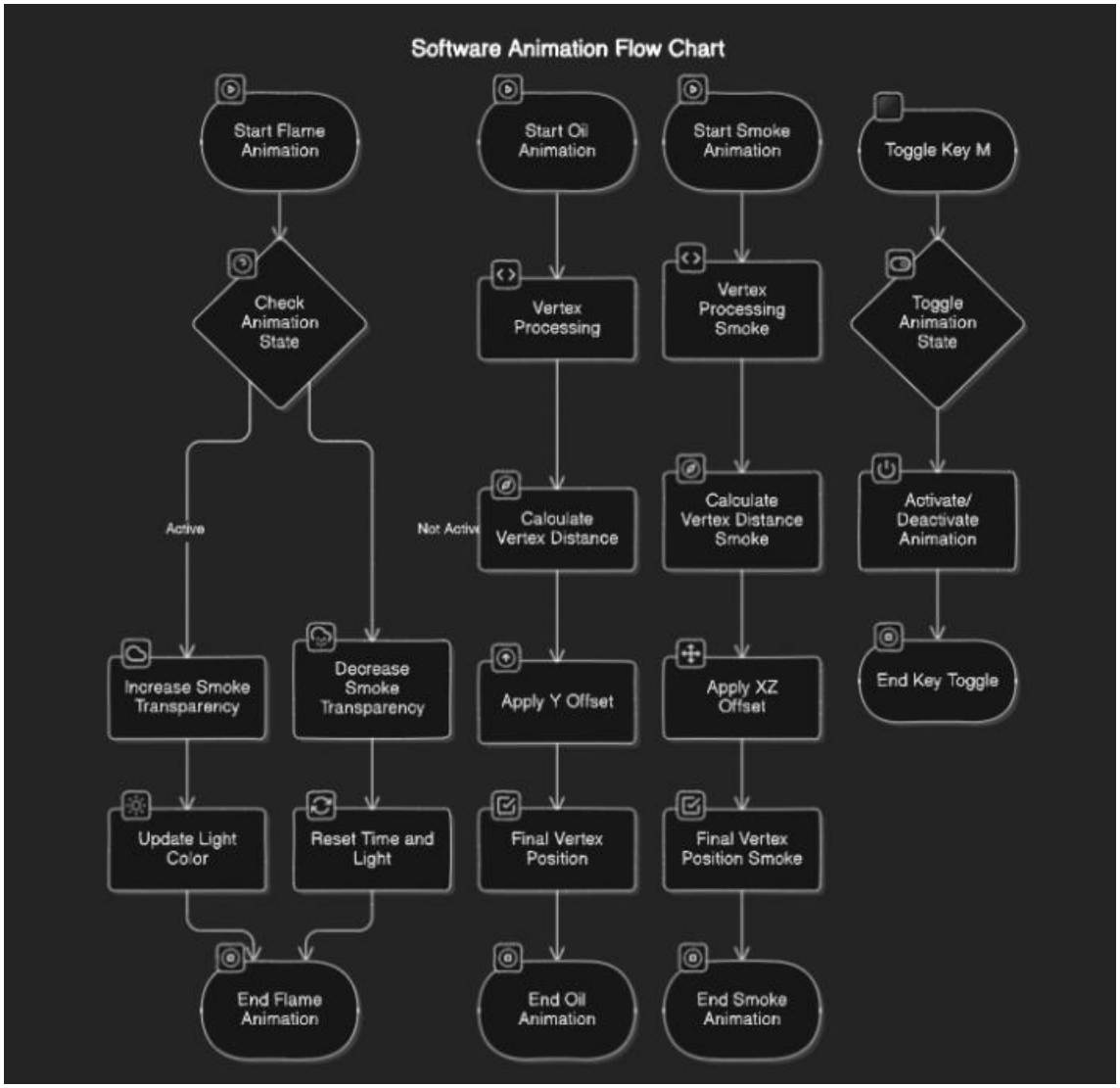
- Simulating the movement of a candle flame.
- Simulating the flames of a fireplace.
- Simulating the flames of a stove.



Complex Animation 02

This animation consists of 3 phases:

- When the stove flame ignites.
- The oil starts moving, simulating frying chicken legs.
- Smoke appears for added ambiance.



Code Documentation

Below, we will explain how the animations were carried out (considering the main code and shaders if they were used).

Libraries Used.

```
ProyectoFinal                                    (Ámbito global)
1   #include <iostream>
2   #include <cmath>
3   #include <irrKlang.h>
4   using namespace irrklang;
5   #pragma comment(lib, "irrKlang.lib")
6   // GLEW
7   #include <GL/glew.h>
8
9   // GLFW
10  #include <GLFW/glfw3.h>
11
12  // Other Libs
13  #include "stb_image.h"
14
15  // GLM Mathematics
16  #include <glm/glm.hpp>
17  #include <glm/gtc/matrix_transform.hpp>
18  #include <glm/gtc/type_ptr.hpp>
19
20  //Load Models
21  #include "SOIL2/SOIL2.h"
22
23
24  // Other includes
25  #include "Shader.h"
26  #include "Camera.h"
27  #include "Model.h"
28  #include "Texture.h"
```

Library irrklang.h

irrKlang is a high-level sound library and audio engine for C++, C#, and all .NET languages. It's known for its ease of use and its ability to play a variety of audio file formats.

The advantages of using irrklang compared to other libraries of the same type are:

- **High-level API:** irrKlang provides a simple and powerful API for sound playback in 2D and 3D applications such as games, scientific visualizations, and multimedia applications.
- **3D sound support:** It has built-in support for 3D sound on all platforms and audio drivers, designed to be efficient and not consume much CPU time.
- **3D audio emulation for low-end hardware:** irrKlang includes a high-performance 3D sound buffer emulator, providing a nearly real sound experience even on low-end hardware.

GLM Library

The GLM (OpenGL Mathematics) library is an essential tool for developers working with OpenGL. Here's a detailed description of what it does:

GLSL Data Types and Functions:

- GLM implements GLSL (OpenGL Shading Language) data types and functions in C++. This allows programmers to work with matrices, vectors, and other data structures similar to how they would in GLSL.
- For example, it defines types like glm: mat3 or glm: vec4 to handle matrices or vectors in graphics applications.

Mathematical Functions and Transformations:

- GLM provides a wide range of mathematical functions and transformation operations. These include:
- Matrix transformations: Rotation, translation, scaling, projection, etc.
- Vector operations: Dot product, cross product, normalization, etc.
- Trigonometric and exponential functions: Sine, cosine, exponential, logarithm, etc.
- Linear algebra functions: Determinant, inverse, diagonalization, etc.

SOIL2 Library

SOIL2 is a small C library primarily used for loading textures in OpenGL. It's based on stb_image, a public domain code. It loads an image file directly into an OpenGL 2D texture.

stb_image Library

stb_image is a public domain (or MIT licensed) image library for C/C++. It's a single-file library that provides functions for loading and decoding images from files or memory.

Prominent Functions

KeyCallback

This function handles detecting which key the user presses, which will be useful for correctly triggering animations.

```
// 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;
        }
    }
```

DoMovement

Once the key is pressed, this function is responsible for making the necessary modifications to display it on the screen or to ensure that the animations work correctly.

```
// 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);
    }
```

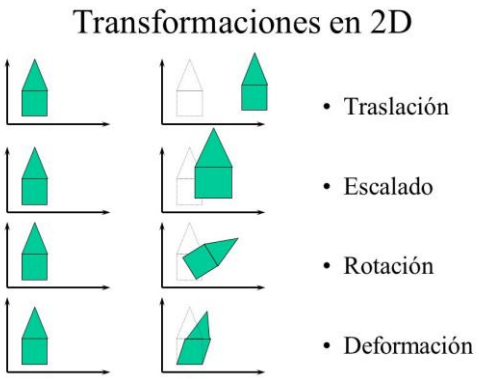
MouseCallback

This function allows us to move the view with the help of the mouse.

```
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);
}
```

Pivots and Basic Transformations.

A pivot is the center point of any basic model or primitive, as applicable. The basic transformations we make will be reflected in the pivot point.



There it is considered that the pivot is centered, but we can also move the pivots wherever we need them but leave the figure in place. These are temporary pivots which are used, for example, to animate when a door opens.

In the code, this is reflected in the model matrix and the temporary matrices use the glm library and finally it would look as shown below:

```
//=============================================================================
//Matrices Auxiliares
glm::mat4 model(1);
glm::mat4 model_Bisagra_izq(1);//Matriz Temporal para bisagras.
glm::mat4 model_Bisagra_der(1);//Matriz Temporal para bisagras.
glm::mat4 model_BisagraLibrero_der(1);//Matriz Temporal para bisagras.
glm::mat4 model_BisagraLibrero_izq(1);//Matriz Temporal para bisagras.
glm::mat4 model1_Libro01(1);//Matriz Temporal para libro 01
glm::mat4 model2_Libro01(1);//Matriz Temporal para libro 01
glm::mat4 model1_Libro02(1);//Matriz Temporal para libro 02
glm::mat4 model2_Libro02(1);//Matriz Temporal para libro 02
glm::mat4 model1_Libro03(1);//Matriz Temporal para libro 03
glm::mat4 model2_Libro03(1);//Matriz Temporal para libro 03

glm::mat4 model_Bisagra_escritorio(1);//Matriz Temporal para bisagras.
glm::mat4 model_Bisagra_caja(1);//Matriz Temporal para bisagras.
```

Simple Animation 01

This animation includes 3 animations that have the same essence, but we will describe the most complete one, which would be the opening of the desk door followed by the safe door:

We use 2 Boolean variables:

- The first one called "animacion04" to ensure the animation isn't interrupted when the same key is pressed.
- The second variable called "puerta_escritorio" which will indicate if the doors are closed or open.

There are also 2 variables (bisagra_escritorio and bisagra_caja) that will serve to show the opening of the doors. Additionally, there are 2 variables (MAXIMA_APERTURA_ESCRITORIO and MAXIMA_APERTURA_CAJA) that will act as the maximum opening value.

```
//=============================================================================================
//Variables para la animación sencilla 04 -> Puertas del escritorio
bool animacion_04 = false;
bool puerta_escritorio = false;
float bisagra_escritorio = 0.0f;
float bisagra_caja = 0.0f;
const float MAXIMA_APERTURA_ESCRITORIO = 90;
const float MAXIMA_APERTURA_CAJA = 82;
```

To activate the animation, you press the C key. To achieve this, we first go through a conditional statement to avoid interrupting the animation if it's already running. If the variable puerta_escritorio is true, when it passes through the third conditional, it will be assigned the value of false or vice versa, and animacion_04 will have the value true.

```
//=============================================================================================
// Tecla C para activar la animación del las puertas del escritorio
if (!animacion_04)
{
    if (key == GLFW_KEY_C && action == GLFW_PRESS)
    {
        if (puerta_escritorio) {
            puerta_escritorio = false;
        }
        else {
            puerta_escritorio = true;
        }
        animacion_04 = true;
    }
}
```

Here's the description of the table of cases considered for the animation:

Puerta_escritorio	Bisagra_escritorio	Bisagra_caja	Action
True	Less than MAXIMO	-------	Increase bisagra_escritorio
True	Greater than or equal to MAXIMO	Less than MAXIMO	Increase bisagra_caja
False	------	Greater than or equal to 0	Decrement bisagra_caja
False	Greater than 0	--------	Decrement bisagra_escritorio

```
//============================================================================
//Animación para realizar el movimiento del las puerta del librero y caja fuerte
if (puerta_escritorio) {
    if (bisagra_escritorio < MAXIMA_APERTURA_ESCRITORIO) {
        bisagra_escritorio += velocidad_rotacion * glfwGetTime();
    } else {
        if (bisagra_caja < MAXIMA_APERTURA_CAJA) {
            bisagra_caja += velocidad_rotacion * glfwGetTime();
        } else {
            animacion_04 = false;
        }
    }
} else {
    if (bisagra_caja >= 0) {
        bisagra_caja -= velocidad_rotacion * glfwGetTime();
    } else {
        if (bisagra_escritorio >= 0) {
            bisagra_escritorio -= velocidad_rotacion * glfwGetTime();
        } else {
            animacion_04 = false;
        }
    }
}

```

Note: We will normalize the speed of the animations using "glfwGetTime".

To be able to draw our doors, we need to use hierarchical modeling considering the following:

- Every object we want to animate must be centered in the world.
- The order of transformations affects the desired animation effect.

Considering this, we need to use 2 pivots to perform the animation. The first pivot will be the point where the rotation will occur, and the second pivot will be where our figure is located. Visualizing this in the code, we will use a temporary model matrix for the correct opening of the doors.

```
//============================================================================
//Puerta del Librero
model = glm::mat4(1);
model_Bisagra_escritorio = model = glm::translate(model, glm::vec3(26.65f, 3.70f, -13.0f));//Matriz temporal para el pivote de apertura
model = glm::rotate(model_Bisagra_escritorio, glm::radians(-bisagra_escritorio), glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::translate(model, glm::vec3(-1.16f, 0.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
objeto04_Puerta.Draw(lightingShader);

//============================================================================
//Puerta de la Caja Fuerte
model = glm::mat4(1);
model_Bisagra_caja = model = glm::translate(model, glm::vec3(24.49f, 3.54f, -11.88f));//Matriz temporal para el pivote de apertura
model = glm::rotate(model_Bisagra_caja, glm::radians(bisagra_caja), glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::translate(model, glm::vec3(0.89f, 0.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
objeto04_Puerta_Caja.Draw(lightingShader);
```

Simple Animation 02

This animation applies to 3 books, each of which is independent. Taking reality as an example, when we pick up a book, we rotate it to see the cover and at the same time tilt it to appreciate it better. Considering this, the corresponding animation is performed:

We use 2 Boolean variables:

- The first one called " animación_02_libro01" to ensure the animation isn't interrupted when the same key is pressed.
- The second variable called " libro_trasladado1" indicates whether the book is in its drawer or positioned for better reading.

We use 3 variables responsible for simulating the process:

- **Traslación_libro01:** Responsible for taking the book out of the drawer or returning it as needed.
- **Rotación_ejey:** Variable responsible for rotating the book around the Y-axis to view the cover.
- **Rotación_ejez:** Variable responsible for tilting the book around the Z-axis for greater comfort.

Variables are used to limit the desired movement:

- **ROTACION_MAXIMA_Y:** To view the book cover, a rotation limit of 90° is set.
- **INCLINACION_MAXIMA:** To improve comfort, it's unnecessary to lay the entire book flat, so a tilt of 15° is set.
- **TRASLACION_MAXIMA:** To prevent the book from moving past the desired point, this limit is defined.
- **Velocidad_rotacion:** Responsible for ensuring the book rotation appears as smooth as possible.
- **velocidad_traslacion:** Responsible for ensuring the book translation appears as smooth as possible.

```
75  //=============================================================
76  //Variables para la aimación sencilla 03 -> Libro 01
77  bool animacion_02_libro01 = false;
78  bool libro_trasladado1 = false;
79  float rotacion_ejey = 0.0f;
80  float rotacion_ejez = 0.0f;
81  const float ROTACION_MAXIMA_Y = 90;
82  const float INCLINACION_MAXIMA = 15;
83  const float velocidad_rotacion = 0.005;
84  const float velocidad_traslacion = 0.0006;
85  const float TRASLACION_MAXIMA = 2.0;
86  float traslacion_libro01 = 0.0f;
```

To activate the animation, we must press the following keys (Note: The books are independent, you can take out just 1 or all 3 at the same time):

Key 1	Book 01
Key 2	Book 02
Key 3	Book 03

First, we check if the animation is not already in progress. If it is running, we wait for it to finish.

```
//=======================================================
// Tecla 1 para activar la animación del libro
if (!animacion_02_libro01)
{
    if (key == GLFW_KEY_1 && action == GLFW_PRESS)
    {
        if (libro_trasladado1) {
            libro_trasladado1 = false;
        } else {
            libro_trasladado1 = true;
        }
        animacion_02_libro01 = true;
    }
}
```

The table of cases considered to perform the animation is described.

libro_trasladado1	traslacion_libro01	rotacion_ejey	rotacion_ejez	Action
True	Less than maximum	-------	------	Increase traslacion_libro01
True	Equal to or greater than maximum and less than maximum on the Y-axis	Less than maximum	------	Increase rotacion_ejey
True	Equal to or greater than maximum on the Y-axis and Less than maximum on the Z-axis	Equal to or greater than maximum	Less than maximum	Increase rotacion_ejez
True	Equal to or greater than maximum on the Y and Z axes	-------	------	animacion_02_libro01 a false
False	Greater than or equal to 0	Greater than or equal to 0	Greater than or equal to 0	Decrement rotacion_ejez, rotacion_ejey, and traslacion_libro01 respectively
False	Less than 0	-------	-------	animacion_02_libro01 a false

```
798     //=============================================================================
799     //Animación para realizar la el movimiento del libro 01 del librero
800     if (libro_trasladado1) {
801         if (traslacion_libro01 < TRASLACION_MAXIMA) {
802             traslacion_libro01 += velocidad_traslacion * glfwGetTime();
803         } else {
804             if (rotacion_ejey < ROTACION_MAXIMA_Y) {
805                 rotacion_ejey += velocidad_rotacion * glfwGetTime();
806             } else {
807                 if (rotacion_ejez < INCLINACION_MAXIMA) {
808                     rotacion_ejez += velocidad_rotacion * glfwGetTime();
809                 } else {
810                 animacion_02_libro01 = false;
811                 }
812             }
813         }
814     } else {
815         if (rotacion_ejez >= 0) {
816             rotacion_ejez -= velocidad_rotacion * glfwGetTime();
817         } else {
818             if (rotacion_ejey >= 0) {
819                 rotacion_ejey -= velocidad_rotacion * glfwGetTime();
820             } else {
821                 if (traslacion_libro01 >= 0) {
822                     traslacion_libro01 -= velocidad_traslacion * glfwGetTime();
823                 } else {
824                     animacion_02_libro01 = false;
825                 }
826             }
827         }
828     }
```

Just like in the previous animation, we will make use of 2 temporary matrices:

- One to position the book where we want it.
- The other matrix where the animation is applied.

```
//=============================================================================
//Libro 01 del librero
model = glm::mat4(1);
model1_Libro01 = model = glm::translate(model, glm::vec3(24.86f, 10.0f, -21.60f)); //Posición en el librero
model2_Libro01 = model = glm::translate(model1_Libro01, glm::vec3(0.0f, 0.0f, traslacion_libro01)); //Posición en el librero
model2_Libro01 = model = glm::rotate(model2_Libro01, glm::radians(-rotacion_ejey), glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::rotate(model2_Libro01, glm::radians(rotacion_ejez), glm::vec3(0.0f, 0.0f, 1.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
objeto03_Libro01.Draw(lightingShader);
```

Simple Animation 03

For this animation, a shader is used to move the star simulating that it is floating.

Two variables are used:

- The first one, called " animaciónSencilla_05", of type bool to start the animation when the same key is pressed.
- The second variable, named "proto", will be used to send the normalized clock (glfwGetTime) to the shader.

```
//================================================================
//Variables para la animación sencilla 05  -> Movimiento de la protogema
bool animacionSencilla_05 = false;
float proto = 0.0f;
```

Here, upon activation with the B key, we pass the value of the clock to the variable "proto".

```
//================================================================
//Animación Sencilla 3 -> Movimento de la protogema
if (animacionSencilla_05) {
    proto = glfwGetTime();
}
else {
    proto = 0.0f;
}
```

The shader we use to move the star is as follows:

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

const float amplitude_x = 0.05; // Amplitud del movimiento en el eje x
const float frequency_x = 7.0; // Frecuencia del movimiento en el eje x
const float amplitude_z = 0.09; // Amplitud del movimiento en el eje z
const float frequency_z = 1.5; // Frecuencia del movimiento en el eje z
const float amplitude_y = 0.09; // Amplitud del movimiento en el eje y (levantar)
const float frequency_y = 1.0; // Frecuencia del movimiento en el eje y (levantar)
const float suavisado = 110.0f;// Suavisar el movimiento del eje x
const float PI = 3.14159;

out vec2 TexCoords;

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

void main()
{
    float x_offset = amplitude_x * cos(PI * frequency_x * time/suavisado);
    float z_offset = amplitude_z * sin(PI * frequency_z * time);
    float y_offset = amplitude_y * sin(PI * frequency_y * time);

    vec3 offset = vec3(x_offset, y_offset, z_offset);

    gl_Position = projection * view * model * vec4(aPos + offset, 1.0);
    TexCoords = vec2(aTexCoords.x,aTexCoords.y);
}
```

Through a sinusoidal function, we modify the translation along the z-axis and y-axis, a cosine function for the x-axis.

```
//=============================================================================================
animacionSencilla_03.Use();
modelLoc = glGetUniformLocation(animacionSencilla_03.Program, "model");
viewLoc = glGetUniformLocation(animacionSencilla_03.Program, "view");
projLoc = glGetUniformLocation(animacionSencilla_03.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);
glUniform1f(glGetUniformLocation(animacionSencilla_03.Program, "time"), proto);
model = glm::translate(model, glm::vec3(25.45f, 2.86f, -11.30f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
objeto04_protogema.Draw(animacionSencilla_03);
```

Finally, to draw our prototype, we'll move it to the desired position.

Simple Animation 04

For this animation, it is used to perform the opening of the door of the second room. The following considerations are taken into account to achieve the movement:

Two boolean variables are used to control the animation:

- The first, called "animacion01_Cuarto02," ensures that the animation is not interrupted when the same key is pressed.
- The second variable, called "puertaCerrada_cuarto02," indicates whether the door is closed or open.

Additionally, three extra variables are used:

- **APERTURA_CUARTO:** This serves as the maximum translation value to prevent the door from moving more than necessary.
- **Riel_puerta**: This is used to perform the translation and simulate the desired movement.
- **Incremento_riel:** This is used to increment the riel_puerta value, but we need to normalize this value, so we multiply it by GLFGETTIME().

```
146 //=============================================================================
147 //Variables para la animación 01 del cuarto 2  -> Apertura de la puerta
148 bool animacion_01_cuarto02 = false;
149 bool puertaCerrada_cuarto02 = false;
150 const float APERTURA_CUARTO = 7.4;
151 float riel_puerta= 0.0f;
152 float incremento_riel = 0.001f;
153
```

To activate the animation, we need to press the key 4 to open or close the door. This is done as follows:

```
1300 //=============================================================================
1301 //Tecla 4 para activar la animación  -> Apertura de Puerta del cuarto 02
1302 if (!animacion_01_cuarto02) {
1303     if (key == GLFW_KEY_4 && action == GLFW_PRESS)
1304     {
1305         // Si la puerta está cerrada, abrir
1306         if (puertaCerrada_cuarto02) {
1307             puertaCerrada_cuarto02 = false; // La puerta se cierra
1308         }
1309         else { // Si la puertaa está abierta, cerrar
1310             puertaCerrada_cuarto02 = true; // La puerta está cerrada nuevamente
1311         }
1312         animacion_01_cuarto02 = true;
1313     }
1314 }
```

Based on the following table of cases, the door movement is considered:

Caso	Condition	State of Variables	Action	Result
1	cajonSup_cuarto02 == true y riel_CajonSup < APERTURA_CAJON	cajonSup_cuarto02 = true riel_CajonSup < APERTURA_CAJON	riel_CajonSup += incremento_cajon * glfwGetTime()	Increment `riel_CajonSup` until it reaches `APERTURA_CAJON`

2	cajonSup_cuarto02 == true y riel_CajonSup >= APERTURA_CAJON	cajonSup_cuarto02 = true riel_CajonSup >= APERTURA_CAJON	animacion_02_cuarto02 = false	Stop the animation
3	cajonSup_cuarto02 == false y riel_CajonSup > 0	cajonSup_cuarto02 = false riel_CajonSup > 0	riel_CajonSup -= incremento_cajon * glfwGetTime()	Decrement `riel_CajonSup` until it reaches 0
4	cajonSup_cuarto02 == false y riel_CajonSup <= 0	cajonSup_cuarto02 = false riel_CajonSup <= 0	animacion_02_cuarto02 = false	Stop the animation

```
1120    //=====================================================================
1121    //Animación para realizar la apertura del cajon superior del cuarto 02
1122    if (cajonSup_cuarto02) {
1123        if (riel_CajonSup < APERTURA_CAJON) { //Si la apertura es menor al máximo
1124            riel_CajonSup += incremento_cajon * glfwGetTime(); //incrementamos las bisagras
1125        }
1126        else {
1127            // La animación ha terminado, restablecer la variable
1128            animacion_02_cuarto02 = false;
1129        }
1130    }
1131    else {
1132        if (riel_CajonSup >= 0) {
1133            riel_CajonSup -= incremento_cajon * glfwGetTime();
1134        }
1135        else {
1136            // La animación ha terminado, restablecer la variable
1137            animacion_02_cuarto02 = false;
1138        }
1139    }
```

To draw our door, we use a temporary matrix to position our door in the desired location.

```
    //=============================================================================
    //Puerta de la entrada del cuarto 02
    model = glm::mat4(1);
    model_puerta_cuarto02 = model = glm::translate(model, glm::vec3(-16.46f, 22.57f, -15.65f));//Matriz tem
    model = glm::translate(model_puerta_cuarto02, glm::vec3(0.0f, 0.0f,riel_puerta));
    glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
    Puerta_Cuarto02.Draw(lightingShader);
```

Simple Animation 05

This animation is responsible for opening the drawers of the cabinet next to the bed. To achieve this animation, the following considerations are taken into account:

- cajonSup_cuarto02: A boolean variable that indicates whether the upper drawer of room 02 should open (true) or close (false).
- riel_CajonSup: A variable that represents the current position of the upper drawer's rail. This variable is incremented or decremented to simulate the drawer's opening or closing movement.
- APERTURA_CAJON: A constant that represents the maximum opening value of the drawer. When riel_CajonSup reaches this value, the drawer is fully open.
- incremento_cajon: A value that indicates how much riel_CajonSup should be incremented or decremented in each update to simulate the smooth movement of the drawer.
- glfwGetTime(): A function that returns the elapsed time since the last call to this function. It is used to normalize the increment or decrement of riel_CajonSup based on the time, ensuring smooth movement.
- animacion_02_cuarto02: A boolean variable that indicates whether the drawer's opening or closing animation is in progress (true) or has finished (false).

```
//============================================================================
//Variables para la animación 02 y 03 del cuarto 2  -> Apertura de los cajones
bool animacion_02_cuarto02 = false;
bool animacion_03_cuarto02 = false;
bool cajonSup_cuarto02 = false;
bool cajonInf_cuarto02 = false;
const float APERTURA_CAJON = 1.35;
float riel_CajonSup = 0.0f;
float riel_CajonInf = 0.0f;
float incremento_cajon = 0.0001f;
```

To activate the animation, we need to press the key 5 or 6 to open the drawers (each one is independent).

```
//=============================================================================================
//Tecla 5 para activar la animación -> Apertura del Cajon SUperior del cuarto 02
if (!animacion_02_cuarto02) {
    if (key == GLFW_KEY_5 && action == GLFW_PRESS)
    {
        // Si la puerta está cerrada, abrir
        if (cajonSup_cuarto02) {
            cajonSup_cuarto02 = false; // La puerta se cierra
        }
        else { // Si la puertaa está abierta, cerrar
            cajonSup_cuarto02 = true; // La puerta está cerrada nuevamente
        }
        animacion_02_cuarto02 = true;
    }
}
//=============================================================================================
//Tecla 6 para activar la animación -> Apertura del Cajon Inferior del cuarto 02
if (!animacion_03_cuarto02) {
    if (key == GLFW_KEY_6 && action == GLFW_PRESS)
    {
        // Si la puerta está cerrada, abrir
        if (cajonInf_cuarto02) {
            cajonInf_cuarto02 = false; // La puerta se cierra
        }
        else { // Si la puertaa está abierta, cerrar
            cajonInf_cuarto02 = true; // La puerta está cerrada nuevamente
        }
        animacion_03_cuarto02 = true;
    }
}
```

To achieve this animation, the following table of cases is considered:

Caso	Condición	Estado de Variables	Acción	Resultado
1	cajonSup_cuarto02 == true y riel_CajonSup < APERTURA_CAJON	cajonSup_cuarto02 = true riel_CajonSup < APERTURA_CAJON	riel_CajonSup += incremento_cajon * glfwGetTime()	Incrementa riel_CajonSup hasta APERTURA_CAJON
2	cajonSup_cuarto02 == true y riel_CajonSup >= APERTURA_CAJON	cajonSup_cuarto02 = true riel_CajonSup >= APERTURA_CAJON	animacion_02_cuarto02 = false	Stop the animation
3	cajonSup_cuarto02 == false y riel_CajonSup > 0	cajonSup_cuarto02 = false riel_CajonSup > 0	riel_CajonSup -= incremento_cajon * glfwGetTime()	Decrementa riel_CajonSup hasta 0
4	cajonSup_cuarto02 == false y riel_CajonSup <= 0	cajonSup_cuarto02 = false riel_CajonSup <= 0	animacion_02_cuarto02 = false	Stop the animation

```
//=============================================================================
//Animación para realizar la apertura del cajon superior del cuarto 02
if (cajonSup_cuarto02) {
    if (riel_CajonSup < APERTURA_CAJON) { //Si la apertura es menor al máximo
        riel_CajonSup += incremento_cajon * glfwGetTime(); //incrementamos las bisagras
    }
    else {
        // La animación ha terminado, restablecer la variable
        animacion_02_cuarto02 = false;
    }
}
else {
    if (riel_CajonSup >= 0) {
        riel_CajonSup -= incremento_cajon * glfwGetTime();
    }
    else {
        // La animación ha terminado, restablecer la variable
        animacion_02_cuarto02 = false;
    }
}
```

To draw our drawers, we use a temporary matrix to position our door in the desired location.

```
//=============================================================================
//Cajon superior del mueble del cuarto 02
model = glm::mat4(1);
model_puerta_cuarto02 = model = glm::translate(model, glm::vec3(30.9f, 18.61f, -16.99f));//Matriz temporal para el pivote de apertura
model = glm::translate(model_puerta_cuarto02, glm::vec3(-riel_CajonSup, 0.0f,0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
objeto04_Cajon.Draw(lightingShader);

//=============================================================================
//Cajon inferior del mueble del cuarto 02
model = glm::mat4(1);
model_puerta_cuarto02 = model = glm::translate(model, glm::vec3(30.96f, 17.11f, -16.99f));//Matriz temporal para el pivote de apertura
model = glm::translate(model_puerta_cuarto02, glm::vec3(-riel_CajonInf, 0.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
objeto04_Cajon.Draw(lightingShader);
glBindVertexArray(0);
```

Complex Animation 01

These are complex animations, ones that don't involve linear movements. As the first complex animation, we have the flame of a candle, which will be used for the chimney and stove.

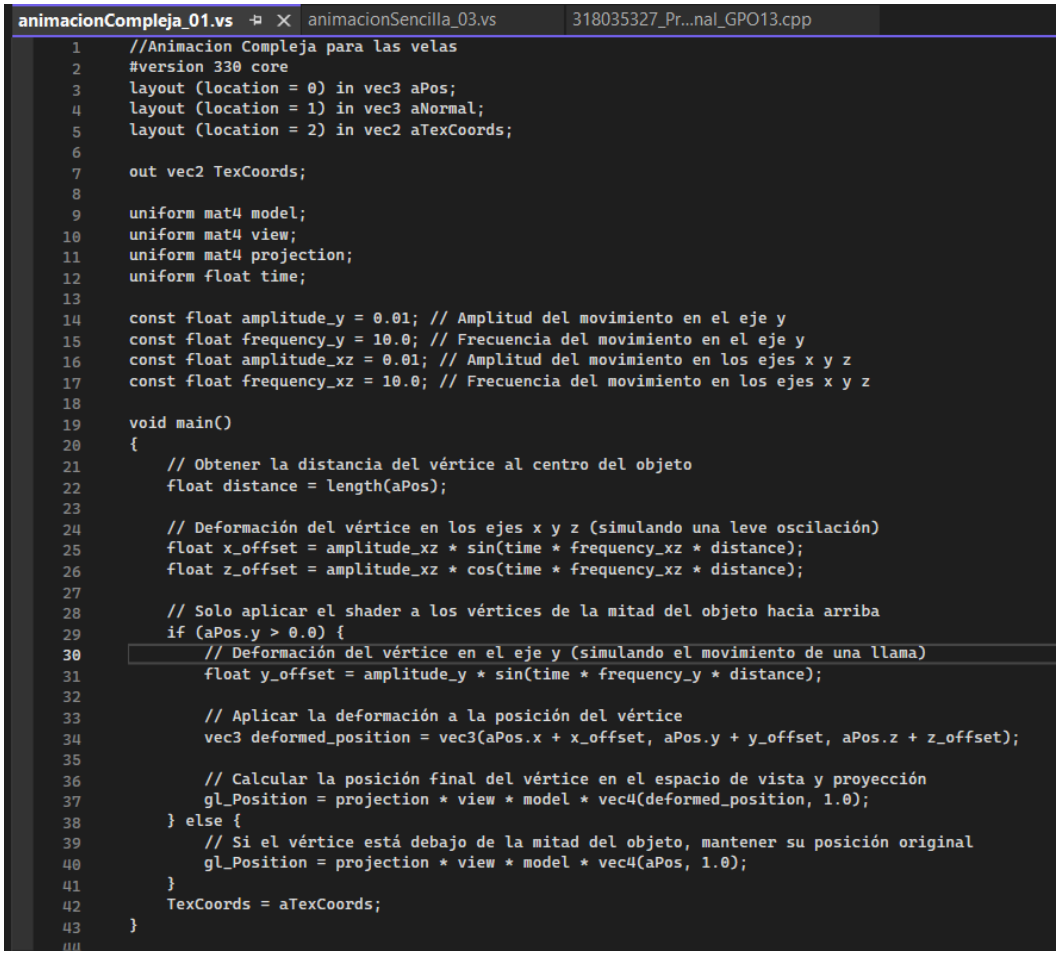
```
//=============================================================================
//Variables para la animación compleja 01 -> Movimiento de la flama
bool Compleja_01 = false;
float tiempo = 0.0f;
float transparenciaFlama = 0.0f;
const float MAXIMA_TRANSPARENCIA_VELA = 0.6f;
float velocidad_aparicion = 0.0001f;
//=============================================================================
```

A Boolean variable is used to activate the animation (sending the normalized clock to the shader to begin). To add more realism, the variable "transparenciaFlama" is used to make the flame disappear when the animation is not active and vice versa.

```
animacionCompleja_01.Use();
//tiempo = glfwGetTime();//normalizar la velocidad
// Get location objects for the matrices on the lamp shader (these could be different on a different shader)
modelLoc = glGetUniformLocation(animacionCompleja_01.Program, "model");
viewLoc = glGetUniformLocation(animacionCompleja_01.Program, "view");
projLoc = glGetUniformLocation(animacionCompleja_01.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));
glUniform1f(glGetUniformLocation(animacionCompleja_01.Program, "activaTransparencia"), 0.0);
//Vela individual
glEnable(GL_BLEND);//Avtiva la funcionalidad para trabajar el canal alfa
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
glUniform4f(glGetUniformLocation(animacionCompleja_01.Program, "colorAlpha"), 1.0f, 1.0f, 0.7f, transparenciaFlama);
model = glm::mat4(1);
glUniform1f(glGetUniformLocation(animacionCompleja_01.Program, "time"), tiempo);
model = glm::translate(model, glm::vec3(18.96f, 6.05f, -10.89f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
objeto04_flama.Draw(animacionCompleja_01);
```

When using a new shader, we must declare the view, projection, and model matrices again. We also add the ability to manipulate the transparency of the object using GL_BLEND, which helps activate the alpha channel of our texture.

Moving on to the most important part, which is the shader we use:




```
//Animacion Compleja para las velas
#version 330 core
layout (location = 0) in vec3 aPos;
layout (location = 1) in vec3 aNormal;
layout (location = 2) in vec2 aTexCoords;

out vec2 TexCoords;

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

const float amplitude_y = 0.01; // Amplitud del movimiento en el eje y
const float frequency_y = 10.0; // Frecuencia del movimiento en el eje y
const float amplitude_xz = 0.01; // Amplitud del movimiento en los ejes x y z
const float frequency_xz = 10.0; // Frecuencia del movimiento en los ejes x y z

void main()
{
    // Obtener la distancia del vértice al centro del objeto
    float distance = length(aPos);

    // Deformación del vértice en los ejes x y z (simulando una leve oscilación)
    float x_offset = amplitude_xz * sin(time * frequency_xz * distance);
    float z_offset = amplitude_xz * cos(time * frequency_xz * distance);

    // Solo aplicar el shader a los vértices de la mitad del objeto hacia arriba
    if (aPos.y > 0.0) {
        // Deformación del vértice en el eje y (simulando el movimiento de una llama)
        float y_offset = amplitude_y * sin(time * frequency_y * distance);

        // Aplicar la deformación a la posición del vértice
        vec3 deformed_position = vec3(aPos.x + x_offset, aPos.y + y_offset, aPos.z + z_offset);

        // Calcular la posición final del vértice en el espacio de vista y proyección
        gl_Position = projection * view * model * vec4(deformed_position, 1.0);
    } else {
        // Si el vértice está debajo de la mitad del objeto, mantener su posición original
        gl_Position = projection * view * model * vec4(aPos, 1.0);
    }
    TexCoords = aTexCoords;
}
```

First, we apply a deformation on the x-axis and z-axis to simulate wind moving the candle. To prevent the entire flame shape from deforming, we pass through a conditional to check that vertices above the halfway point of the object deform, while those below do not.

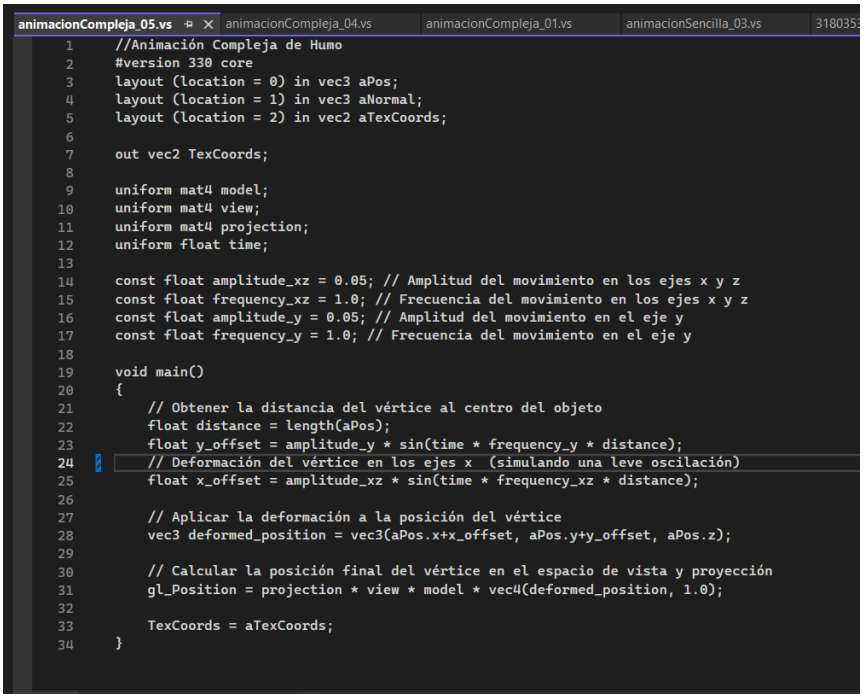
To achieve this, we need the distance between the vertices, which is obtained using the variable "distance" assigned to length(aPos).

Textures are not subjected to the same deformation to simulate color changes.

Complex Animation 02

This animation includes 3 complex animations, which I'll describe the shader for each one (above, the shader for the flame, which was used for the stove and chimney fire, was described).

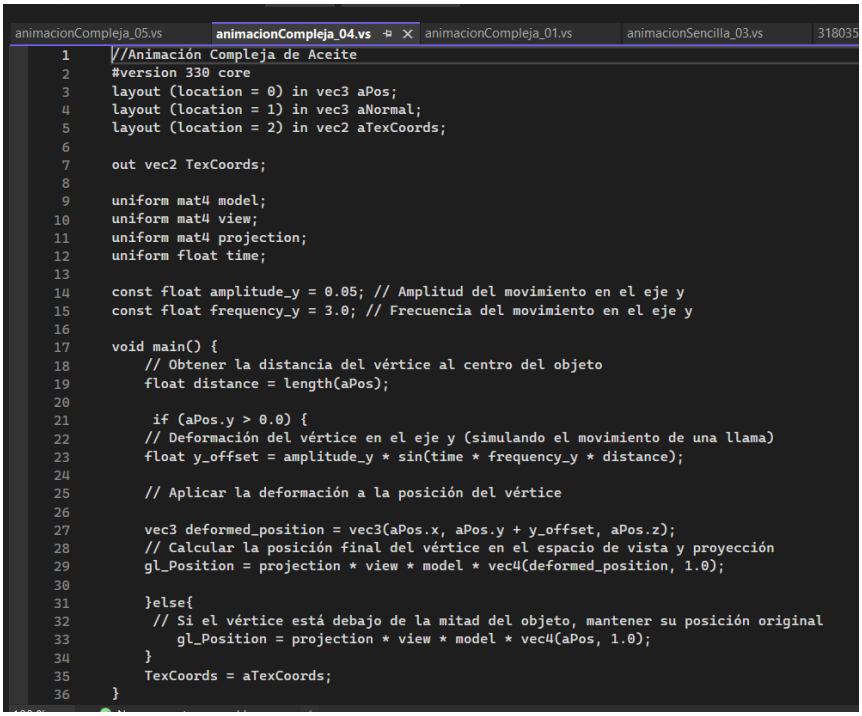
Smoke Shader:



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

out vec2 TexCoords;

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

const float amplitude_xz = 0.05; // Amplitud del movimiento en los ejes x y z
const float frequency_xz = 1.0; // Frecuencia del movimiento en los ejes x y z
const float amplitude_y = 0.05; // Amplitud del movimiento en el eje y
const float frequency_y = 1.0; // Frecuencia del movimiento en el eje y

void main()
{
    // Obtener la distancia del vértice al centro del objeto
    float distance = length(aPos);
    float y_offset = amplitude_y * sin(time * frequency_y * distance);
    // Deformación del vértice en los ejes x  (simulando una leve oscilación)
    float x_offset = amplitude_xz * sin(time * frequency_xz * distance);

    // Aplicar la deformación a la posición del vértice
    vec3 deformed_position = vec3(aPos.x+x_offset, aPos.y+y_offset, aPos.z);

    // Calcular la posición final del vértice en el espacio de vista y proyección
    gl_Position = projection * view * model * vec4(deformed_position, 1.0);

    TexCoords = aTexCoords;
}
```

In this shader, we only perform a deformation on the x-axis and the y-axis to simulate the typical movement of smoke. The texture does not undergo the same deformation.

Oil Shader:



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

out vec2 TexCoords;

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

const float amplitude_y = 0.05; // Amplitud del movimiento en el eje y
const float frequency_y = 3.0; // Frecuencia del movimiento en el eje y

void main() {
    // Obtener la distancia del vértice al centro del objeto
    float distance = length(aPos);

     if (aPos.y > 0.0) {
    // Deformación del vértice en el eje y (simulando el movimiento de una llama)
    float y_offset = amplitude_y * sin(time * frequency_y * distance);

    // Aplicar la deformación a la posición del vértice

    vec3 deformed_position = vec3(aPos.x, aPos.y + y_offset, aPos.z);
    // Calcular la posición final del vértice en el espacio de vista y proyección
    gl_Position = projection * view * model * vec4(deformed_position, 1.0);

    }else{
     // Si el vértice está debajo de la mitad del objeto, mantener su posición original
        gl_Position = projection * view * model * vec4(aPos, 1.0);
    }
    TexCoords = aTexCoords;
}
```

Just like in the fire shader, this one is only applied to half of the vertices that are above the halfway point. This is to prevent the object from deforming completely.

We move on to the most interesting part because, like the fire, we play with transparency thanks to GL_BLEND. When the M key is pressed, the animation starts with the help of a Boolean variable:

```
//=============================================================
//Tecla M para activar la animación Compleja 03
if (key == GLFW_KEY_M && action == GLFW_PRESS) {
    Compleja_03 = !Compleja_03;
}
```

Later, when that boolean variable (Complex_03) is true, we move on to the next part:

```
//=============================================================
//Animación Compleja 03 -> Flama de la Estufa, aceite y humo
if (Compleja_03) {
    tiempo3 = glfwGetTime();
    if (transparenciaHumo <= MAXIMA_TRANSPARENCIA) {
        transparenciaHumo += velocidad_humo * glfwGetTime();
    }

    if (transparenciaEstufa <= MAXIMA_TRANSPARENCIA_ESTUFA) {
        transparenciaEstufa += velocidad_aparicion * glfwGetTime();
    }
    Light2 = glm::vec3(1.0f, 0.2715f, 0.0f);
}
else {
    if (transparenciaHumo >= 0.0f) {
        transparenciaHumo -= velocidad_humo * glfwGetTime();
    }

    if (transparenciaEstufa >= 0.0f) {
        transparenciaEstufa -= velocidad_aparicion * glfwGetTime();
    }
    tiempo3 = 0.0f;
    Light2 = glm::vec3(0);//Cuado es solo un valor en los 3 vectores pueden dejar solo una componente
}
```

This animation was made considering the following table of cases:

Compleja_03	transparenciaHumo	transparenciaEstufa	Action	Light2
True	Less than or equal to maximum	Less than or equal to maximum	Increase transparenciaHumo y transparenciaEstufa	(1.0, 0.2715, 0.0)
False	Greater than or equal to 0	Greater than or equal to 0	Decrement transparenciaHumo y transparenciaEstufa	(0)

To be able to draw our objects, we need to position them where we want them, so it ends up looking like this:

```
//Aceite de la estufa
animacionCompleja_04.Use();
modelLoc = glGetUniformLocation(animacionCompleja_04.Program, "model");
viewLoc = glGetUniformLocation(animacionCompleja_04.Program, "view");
projLoc = glGetUniformLocation(animacionCompleja_04.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));
glUniform1f(glGetUniformLocation(animacionCompleja_04.Program, "activaTransparencia"), 0.0);
glUniform4f(glGetUniformLocation(animacionCompleja_04.Program, "colorAlpha"), 1.0f, 1.0f, 0.0f, 0.10f);
model = glm::mat4(1);
glUniform1f(glGetUniformLocation(animacionCompleja_04.Program, "time"), tiempo3);

model = glm::translate(model, glm::vec3(25.15f, 3.58f, -1.88f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
objeto07_aceite.Draw(animacionCompleja_04);
glBindVertexArray(0);

//============================================================================================
//Humo de la estufa
animacionCompleja_05.Use();
modelLoc = glGetUniformLocation(animacionCompleja_05.Program, "model");
viewLoc = glGetUniformLocation(animacionCompleja_05.Program, "view");
projLoc = glGetUniformLocation(animacionCompleja_05.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));
glUniform1f(glGetUniformLocation(animacionCompleja_05.Program, "activaTransparencia"), 0.0);
glUniform4f(glGetUniformLocation(animacionCompleja_05.Program, "colorAlpha"), 1.0f, 1.0f, 1.0f, transparenciaHumo);
model = glm::mat4(1);
glUniform1f(glGetUniformLocation(animacionCompleja_05.Program, "time"), tiempo3);

model = glm::translate(model, glm::vec3(25.19f, 4.72f, -2.01f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
objeto07_humo.Draw(animacionCompleja_05);
```

SkyBox

To create our skybox, we'll use a giant cube that will be generated directly from OpenGL, meaning we won't need any .obj file for this.

We define an array called skyboxVertices which contains the (x, y, z) coordinates of the vertices of the cube representing the skybox. Each set of three values represents the coordinates of a vertex:

```
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
};
```

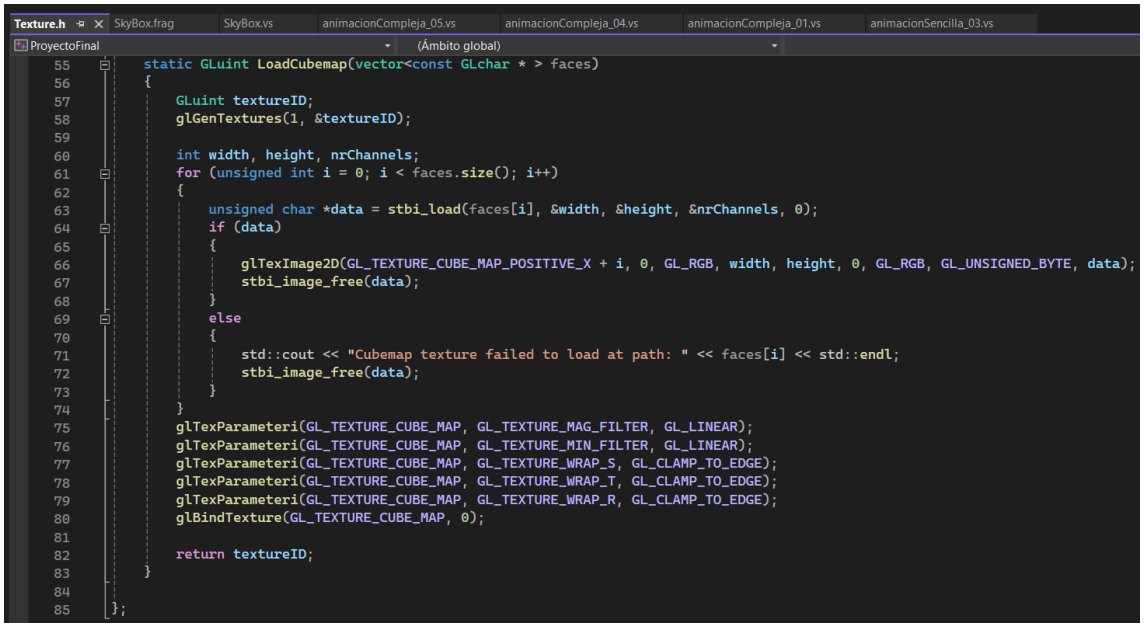
Next:

Identifiers are generated for the vertex array and the vertex buffer using the functions glGenVertexArrays and glGenBuffers. These objects are necessary for storing and managing vertex data.

A vector called "faces" is defined, containing the paths of the textures that will be used for the skybox on each of the six faces (right, left, up, down, back, front). Then, the cubemap (cube mapping texture) is loaded using the function TextureLoading: LoadCubemap, which loads and configures the textures according to the paths specified in the "faces" vector.

```
//SkyBox
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);

// Load textures del skybox
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);
```

To ensure the skybox works correctly, the cube faces must be modified with normals pointing inwards, and a parallax effect must be generated within this cube. This effect is incorporated into the use of the texture. Textures in .tga format are used here. This modification is done in the Texture.h file.




```
static GLuint LoadCubemap(vector<const GLchar * > faces)
{
    GLuint textureID;
    glGenTextures(1, &textureID);

    int width, height, nrChannels;
    for (unsigned int i = 0; i < faces.size(); i++)
    {
        unsigned char *data = stbi_load(faces[i], &width, &height, &nrChannels, 0);
        if (data)
        {
            glTexImage2D(GL_TEXTURE_CUBE_MAP_POSITIVE_X + i, 0, GL_RGB, width, height, 0, GL_RGB, GL_UNSIGNED_BYTE, data);
            stbi_image_free(data);
        }
        else
        {
            std::cout << "Cubemap texture failed to load at path: " << faces[i] << std::endl;
            stbi_image_free(data);
        }
    }
    glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
    glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
    glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_WRAP_R, GL_CLAMP_TO_EDGE);
    glBindTexture(GL_TEXTURE_CUBE_MAP, 0);

    return textureID;
}

};
```

Finally, to draw our cube, we use glDrawArrays.

```
//==========================================================================================
// 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
```

Audio with IrrKlang

To integrate audio into the project, there are several libraries available, but for ease of use, the IrrKlang library was used.

To use it correctly, the following steps are taken:

```
3    #include <irrKlang.h>
4    using namespace irrklang;
5    #pragma comment(lib, "irrKlang.lib")
6    // GLEW
```

Where:

- **using namespace irrklang:** This line tells the compiler that the irrklang namespace will be used in this code file. This means that it's not necessary to specify `irrklang: ` before each use of a class or function from irrKlang.
- **#pragma comment (lib, "irrKlang.lib"):** This line is specific to Windows-based development environments. It instructs the linker compiler to include the irrKlang linking library `irrKlang.lib` when compiling the code. This library contains the irrKlang implementation that the program needs to function correctly at runtime.

Inside our Main:

```
146  int main()
147  {
148      // start the sound engine with default parameters
149      ISoundEngine* engine = createIrrKlangDevice();
150      if (!engine)
151          return 0; // error starting up the engine
152
153      // play some sound stream, looped
154      engine->play2D("Music/musicaAmbiental.mp3", true);
155
156
```

- **ISoundEngine* engine = createIrrKlangDevice ():** This line creates an instance of the irrKlang sound engine using the `createIrrKlangDevice () ` function. This function initializes the sound engine with default parameters and returns a pointer to an ISoundEngine object, which is the main interface for interacting with the irrKlang sound engine.
- **if (! engine) return 0:** After creating the sound engine, this line checks if the initialization was successful. If the engine pointer is null, it means an error occurred during sound engine initialization, so the application exits with an error code (0 in this case).
- **engine->play2D ("Music/musicaAmbiental.mp3", true):** Once the sound engine has been initialized successfully, this line plays an audio file in MP3 format.

Cost Analysis

The following section aims to analyze the costs anticipated by the author for the sale of this project. The following points are addressed:

- As the original author of the work, the number of hours spent on the project is reflected in the Gantt chart.
- The daily cost, based on the minimum wage, is: $248.93 MXN.
- Cost for support included in the project: $450.00 MXN.

The following table provides a breakdown of the total expenses incurred by the author for its creation.

Author	Hours Spent	Sueldo x Salary	Working Days	Total
Jimenez Cervantes Angel Mauricio	435	248.93	18.12	$ 4 510.61 MXN

Operating Costs

During the project, specialized computing was required, for which the equipment rental is $4200 MXN per month from a company dedicated to these types of situations.

The use of electricity for one person is also considered, amounting to approximately $150 MXN per month, and the use of the internet, approximately $300 MXN per month.

With this, the total sum of the anticipated costs in the previous table plus the operational costs would be:

Total = $4 510.61 + $4 650.00 = $9,161.00 MXN Rounding.

If the purchase of the specialized computing equipment is made, the price is $9799.00 MXN, so the total cost would be:

Total = $4 510.61 + $10249.00 = $14,760.00 MXN Rounding.

Conclusions

This project demonstrates a solid application of the knowledge acquired during the course, highlighting the ability to recreate three-dimensional environments using OpenGL. Specific objectives include the selection and recreation of a facade and a space, as well as capturing the essence of the atmosphere from reference images. The bilingual delivery and the use of an official repository on GitHub meet the established requirements.

Individual limitations and evaluation criteria emphasize the importance of project quality and originality, as well as the need to meet established standards.

Creative constraints, such as the prohibition of certain themes and the requirement for graphic quality, ensure the integrity of the project and foster creativity within established boundaries.

The project makes effective use of various tools and resources, from modeling software like Maya and Paint 3D to libraries like irrKlang for sound management. The software methodology, including the use of Kanban for project management, contributes to an efficient and organized execution of the project.

The detailed documentation of the code, including explanations of the animations and libraries used, provides a clear understanding of the processes and decisions made during the project's development. In summary, this project exhibits a methodical, creative, and technically competent approach to recreating three-dimensional environments using OpenGL.

Attachments

For more detail on the libraries used, you can access the following links:

- Documentation of OpenGL version 3.30: GLSLangSpec.3.30.pdf (khronos.org)
- Documentation of GLM: 0.9.8: OpenGL Mathematics (g-truc.net)
- Documentation of SOIL2: SpartanJ/SOIL2: SOIL2 is a tiny C library used primarily for uploading textures into OpenGL. (github.com)
- Documentation of irrKlang: irrKlang: irrKlang 1.6.0 API documentation (ambiera.com)

- Documentation of stb_image: nothings/stb: stb single-file public domain libraries for C/C++ (github.com)
- Documentation of assimp: The Asset-Importer-Lib Documentation — Asset-Importer-Lib March 2022 v5.2.3 documentation (assimp-docs.readthedocs.io)

To access the code or the models, you can consult the following repositories.

Implementation Repository	Models Repository