National Autonomous University of Mexico

Faculty of Engineering

Computer Graphics and Human-Computer Interaction

Group: 05 - Semester: 2023-1

Documentation:

Technical manual

Delivery date: 10/01/2022

Professor:

Carlos Aldair Román Balbuena.

Alumn:

Cruz Rangel Leonardo Said

Index

1. Synthetic camera

To use the synthetic camera, it is necessary to import the camera.h header into the main program, provided by the laboratory professor during the practices.

```
#include <camera.h>
```

To use the functions in that header, you must create an object that belongs to that class, so the code looks like this:

```
// camera
Camera camera(glm::vec3(0.0f, 200.0f, 700.0f));
```

Inside the header, we can find default constants that define the speed (SPEED) at which the camera moves, its sensitivity, the zoom it applies, etc. These values can be modified according to the user's experience and comfort moving through the scene.

```
// Default camera values
const float YAW        = -90.0f;
const float PITCH      =  0.0f;
const float SPEED      =  0.5f;
const float SENSITIVITY =  0.7f;
const float ZOOM       =  45.0f;
```

To move the camera with the keyboard keys, we declared the function **my_input**. This function allows detecting when someone presses a key in the program. It is a listening action provided by GLFW, and then we check which key has been pressed with an if control structure, as shown below:

```
void my_input(GLFWwindow *window, int key, int scancode, int action, int mode)
{
        if (glfwGetKey(window, GLFW_KEY_ESCAPE) == GLFW_PRESS)
                glfwSetWindowShouldClose(window, true);
        if (glfwGetKey(window, GLFW_KEY_W) == GLFW_PRESS)
                camera.ProcessKeyboard(FORWARD, (float)deltaTime);
        if (glfwGetKey(window, GLFW_KEY_S) == GLFW_PRESS)
                camera.ProcessKeyboard(BACKWARD, (float)deltaTime);
```

```
if (glfwGetKey(window, GLFW_KEY_A) == GLFW_PRESS)
        camera.ProcessKeyboard(LEFT, (float)deltaTime);
if (glfwGetKey(window, GLFW_KEY_D) == GLFW_PRESS)
        camera.ProcessKeyboard(RIGHT, (float)deltaTime);
```

For the camera case, the use of **FORWARD, BACKWARD, LEFT** and **RIGHT** is made to specify the movement to which the camera will be subjected when a key is pressed.

2. Static objects

It is essential to import models in .OBJ format in order to use static objects in the project, as the program is designed to work almost exclusively with this file type, as it has the highest compatibility and generates fewer errors during import.

To organize and keep track of the objects that are added to the project, the 'objects' folder has been created within the 'resources' folder, whose full path is as follows: ProyectoFinal/ProyectoFinal/resources/objects. This folder, in turn, contains subfolders that store both the textures and the corresponding .OBJ and .MTL files for each of the elements that are part of the final scene of the project. It is important to make sure that the .MTL file correctly references the textures, as otherwise they will not be loaded into the program.

Below is the process for loading a static object into the scene:

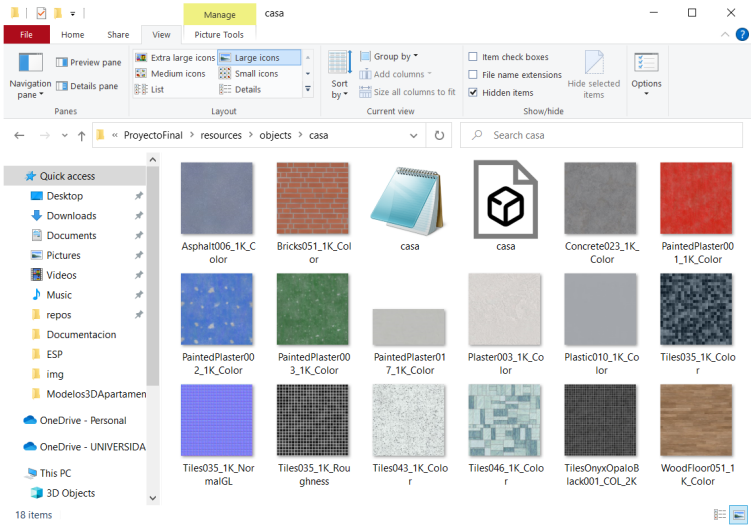

Figura 1. Files corresponding to the house model.

Once the files have been properly stored in the 'objects' folder, the next step is to declare that object. In this case, as it is a static object, the Static Shader and the Model class are used.

Below is the declaration of some of the static objects in the project:

```
// CARGA DE MODELOS
// ----------
// MODELOS ESTÁTICOS
// ----------
//ARBUSTOS
Model arbustos("resources/objects/arbustos/bush.obj");

//CASA
Model casa("resources/objects/casa/casa.obj");

//COCINA
Model licuadora("resources/objects/cocina/blender.obj");
Model comedor("resources/objects/cocina/comedor.obj");
Model estufa("resources/objects/cocina/estufa.obj");
Model mantel("resources/objects/cocina/mantel.obj");
Model microwave("resources/objects/cocina/microwave.obj");
Model mueble("resources/objects/cocina/mueble.obj");
Model platos("resources/objects/cocina/platos.obj");
Model refrigerador("resources/objects/cocina/refrigerador.obj");
Model rollo("resources/objects/cocina/rollo.obj");

//CUARTO1
//cama1
Model almohadas1("resources/objects/cuarto1/cama/almohadas.obj");
Model base1("resources/objects/cuarto1/cama/base.obj");
Model cabecera1("resources/objects/cuarto1/cama/cabecera.obj");
Model colchon1("resources/objects/cuarto1/cama/colchon.obj");
Model sabana1("resources/objects/cuarto1/cama/sabana.obj");
```

Finally, all that is left is to instantiate the objects, apply any desired transformations, and call the Static Shader to draw the objects in the scene.

```
// ---------------------------------------
// ESCENARIO
// ---------------------------------------
staticShader.use();
staticShader.setMat4("projection", projection);
staticShader.setMat4("view", view);

model = glm::mat4(1.0f);
model = glm::translate(model, glm::vec3(0.0f, 0.0f, 0.0f));
model = glm::scale(model, glm::vec3(30.0f));
staticShader.setMat4("model", model);
piso.Draw(staticShader);
```

3. Dynamic objects

The process for importing dynamic models is similar to that used for static objects, with the only difference being that it is necessary to import the objects in .dae format, as this is the only compatible animation file with the Anim Shader in this project.

The declaration of a dynamic object is as follows:

```
//MODELOS DINÁMICOS
// ----------
ModelAnim running("resources/objects/Deportista/Running.dae");
running.initShaders(animShader.ID);
```

Afterwards, the object is simply called and any necessary transformations are applied to it, as well as the use of the Anim Shader to draw the object.

```
// ANIMACIONES
// ------------------------------------------------------------------------------------------------------------
//Remember to activate the shader with the animation
animShader.use();
animShader.setMat4("projection", projection);
animShader.setMat4("view", view);
```

```
animShader.setVec3("material.specular", glm::vec3(0.5f));
animShader.setFloat("material.shininess", 32.0f);
animShader.setVec3("light.ambient", ambientColor);
animShader.setVec3("light.diffuse", diffuseColor);
animShader.setVec3("light.specular", 1.0f, 1.0f, 1.0f);
animShader.setVec3("light.direction", lightDirection);
animShader.setVec3("viewPos", camera.Position);

model = glm::translate(glm::mat4(1.0f), glm::vec3(-255.936f + movSport_x, 8.3391f,
-428.04f + movSport_z));
model = glm::rotate(model, glm::radians(orientation), glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::scale(model, glm::vec3(0.35f));
animShader.setMat4("model", model);
running.Draw(animShader);
```

This only draws the object in the scene, creating an infinite animation in its place. Therefore, to define a path, modifications to the Animate function must be made, as shown below.

4. Animation information

Car Animation:

The following global variables were created for purposes related to this animation:

```
float   movAuto_x = 0.0f,
        movAuto_z = 0.0f,
        orienta = 0.0f,
        girollantas = 0.0f,
        movAuto_y = 0.0f,
        escala = 30.0f;
```

In addition, the object was divided into body and wheel.

The movAuto variable applies the translation operation on the three axes (x, y, z) on the vertices that make up the car body. The orienta variable indicates a rotation angle for the body on the Y axis, and the girollantas variable gradually rotates the wheels on the X axis. Finally, the scale variable modifies the size of the object within the scene.

The car is drawn with the initial values, and a temporary model is created on the body which is then used to define it as the starting point for drawing the wheels, creating a hierarchical model that conditions the wheels to move with the body.

```
//COCHE
model = glm::mat4(1.0f);
model = glm::translate(model, glm::vec3(-91.692f + movAuto_x, 39.831f + movAuto_y,
92.421f + movAuto_z));
tmp = model = glm::rotate(model, glm::radians(orienta), glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::scale(model, glm::vec3(escala));
staticShader.setMat4("model", model);
coche.Draw(staticShader);

model = glm::translate(tmp, glm::vec3(-24.8199f, -20.589f, -61.383f));
model = glm::scale(model, glm::vec3(escala));
model = glm::rotate(model, glm::radians(girollantas), glm::vec3(1.0f, 0.0f, 0.0f));
staticShader.setMat4("model", model);
llanta.Draw(staticShader);//izq delantera

model = glm::translate(tmp, glm::vec3(25.05f, -20.589f, -61.383f));
model = glm::rotate(model, glm::radians(180.0f), glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::scale(model, glm::vec3(escala));
model = glm::rotate(model, glm::radians(girollantas), glm::vec3(-1.0f, 0.0f, 0.0f));
staticShader.setMat4("model", model);
llanta.Draw(staticShader);//der delantera

model = glm::translate(tmp, glm::vec3(-24.8199f, -20.589f, 29.823));
model = glm::scale(model, glm::vec3(escala));
model = glm::rotate(model, glm::radians(girollantas), glm::vec3(1.0f, 0.0f, 0.0f));
staticShader.setMat4("model", model);
llanta.Draw(staticShader);//izq trasera

model = glm::translate(tmp, glm::vec3(25.05f, -20.589f, 29.823f));
model = glm::rotate(model, glm::radians(180.0f), glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::scale(model, glm::vec3(escala));
```

```
model = glm::rotate(model, glm::radians(girollantas), glm::vec3(-1.0f, 0.0f, 0.0f));
staticShader.setMat4("model", model);
llanta.Draw(staticShader);//der trasera
```

For the car animation, a path within the scene was defined where the car will pass through different states, a total of 6 states from 1 to 6 which are represented by applying the Switch/Case control structure where the car and wheels will change their position, rotate, etc.

This animation ends when it reaches the sixth state, where the car changes its scale to 0 and therefore disappears from the scene.

```
//Vehículo
	if (animacion)
	{
		switch (car)
		{
		case 1:
			movAuto_z += 3.0f;
			girollantas += 3.0f;
			if (movAuto_z >= 240.0f) {
				orienta = -90.0f;
				movAuto_y = -9.5595;
				car = 2;
			}
			break;
		case 2:
			movAuto_x -= 3.0f;
			girollantas += 3.0f;
			if (movAuto_x <= -240.0f)
			{
				orienta = -180.0f;
				car = 3;
			}
			break;
```

```
            case 3:
                movAuto_z -= 3.0f;
                girollantas += 3.0f;
                if (movAuto_z <= 0.0f) {
                    car = 4;
                }
                break;
            case 4:
                movAuto_z += 5.0f;
                girollantas -= 5.0f;
                if (movAuto_z >= 320.0f) {
                    orienta = -90.0f;
                    car = 5;
                }
                break;
            case 5:
                movAuto_x += 5.0f;
                girollantas -= 5.0f;
                if (movAuto_x >= 500.0f) {
                    orienta = 0.0f;
                    car = 6;
                }
                break;
            case 6:
                movAuto_z -= 5.0f;
                girollantas -= 5.0f;
                if (movAuto_z <= -600.0f) {
                    escala = 0.0f;
                    animacion = false;
                }
                break;
            }
        }
```

To activate the animation, the user must press the 'SPACE' key and once the car has disappeared, they will have the possibility to reset the animation by pressing the 'R' key.

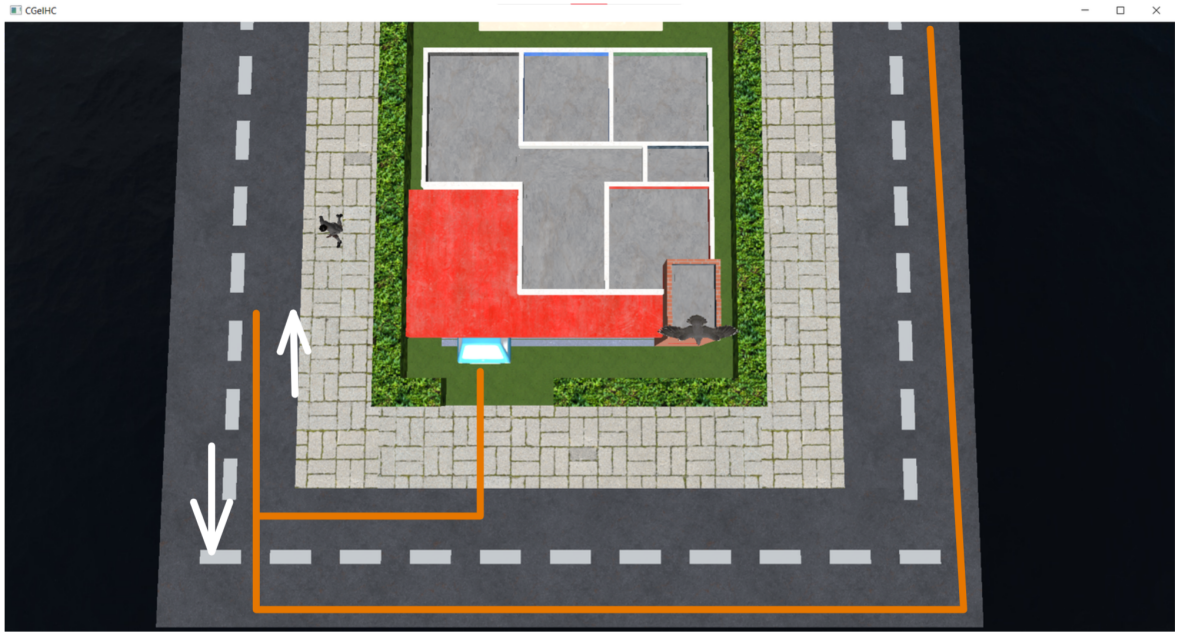

Figura 2. Route that the car follows until its animation ends.

Athlete Animation:

The following global variables were created to generate this animation:

```
float   movSport_x = 0.0f,
        movSport_z = 0.0f,
        orientation = 0.0f;
```

These control the movement of the athlete on the x and z axes, as well as a variable that modifies the orientation that the athlete is facing.

This animation is the only one that uses dynamic objects, specifically the athlete model. This model was downloaded from Mixamo along with the 'Running' animation, and it was specified that the animation had no displacement so that it could be generated through code.

The object was drawn through the Anim Shader, but before that, translation, rotation and scale transformations were applied to it along with the variables that allow rotation and movement on the X and Z axes.

```
model = glm::translate(glm::mat4(1.0f), glm::vec3(-255.936f + movSport_x, 8.3391f,
-428.04f + movSport_z));
model = glm::rotate(model, glm::radians(orientation), glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::scale(model, glm::vec3(0.35f));
animShader.setMat4("model", model);
running.Draw(animShader);
```

Through a Switch structure, the different states through which the athlete will pass are chosen, which in this case correspond to its movement on the X and Z axes. Within each Case is an if statement that makes a validation which allows the change of state and rotation of the athlete to follow a certain route.

```
switch (sport)
	{
	case 1:
		movSport_z += 5.0f;
		if (movSport_z >= 700.0f) {
			orientation = 90.0f;
			sport = 2;
		}
		break;
	case 2:
		movSport_x += 5.0f;
		if (movSport_x >= 500.0f) {
			orientation = 180.0f;
			sport = 3;
		}
		break;
	case 3:
		movSport_z -= 5.0f;
		if (movSport_z <= 0.0f)
		{
			orientation = 0.0f;
			movSport_z = 0;
			movSport_x = 0;
```

```
                sport = 1;
            }
            break;
        default:
            break;
    }
```

It is not necessary for the user to press a key for the animation to start executing, as it has been defined to repeat every cycle of the program.

Figura 3. Route that the athlete follows

Falcon Animation:

This animation was done using KeyFrames through the code's timeline, which is able to save the properties (translation, rotation, scale, etc.) of an object at a given moment and then reproduce them to create a simple animation through automatic interpolation.

The following global variables are used for this purpose, the first three are for modifying the position of the falcon on the three axes, rotAla allows to control the rotation of the Wings on the Z axis and giro to control the orientation of the falcon on the Y axis.

Next, the increments (inc) are defined, which will be the values for the intermediate frames between each KeyFrame, that is, the interpolation.

```
//Keyframes (Manipulación y dibujo)
float	posX = 0.0f,
		posY = 0.0f,
		posZ = 0.0f,
		rotAla = 0.0f,
		giro = 0.0f;
float	incX = 0.0f,
		incY = 0.0f,
		incZ = 0.0f,
		rotInc = 0.0f,
		giroInc = 0.0f;
```

Later, the functions that allow us to save the properties of the objects (the KeyFrames) and some others that are necessary for the interpolation are defined. Likewise, variables and constants are described to establish the maximum number of keyframes that can be saved, the number of steps between each frame, etc.

```
#define MAX_FRAMES 18
int i_max_steps = 60;
int i_curr_steps = 0;
typedef struct _frame
{
	//Variables para GUARDAR Key Frames
	float posX;		//Variable para PosicionX
	float posY;		//Variable para PosicionY
	float posZ;		//Variable para PosicionZ
	float rotAla;
	float giro;

}FRAME;

FRAME KeyFrame[MAX_FRAMES];
```

```
int FrameIndex = 17;                    //introducir datos
bool play = false;
int playIndex = 0;

void saveFrame(void)
{
        //printf("frameindex %d\n", FrameIndex);
        //std::cout << "Frame Index = " << FrameIndex << std::endl;

        KeyFrame[FrameIndex].posX = posX;
        KeyFrame[FrameIndex].posY = posY;
        KeyFrame[FrameIndex].posZ = posZ;

        KeyFrame[FrameIndex].rotAla = rotAla;
        KeyFrame[FrameIndex].giro = giro;

        cout << posX << endl;
        cout << posY << endl;
        cout << posZ << endl;
        cout << rotAla << endl;
        cout << giro << endl;

        FrameIndex++;
}

void resetElements(void)
{
        posX = KeyFrame[0].posX;
        posY = KeyFrame[0].posY;
        posZ = KeyFrame[0].posZ;

        rotAla = KeyFrame[0].rotAla;
        giro = KeyFrame[0].giro;
}
```

```
void interpolation(void)
{
	incX = (KeyFrame[playIndex + 1].posX - KeyFrame[playIndex].posX) /
i_max_steps;
	incY = (KeyFrame[playIndex + 1].posY - KeyFrame[playIndex].posY) /
i_max_steps;
	incZ = (KeyFrame[playIndex + 1].posZ - KeyFrame[playIndex].posZ) /
i_max_steps;

	rotInc = (KeyFrame[playIndex + 1].rotAla - KeyFrame[playIndex].rotAla) /
i_max_steps;
	giroInc = (KeyFrame[playIndex + 1].giro - KeyFrame[playIndex].giro) /
i_max_steps;

}
```

Then, in the Animate function, the steps and process followed by the interpolation when the animation is executed, that is, when "play" is given to interpolate between each saved KeyFrame, are described.

```
if (play)
	{
		if (i_curr_steps >= i_max_steps) //end of animation between frames?
		{
			playIndex++;
			if (playIndex > FrameIndex - 2)		//end of total animation?
			{
				//std::cout << "Animation ended" << std::endl;
				//printf("termina anim\n");
				playIndex = 0;
				play = false;
			}
			else //Next frame interpolations
			{
				i_curr_steps = 0; //Reset counter
```

```
                                                    //Interpolation
                        interpolation();
                    }
                }
                else
                {
                    //Draw animation
                    posX += incX;
                    posY += incY;
                    posZ += incZ;

                    rotAla += rotInc;
                    giro += giroInc;

                    i_curr_steps++;
                }
            }
```

In the program, to save a KeyFrame, the 'L' key must be pressed and to run the animation, the 'P' key must be pressed.

At this point, there is only one way to save KeyFrames to create animations that last only the duration of the program's execution, because once the program is closed, the animation will be lost.

For this reason, additional code was added to read .txt files that contained the values of the object properties for each KeyFrame, so that we can execute the animation by pressing the 'P' key, allowing us to create longer animations and store them in .txt files

```
//ANIMACIÓN KEYFRAMES CON ARCHIVO

    //1. Abrimos el archivo
    ifstream archivo;
    archivo.open("resources/animacion/keyframes.txt");
```

```
//2. Leemos línea por línea
if (archivo.is_open()) {
        for (int i = 0; i < MAX_FRAMES; i++)
        {
                for (int j = 0; j < 5; j++)
                {
                        getline(archivo, line);
                        switch (j) {
                        case 0: KeyFrame[i].posX = stof(line);
                                break;
                        case 1: KeyFrame[i].posY = stof(line);
                                break;
                        case 2: KeyFrame[i].posZ = stof(line);
                                break;
                        case 3: KeyFrame[i].rotAla = stof(line);
                                break;
                        case 4: KeyFrame[i].giro = stof(line);
                                break;
                        }
                }
        }
}

//3. Cerramos el archivo
archivo.close();
```

This is precisely what was done with the falcon, first an animation was created within the program and the object properties for each KeyFrame were stored in a .txt file which was then read, and by pressing the 'P' key, the interpolation was made to create the falcon's flight animation, in which its position and orientation are modified.

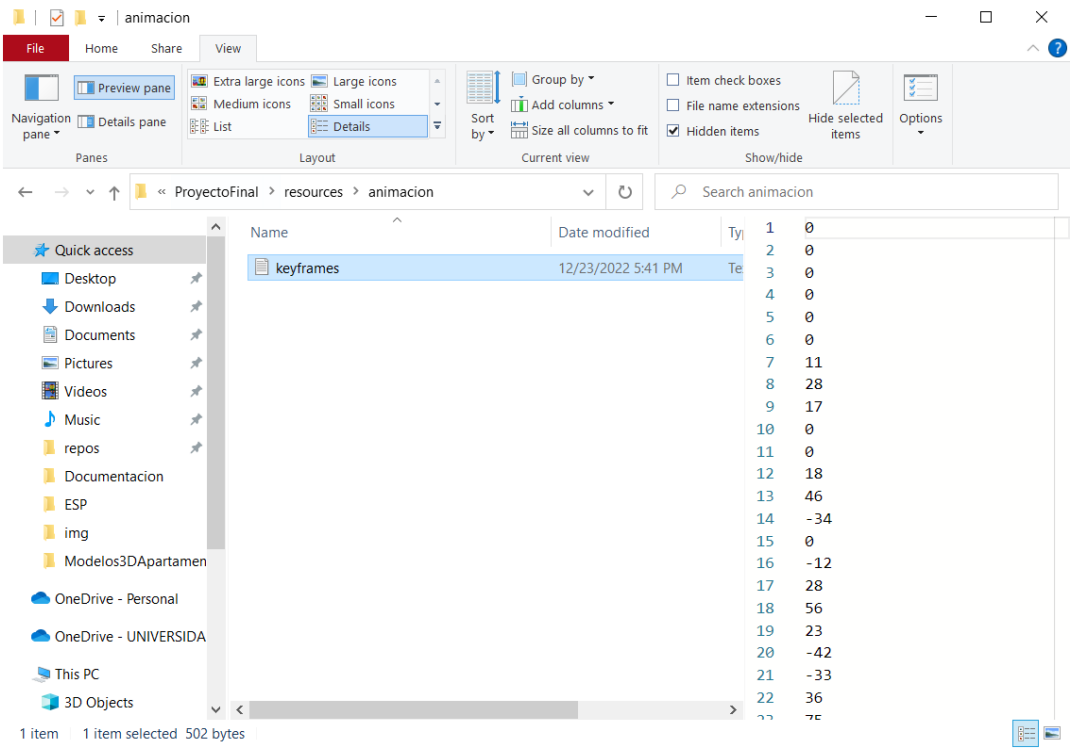

Figura 4. Location of the file containing the values for the KeyFrames.

Falcon drawing, it is drawn along with the variables that allow its change of position and rotation, in addition a temporary matrix is created that serves as a starting point to draw the wings and that they move with the falcon's body:

```
//AVE
model = glm::mat4(1.0f);
model = glm::translate(model, glm::vec3(119.259f, 155.056f, 139.356f));
model = glm::translate(model, glm::vec3(posX, posY, posZ));
tmp = model = glm::rotate(model, glm::radians(giro), glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::scale(model, glm::vec3(20.0f));
staticShader.setMat4("model", model);
cuerpo.Draw(staticShader);

//Ala izq
model = glm::translate(tmp, glm::vec3(-2.883f, 7.0504f, 0.51212f));
model = glm::rotate(model, glm::radians(-rotAla), glm::vec3(0.0f, 0.0f, 1.0f));
model = glm::scale(model, glm::vec3(20.0f));
staticShader.setMat4("model", model);
alaizq.Draw(staticShader);
```

```
//Ala der
model = glm::translate(tmp, glm::vec3(2.883f, 7.0504f, 0.51212f));
model = glm::rotate(model, glm::radians(rotAla), glm::vec3(0.0f, 0.0f, 1.0f));
model = glm::scale(model, glm::vec3(20.0f));
staticShader.setMat4("model", model);
alader.Draw(staticShader);
```

Reminder: To execute the animation, press the 'P' key.

Remote Control Helicopter Animation:

The following global variables were created for purposes related to this animation:

```
float   movHeli_x = 0.0f,
        movHeli_y = 0.0f,
        movHeli_z = 0.0f,
        orient = 0.0f;
```

Which, as already explained, allow to modify the position of the helicopter on the different reference axes as well as its rotation to point it in different directions.

The animation follows a very similar process to that of the car. The model was divided into the helicopter body and the propellers. In addition, its route is defined through states and an if-else structure. The helicopter has 6 different states or routes.

```
//Helicoptero
        if (animacion2) {
                if (recorrido1) {
                        movHeli_x = 0.0f;
                        movHeli_z += 3.0f;
                        orient = 0.0f;
                        if (movHeli_z >= 200.0f) {
                                recorrido1 = false;
                                recorrido5 = true;
                        }
                }
                else if (recorrido2) {
```

```
            movHeli_x -= 4.0f;
            movHeli_z = 200.0f;
            orient = 270.0f;
            if (movHeli_x <= -250.0f) {
                recorrido2 = false;
                recorrido3 = true;
            }
        }
        else if (recorrido3) {
            movHeli_x = -250.0f;
            movHeli_z -= 3.0f;
            orient = 180.0f;
            if (movHeli_z <= -200.0f) {
                recorrido3 = false;
                recorrido4 = true;
            }
        }
        else if (recorrido4) {
            movHeli_x += 4.0f;
            movHeli_z = -200.0f;
            orient = 90.0f;
            if (movHeli_x >= 0.0f) {
                recorrido4 = false;
                recorrido1 = true;
            }
        }
        else if (recorrido5) {
            movHeli_x -= 3.0f;
            movHeli_z -= 3.0f * (400.0f / 250.0f);
            orient = 212.0f;
            if (movHeli_x <= -250.0f && movHeli_z <= -200.0f) {
                recorrido5 = false;
                recorrido6 = true;
            }
        }
```

```
        else if (recorrido6) {
            movHeli_x += 3.0f;
            movHeli_z += 3.0f * (200.0f / 250.0f);
            orient = 90.0f - 38.65f;
            if (movHeli_x >= 0.0f && movHeli_z >= 0.0f) {
                recorrido6 = false;
                recorrido1 = true;
            }
        }
    };
```

The main difference with the car animation is that this animation has a higher complexity, as it includes turns that are made gradually and even changes in the helicopter's speed during its routes. To achieve this, mathematical calculations and experimentation through trial and error were required to achieve the expected behavior.

To activate the animation, press the '5' key.

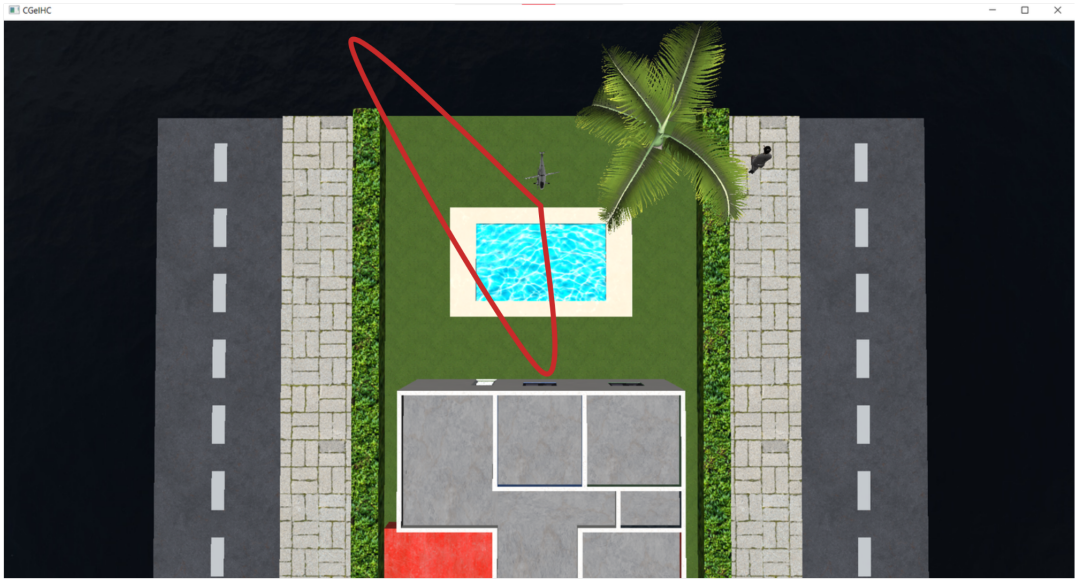

Figura 5. Curved route taken by the helicopter.

5. Conclusiones

The result obtained from this project has left me satisfied, since through the tools and skills acquired during the course it was possible to adequately reflect the idea captured in the proposal; creating a facade of a house with the essential elements and their respective exteriors to give it realism.

The project required a significant investment of time and effort due to the complexity of modeling, texturing, and animating models to meet the project requirements. However, I consider that there is the opportunity to add more details to the house, such as increasing the number of models, adding different types of lighting and animations, among others.

In conclusion, the objectives set out in this project have been achieved by succeeding in capturing the ideas developed in the proposal and applying the knowledge acquired during the laboratory and theory course. This experience will undoubtedly be valuable for my development as a Computer Engineer, since graphics have a wide presence in everyday life and it is important to have the necessary skills to perform effectively in the profession.

6. References

Computación Gráfica Introducción y Conceptos Fundamentales. (n.d.). U-Cursos. Retrieved December 8, 2022, from https://www.u-cursos.cl/ingenieria/2010/2/CC3501/1/material_docente/bajar%3Fid_material%3D306628

ilet. (n.d.). *La computación gráfica: Historia, objetivos y aplicaciones*. Ilet. Retrieved December 8, 2022, from https://ilet.mx/toluca/concepto-computacion-grafica/

Plano de casa de 1 piso con 3 habitaciones 2 baños (DWG). (n.d.). Verplanos.com. Retrieved November 18, 2022, from https://verplanos.com/plano-de-casa-con-cubierta-mediterranea-de-1-piso-y-3-habitaciones/

Villalta, H. (n.d.). *Tutorial Comentado Apartamento en Blender 3.0*. YouTube. Retrieved November 20, 2022, from https://www.youtube.com/watch?v=lekcqJNMGHA&t=7490s

7. Link to virtual tour

https://www.youtube.com/watch?v=XWHkK6jfjIg

8. Link to animation demostration

https://www.youtube.com/watch?v=0kX-gMDSoKM