Technical Manual

1.- Introduction

This document provides a detailed overview of the methodology, architecture, variable and function dictionary, mathematical fundamentals of complex animations, Gantt chart, limitations, objectives, and conclusions of our project. In it, we modeled the Up House from the movie "Up" using OpenGL and Maya as the engine for the models.

2.- Objectives

1. Develop a detailed technical and user manual that includes the applied software methodology, providing a comprehensive guide for interaction within the recreated virtual environment.
2. Maintain an updated and functional project repository, clearly showing changes and progress made during the semester, without using compressed files.
3. Achieve a high degree of realism in the recreated virtual space, compared to the provided reference photo.
4. Develop an executable file that can be opened and run correctly, ensuring that the source code is not included in the final delivery.
5. Accurately model a minimum of 7 elements specified in the reference document, ensuring correct texturing and adequate geometry for furniture and buildings.
6. Create 5 animations, including 3 simple and 2 complex ones, consistent with the context of the recreated space.
7. Set the virtual space atmosphere with proper lighting, contributing to the project's overall atmosphere.
8. Implement proper camera handling to allow a smooth and natural user experience within the virtual environment.

3.- Requirements:

3.1 Functional Requirements:

1. **3D Recreation:** The system must allow the 3D recreation of selected facade and space, based on reference images provided by the user.
2. **Objects to Recreate:** 7 specific objects must be recreated, as close as possible to their reference image, as well as their environment.
3. **Manuals:** A user manual must be provided that explains each interaction within the recreated virtual environment, as well as a technical manual containing project documentation, objectives, software flowchart, Gantt chart, project scope, limitations, software methodology, code documentation, and conclusions.
4. **Delivery and Platform:** The project must be delivered individually in a GitHub repository and uploaded to the classroom platform before the specified deadline.
5. **Camera Handling:** Controls must be implemented for camera handling within the recreated virtual space.
6. **Executable:** An executable file must be provided that exists, opens, works, and does not contain the source code.
7. **Atmosphere and Realism:** Proper lighting use within the space and achieving realism of the virtual space against the reference photo must be ensured.
8. **Modeling and Texturing:** A minimum of 7 elements specified in the reference images must be recreated with correct texturing and suitable geometry.

3.2 Non-functional Requirements:

1. **Documentation Language:** The project documentation must be delivered in both Spanish and English, without using Google Translate.
2. **Use of Base Code:** The base code provided during the course must be used; otherwise, the project will be canceled, and the student will not pass the lab.

3. **Theme Restrictions:** It is prohibited to recreate spaces belonging to certain specific themes, such as universities, animated series, video games with low graphical characteristics, among others.
4. **Animation Complexity:** Animations must have context and cannot be linear; 5 animations must be created, where 3 are simple and 2 are complex.
5. **Changes and Progress Visualization:** The repository must show a visualization of changes and progress during the semester, without compressed files and functional.

4.- Methodology

The project "The Up House: An Adventure in Altitude" was developed following the Agile Scrum methodology. Scrum is a framework that allows us to efficiently manage the development of this complex and creative project, ensuring the delivery of a high-quality product within controlled times and costs.

4.1 Development Teams

- The development team consists of one person, who carried out the actions of modeling, programming, testing, and animation.

4.2 Product Backlog

- At the beginning of the project, a product backlog was created containing all the features and functionalities that were desired to be included in "The Up House: An Adventure in Altitude."
- The backlog is prioritized based on user value and technical complexity.

House Modeling

- **Up House Modeling:** Includes all exterior and interior details.

Objects Modeling

- **Modeling of the following interior objects of the house:** sofa, coffee table, lamp, Carl's armchair, Ellie's chair, rug, and cushions.

Simple Animations

- Animation of the lamp turning on and off.
- Animation of the main door opening and closing.
- Animation of the drawer opening and closing.

Complex Animations

- Animation of the balloons floating at the top of the facade.
- Animation of the balloon floating in the vase inside the house.

House Facade

- Modeling and texturing of the Up House facade, including worn paint and roof tiles.

User Interaction

- Implementation of user interaction to allow opening and closing the door and window, turning the lamp on and off, and opening and closing the furniture drawer.

Optimization and Testing

- Scene optimization through cleaning, low topology, and texturing to avoid high loading times.

Sprints

- Work was divided into fixed duration sprints of one week.
- At the beginning of each sprint, planning was carried out in which the tasks to be performed in that sprint were selected from the product backlog and supported by the Gantt chart.

Review and Retrospective Meetings

- At the end of each sprint, a review was conducted to demonstrate the work done.

Incremental Delivery

- Following the agile approach, incremental deliveries of functionalities were made, allowing for early feedback and adjusting development according to project needs.

5.- System Architecture

The program is designed to provide an interactive experience to users. It consists of several key components that work together to create a detailed and realistic virtual environment. The main components of the system architecture are described below:

1. **OpenGL:** Used as a graphics API to render real-time graphics. OpenGL provides the necessary tools to create advanced visual effects and render 3D models efficiently.
2. **Maya:** Used as modeling and animation software to create the 3D models of the Up House and other objects within the environment. Maya integrates with OpenGL to export models and animations to the format compatible with the project.
3. **User Interaction:** An intuitive user interface is implemented that allows users to interact with the virtual environment. This interface is based on user input events, such as mouse movement and key presses, to control the camera and trigger animations and special effects.
4. **Asset Management:** An asset management system like SOIL 2 is used to load and manage 3D models, textures, animations, and other resources used in the project. This ensures that resources are loaded efficiently and kept in memory during program use.
5. **Optimization and Performance:** Optimization techniques have been applied to ensure smooth performance on a wide range of devices. This includes the use of efficient rendering techniques such as triangulation, texture mapping, low-cost directional lighting, and low topology of elements.

6.- Variable and Function Dictionary

Global Variables:

- **rotp:** Controls the rotation of an object.
- activaanim1, activaanim2, activaanim3: Boolean variables that control the activation of different animations.
- **tras1:** Controls the translation of an object.
- **WIDTH, HEIGHT:** Window dimensions.
- **SCREEN_WIDTH, SCREEN_HEIGHT:** Framebuffer dimensions.
- **camera:** Object of the Camera class that controls the camera in the scene.
- **lastX, lastY:** Last cursor coordinates.
- **keys:** Array of keys to control movement.
- **firstMouse:** Boolean indicating if it's the first time the cursor is moved.
- **lightPos:** Light position in the scene.
- **active, active2, active3:** Boolean variables to control light activation.
- time: Variable to store the current application time.
- **pointLightPositions, ubfoc, dirlamp:** Positions of different lights in the scene.
- **vertices:** Array containing the vertices of a cube.

Functions:

- **main():** Main function that initializes GLFW, GLEW, creates the window and OpenGL context, loads shaders and models, and contains the main loop of the application.
- **KeyCallback():** Callback function that handles keyboard events.
- **MouseCallback():** Callback function that handles mouse events.
- **DoMovement():** Function that processes camera movement according to the pressed keys.

7.- Mathematical Fundamentals of Complex Animations

Animation 1 - Shader Anim

The expression sin(time * speed) * 0.1 in the shader represents the calculation of vertical displacement based on a sinusoidal function.

- *time* is a variable that provides the current system time in seconds.
- *speed* is a value that determines the speed at which oscillation occurs. The higher the speed, the faster the displacement changes.
- *time * speed* multiplies time by speed, meaning the argument of the sine function changes with time and speed.
- *sin(time * speed)* calculates the sine of time * speed, resulting in an oscillating value between -1 and 1 over time.
- *sin(time * speed) * 0.1* multiplies the result by 0.1, reducing the amplitude of the oscillation to a smaller range, resulting in a more subtle displacement.

Animation 2 - Shader 2

The function asin(time/100)*PI calculates the angle whose sine is time/100, scaling time to adjust the input range of the asin function. Then, this angle is multiplied by PI to convert it to radians. This calculation is used to generate a periodic wave effect over time.

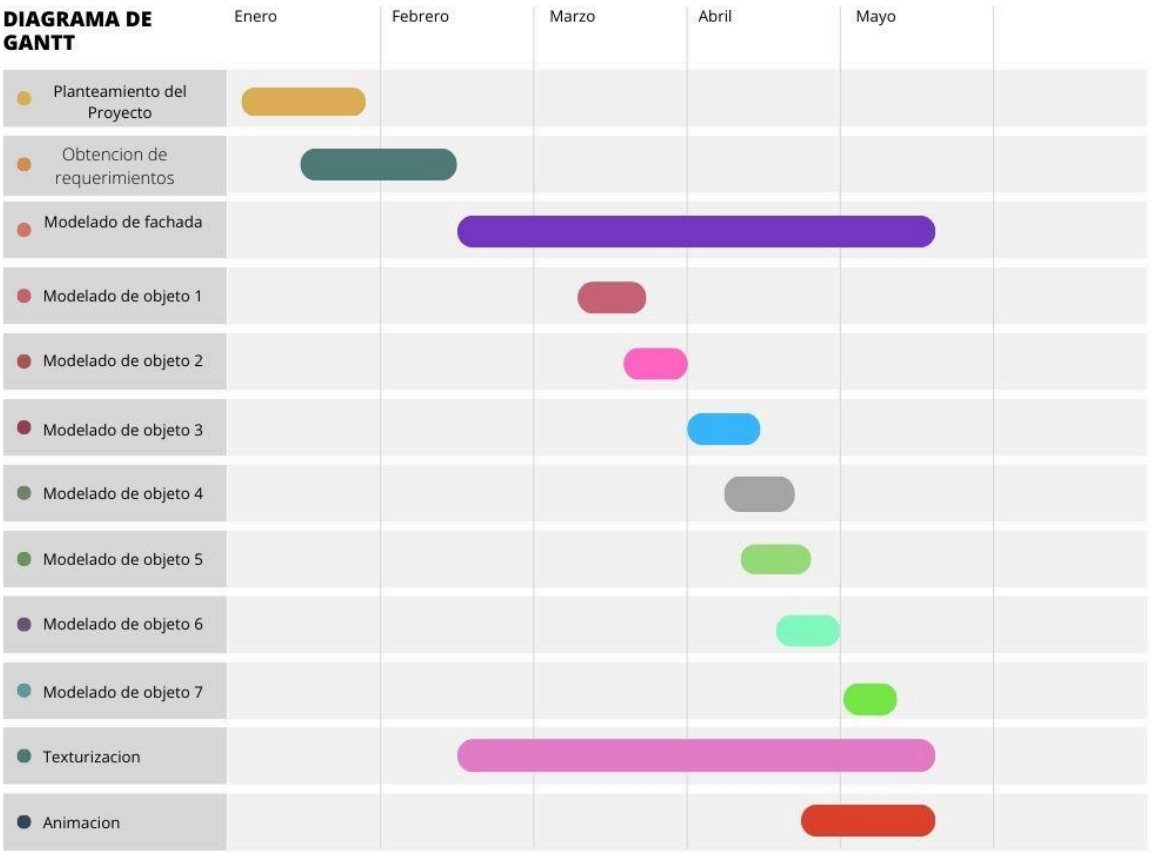
8.- Update and Versioning

For project versioning, GitHub was used along with Visual Studio. GitHub is a code hosting platform that uses Git for version control. Git is a distributed version control system that allows managing code changes efficiently, maintaining a complete history of all modifications made.

- Visual Studio, on the other hand, is an integrated development environment (IDE) that provides tools for writing, debugging, and compiling code. Visual Studio has Git integration, making version control management directly from the IDE easier.
- By using GitHub with Visual Studio, a detailed record of changes made to the code can be maintained, allowing effective collaboration with other developers

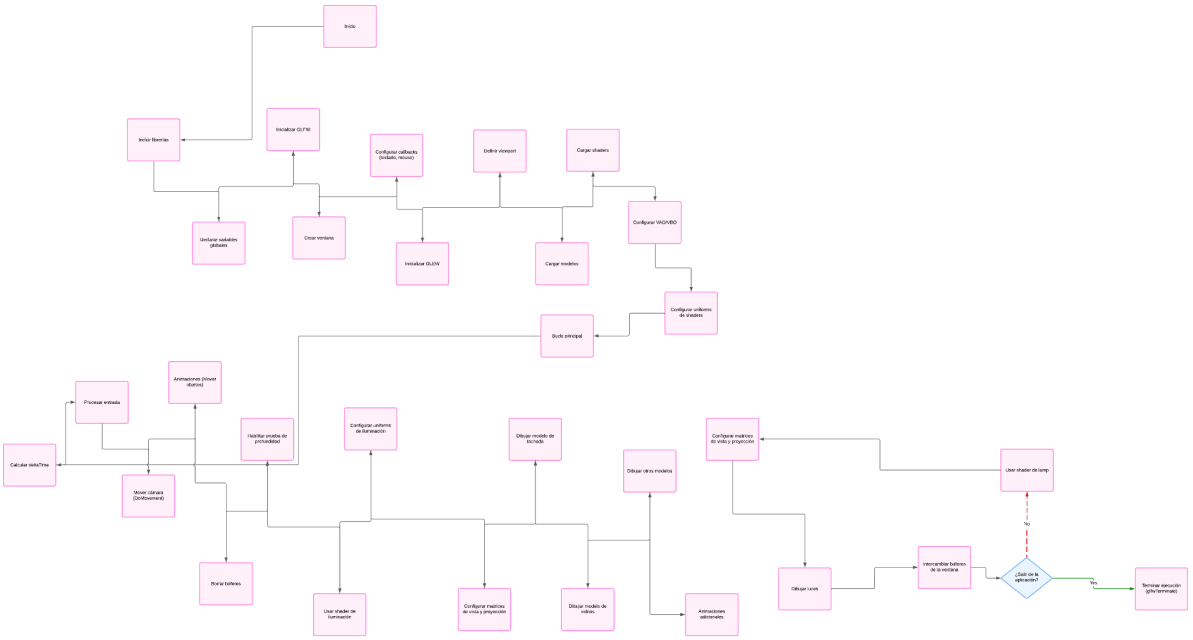
and maintaining a version history that facilitates the identification and correction of errors.

9.-Gantt Chart

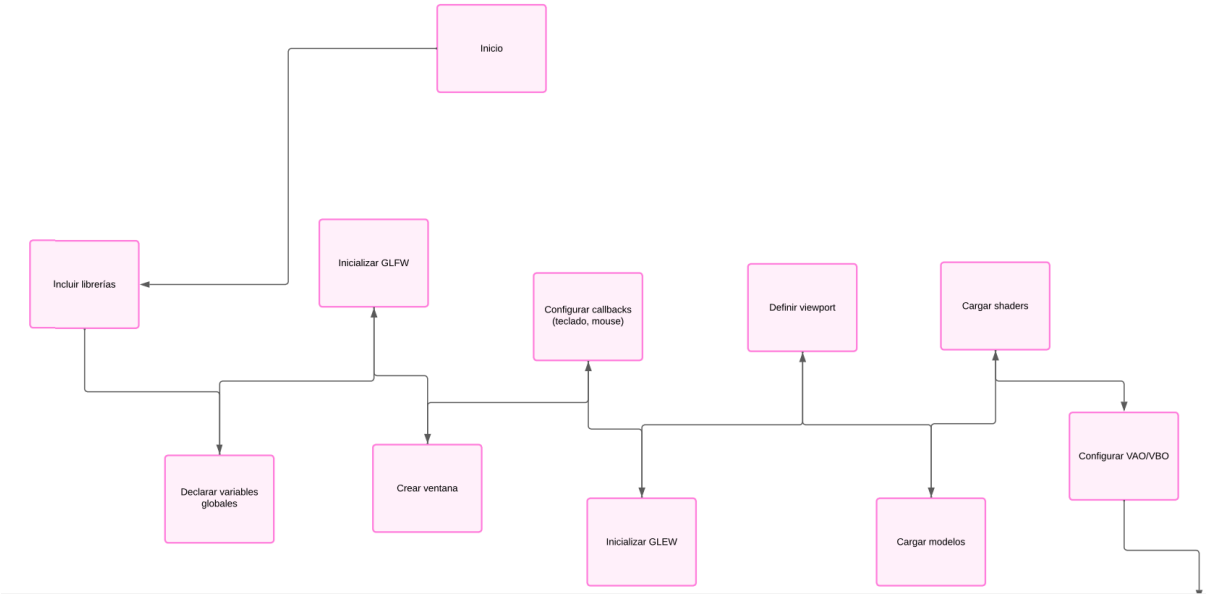


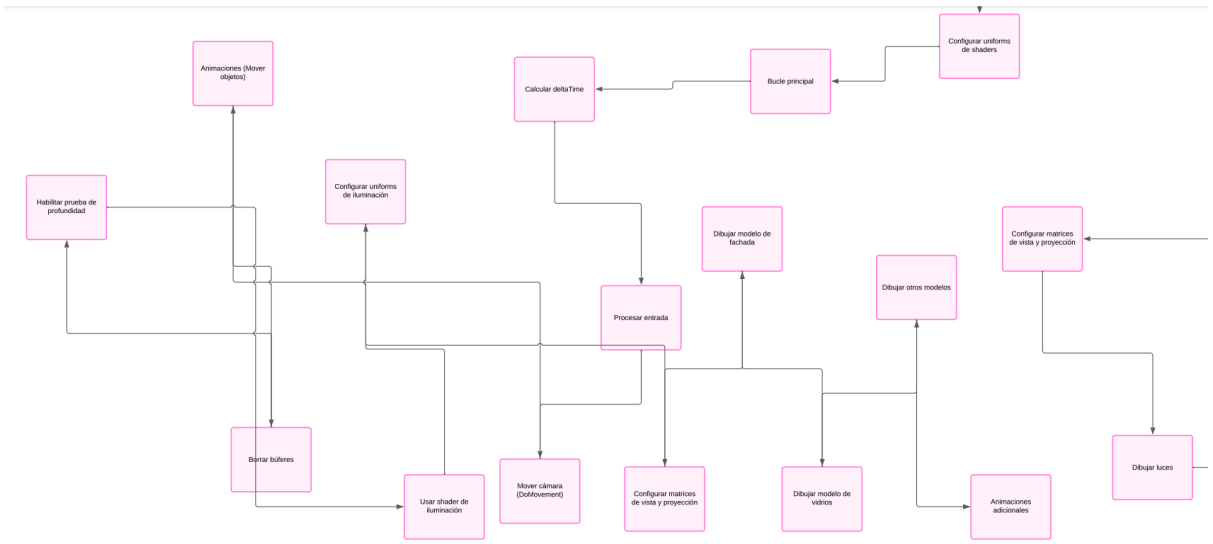
10. Flowchart

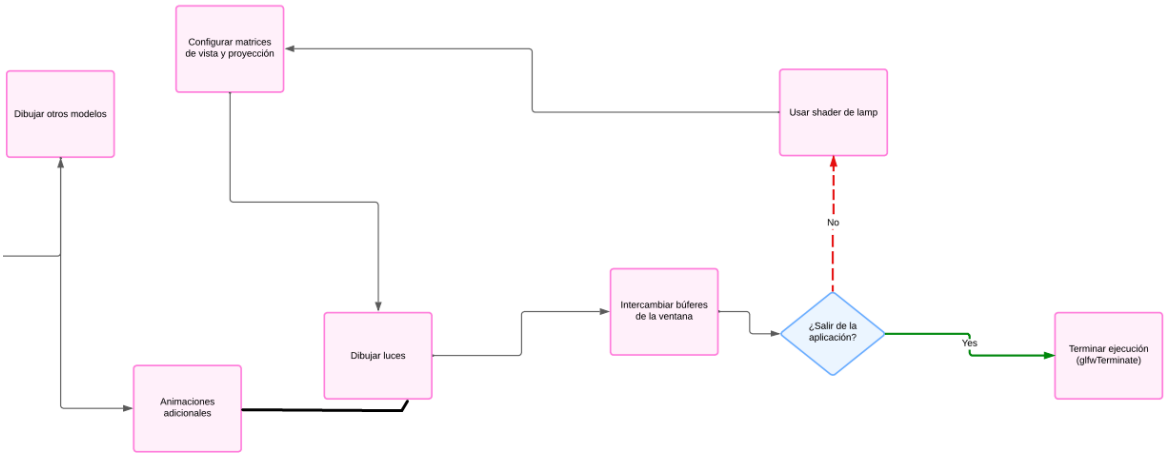
Complete Flowchart



Flowchart by Parts







11.- Explanation of the code

Shader loading: In this section, shaders are loaded, including those for animation (`anim` for complex animation 1, `anim2` for complex animation 2), `lightning`, and `lampshader` for lighting.

```
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");
```

Model loading:

Here the models are loaded, specifying the path and the name of the object.

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

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


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


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


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

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

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

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

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

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

Model loading:

Here the model loading process is described. First, the matrix is configured, then characteristics such as translations, transparencies, and rotations are applied.

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


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


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

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


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

Shader loading for animations:

Here you can use shaders, employing two different ones where complex animations are loaded.

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

    // Set matrices
    glUniformMatrix4fv(viewLoc, 1, GL_FALSE, glm::value_ptr(view));
    glUniformMatrix4fv(projLoc, 1, GL_FALSE, glm::value_ptr(projection));
    glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));

    model = glm::mat4(1);

    glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
    glUniform1f(glGetUniformLocation(Anim2.Program, "time"), tiempo);
    globos.Draw(Anim2);
    glBindVertexArray(0);


}

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

// Set matrices
glUniformMatrix4fv(viewLoc, 1, GL_FALSE, glm::value_ptr(view));
glUniformMatrix4fv(projLoc, 1, GL_FALSE, glm::value_ptr(projection));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));

model = glm::mat4(1);

glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1f(glGetUniformLocation(Anim2.Program, "time"), tiempo);
globo.Draw(Anim);
glBindVertexArray(0);
```

Camera movement:

In this section, the keys and the DoMovement function are used for movement within the space.

```
void DoMovement()
{

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

    }

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


    }

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


    }

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


    }
```

12.- Limitations

- Processing Capacity: Handling larger object topologies requires higher processing capacity, which can limit the speed and fluidity of the application on less powerful hardware.
- UV Map Generation: Lack of experience in UV map generation can limit the quality and variety of textures applicable to objects, affecting the visual appearance of the scene.
- 3D Modeling Knowledge: Lack of knowledge in 3D modeling can limit the diversity and complexity of objects that can be created, restricting creativity and variety in the scene.
- Animation Complexity: Animation complexity may be limited by processing capacity and desired frame rate, which can affect the smoothness and realism of animations.
- Limited Interactivity: Application interactivity may be limited by processing capacity and the complexity of required calculations, which can affect the user experience.

12.- Conclusions

This project has served to demonstrate the skills acquired during the computer graphics course. It has been a challenging project due to the use of animations, basic transformations, 3D modeling, complex animations, texturing, and understanding of 3D space. The objectives acquired during the course and the specific objectives of the project have been achieved.