Final Project-User Manual

National Autonomous University of Mexico

Computer Engineer

Ninth Semester

Members:
Amaro Cantoral Edgar

Account Number:
316625368

Subject: Computer Graphics and Human Interaction

Group: 5

Delivery date: 10/01/23

Index

Camera

To carry out the use of the synthetic camera the library called "camera.h" is imported, so that the main code of the project can use it.

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

Later to use all its functions, an object of that class is created along with its variables, so that its syntax is as follows:

```
45      // camera
46      Camera camera(glm::vec3(0.0f, 10.0f, 90.0f))
47      float MovementSpeed = 0.1f;
48      float lastX = SCR_WIDTH / 2.0f;
49      float lastY = SCR_HEIGHT / 2.0f;
50      bool firstMouse = true;
```

These values handle a default value that can be modified depending on the result expected at execution. For example, "Speed" represents the speed of the camera, which performs a translation according to the camera and this value mentioned above.

```
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 to move with the keyboard keys, a function called "my_input" is declared, which will allow detecting when a key is entered in the program or code. However, an IF control structure must be performed to recognize the key that has been entered, as well as the GLFW library to compare whether the key has been pressed, if this condition is met the operation assigned to it is performed.

For this case of the synthetic camera, using the keys "W," "S," "A" and "D" is specified to each key that direction the camera will move, as shown in the image above.

Static Objects

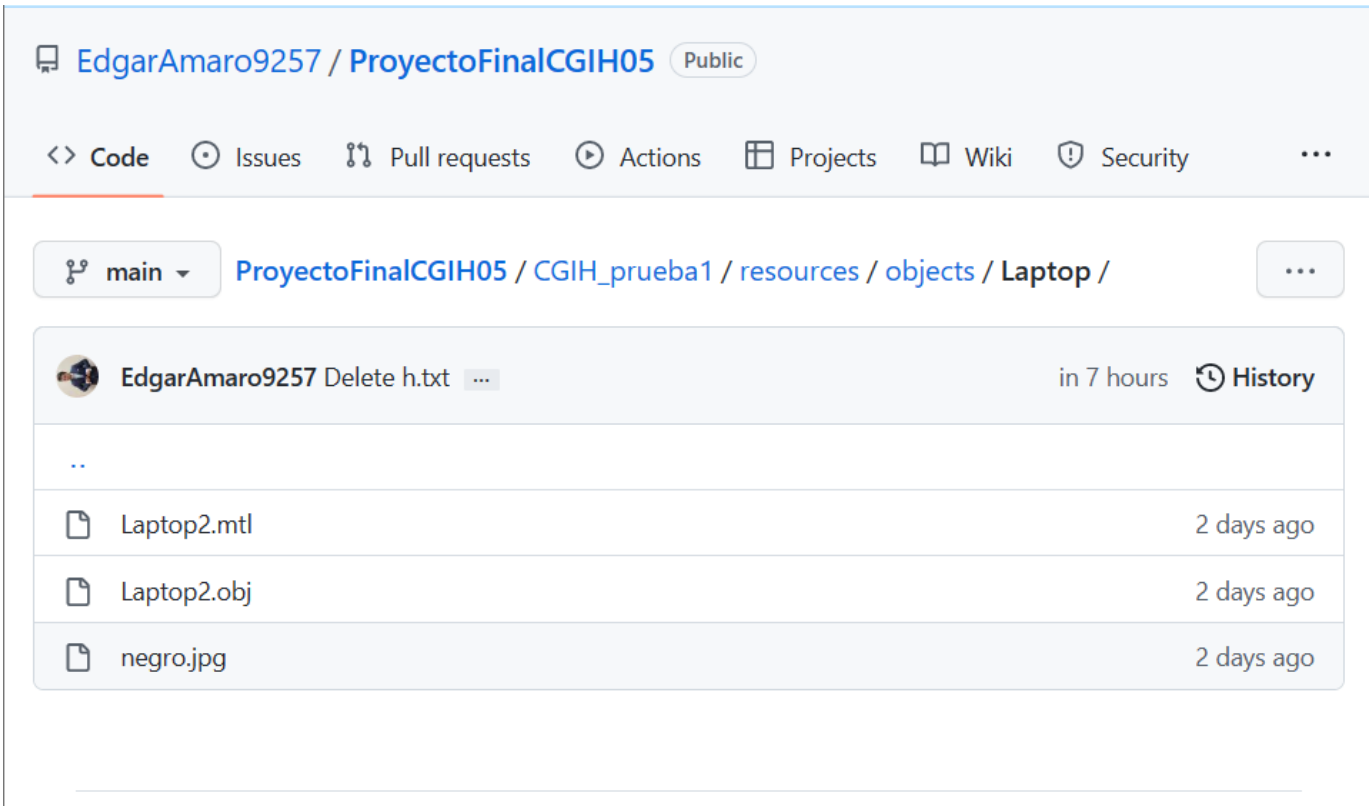
To use static objects in the project it is necessary to import the models with extension. "obj," since the program is thinking to use this type of file.

To import a file. "obj" into the project, you must enter the folder called "objects," which is located at the following address:

“ProyectoFinalCGIH05/CGIH_prueba1/resources”

Once you are in the folder you must create a folder with the name of the object, within it you will add both the textures as well as the file. "obj" and the "mtl," thus successfully importing the object. It should be noted that the file. "mtl" must have correctly assigned the address of the textures because otherwise, it will not load them in the executable program.

The following is an example of importing a static object:



Once the file with its corresponding textures has been successfully imported, that object must be declared. In this case, we make use of the function called Static Shader.

The declared objects are displayed:

The following is an example of importing a static object:

Final.cpp

CGIH_prueba1 (Ámbito global)

```
334     Model casaproy("resources/objects/Casaproyecto/house3.obj");
335     Model cama("resources/objects/Cama/cama.obj");
336     Model cama2("resources/objects/Cama/cama.obj");
337     Model cama3("resources/objects/Cama/cama.obj");
338     Model cama4("resources/objects/Cama/cama.obj");
339     Model cama5("resources/objects/Cama/cama.obj");
340     Model sidetable1("resources/objects/SideTable/Side_Table.obj");
341     Model sidetable2("resources/objects/SideTable/Side_Table.obj");
342     Model sidetable3("resources/objects/SideTable/Side_Table.obj");
343     Model sidetable4("resources/objects/SideTable/Side_Table.obj");
344     Model lampara("resources/objects/Lampara/lamp2.obj");
345     Model lampara2("resources/objects/Lampara/lamp2.obj");
346     Model lampara3("resources/objects/Lampara/lamp2.obj");
347     Model lampara4("resources/objects/Lampara/lamp2.obj");
348     Model escrit1("resources/objects/Escritorio/desk.obj");
349     Model escrit2("resources/objects/Escritorio/desk.obj");
350     Model escrit3("resources/objects/Escritorio/desk.obj");
351     Model laptop1("resources/objects/Laptop/Laptop2.obj");
352     Model laptop2("resources/objects/Laptop/Laptop2.obj");
353     Model laptop3("resources/objects/Laptop/Laptop2.obj");
354     Model sillaoff("resources/objects/SillaOffice/sillaModerno.obj");
355     Model sillaoff2("resources/objects/SillaOffice/sillaModerno.obj");
356     Model sillaoff3("resources/objects/SillaOffice/sillaModerno.obj");
```

Final.cpp

CGIH_prueba1 (Ámbito global)

```
346     Model lampara3("resources/objects/Lampara/lamp2.obj");
347     Model lampara4("resources/objects/Lampara/lamp2.obj");
348     Model escrit1("resources/objects/Escritorio/desk.obj");
349     Model escrit2("resources/objects/Escritorio/desk.obj");
350     Model escrit3("resources/objects/Escritorio/desk.obj");
351     Model laptop1("resources/objects/Laptop/Laptop2.obj");
352     Model laptop2("resources/objects/Laptop/Laptop2.obj");
353     Model laptop3("resources/objects/Laptop/Laptop2.obj");
354     Model sillaoff("resources/objects/SillaOffice/sillaModerno.obj");
355     Model sillaoff2("resources/objects/SillaOffice/sillaModerno.obj");
356     Model sillaoff3("resources/objects/SillaOffice/sillaModerno.obj");
357     Model cocina("resources/objects/Cocina/fkc.obj");
358     Model setmcs("resources/objects/MesaconSillas/Table_witch_chairs.obj");
359     Model sofa("resources/objects/Sofa/sofa3.obj");
360     Model piscina("resources/objects/PiscinaJardin/piscina.obj");
```

Finally, to complete this process you must call these objects and perform their corresponding transformations, as well as Static Shader so that it can be displayed in the executable program.

Here is an example of the process mentioned above:

```
//Lamparas
model = glm::translate(glm::mat4(1.0f), glm::vec3(-24.0f, 5.6f, -41.5f));
model = glm::scale(model, glm::vec3(0.05f));
staticShader.setMat4("model", model);
lampara.Draw(staticShader);

model = glm::translate(glm::mat4(1.0f), glm::vec3(0.0f, 5.6f, -41.5f));
model = glm::scale(model, glm::vec3(0.05f));
staticShader.setMat4("model", model);
lampara2.Draw(staticShader);
```

Dynamic Objects

For this type of objects, they are imported in the same way as static objects, with the difference that they only use ".dae files" and their respective textures, since a function called "Animate Shader" is used.

The dynamic object is first declared and then called by corresponding transformations, as well as using the "Animate Shader."

```
// ------------------------------------------------------------------------------------------
// Personaje Animacion
// ------------------------------------------------------------------------------------------
//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(-40.3f, 1.75f, 0.3f)); // translate it down so it'
model = glm::scale(model, glm::vec3(1.2f)); // it's a bit too big for our scene, so scale it down
model = glm::rotate(model, glm::radians(90.0f), glm::vec3(0.0f, 1.0f, 0.0f));
animShader.setMat4("model", model);
```

Finally, for that object to perform an animation in the environment, it is necessary to make the modifications to the function called "Animate," which will be seen below.

Information of the animations

1-Animación of yellow car Lamborghini:

To generate this animation, the following global variables were created:

```
// posiciones
//float x = 0.0f;
//float y = 0.0f;
float   movAuto_x = 0.0f,
        movAuto_y = 0.0f,
        movAuto_z = 0.0f,
        //avanza = 0.0f,
        orienta = 0.0f,
        giraLlanta = 0.0f;
```

The animation is compounded by the different objects:

- Box of car Lamborghini
- Wheels

It is important to note that the animation of the Lamborghini car along with its model 3D were previously created by my Laboratory professor Sergio Valencia to perform some animation practices. This animation that I proposed and will see next will modify it, that is, recycle a model 3D with the difference that it modifies the kind of animation that it will perform for the executable program.

Still, the construction of the yellow Lamborghini car is explained below.

First, we draw it using the motion variables, as well as applying their corresponding transformations so that the program shows the yellow car together are their corresponding tires.

```
// ------------------------------------------------------------------------------------
// Carro
// ------------------------------------------------------------------------------------
model = glm::rotate(glm::mat4(1.0f), glm::radians(90.0f), glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::translate(model, glm::vec3(15.0f + movAuto_x,  movAuto_y, movAuto_z));
tmp = model = glm::rotate(model, glm::radians(orienta), glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::scale(model, glm::vec3(0.1f, 0.1f, 0.1f));
staticShader.setMat4("model", model);
carro.Draw(staticShader);

model = glm::translate(tmp, glm::vec3(8.5f, 2.5f, 12.9f));
model = glm::scale(model, glm::vec3(0.1f, 0.1f, 0.1f));
model = glm::rotate(model, glm::radians(giraLlanta), glm::vec3(1.0f, 0.0f, 0.0f));
staticShader.setMat4("model", model);
llanta.Draw(staticShader);  //Izq delantera

model = glm::translate(tmp, glm::vec3(-8.5f, 2.5f, 12.9f));
model = glm::scale(model, glm::vec3(0.1f, 0.1f, 0.1f));
model = glm::rotate(model, glm::radians(180.0f), glm::vec3(0.0f, 1.0f, 0.0f));
staticShader.setMat4("model", model);
llanta.Draw(staticShader);  //Der delantera

model = glm::translate(tmp, glm::vec3(-8.5f, 2.5f, -14.5f));
model = glm::scale(model, glm::vec3(0.1f, 0.1f, 0.1f));
model = glm::rotate(model, glm::radians(180.0f), glm::vec3(0.0f, 1.0f, 0.0f));
staticShader.setMat4("model", model);
llanta.Draw(staticShader);  //Der trasera

model = glm::translate(tmp, glm::vec3(8.5f, 2.5f, -14.5f));
model = glm::scale(model, glm::vec3(0.1f, 0.1f, 0.1f));
staticShader.setMat4("model", model);
llanta.Draw(staticShader);  //Izq trase
```

For the animation of the cart a state machine was made using a global variable that was declared at the beginning of the code called "advance." Another important variable was also declared to carry out this technique, called "animation."

```
79      int     avanza = 0;

73      bool    animacion = false,
```

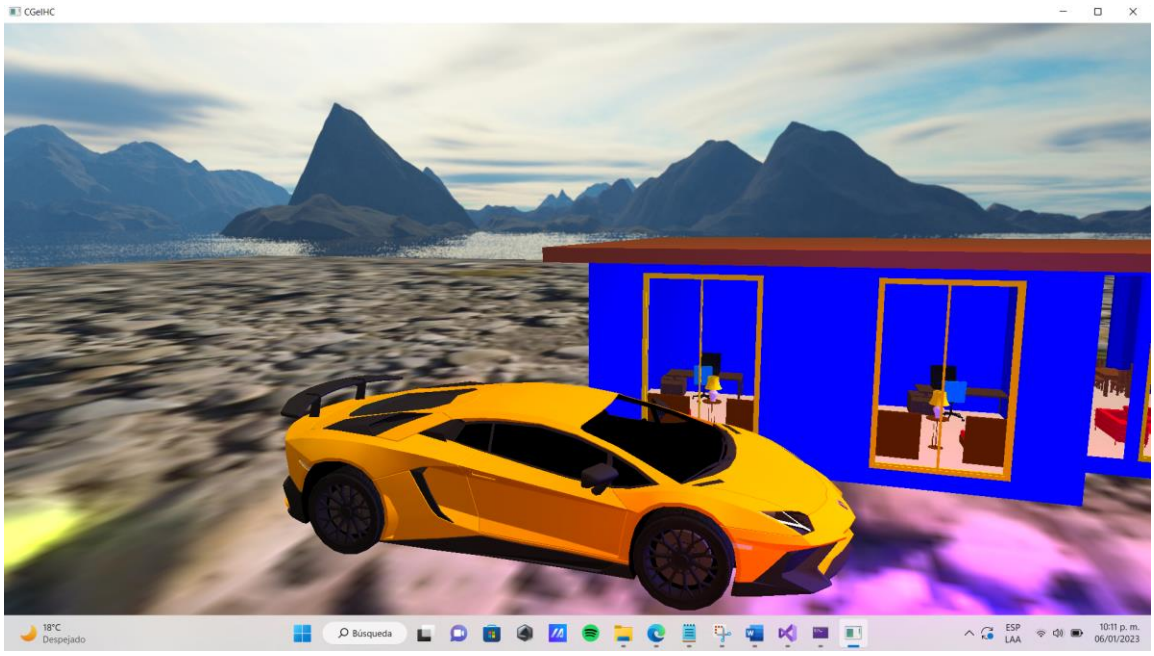
Finally, in the function "animate (void)," we create an IF conditional structure to perform the state machine sequence. In this conditional you have only 2 states that are 0 and 1. For example, if the variable "forward" is 0, then the car will reverse at the z-axis position by the variable "movAuto_z" and if this variable is less than -180.0 then forward is equal to one and move forward on the same z-axis.

```
//Vehículo, que tan rapido se va a mover
if (animacion) //movAuto es la variable que guarda
{
    //ANIMACION DEL AUTO LAMBORGHINI
    //6 estados
    if (avanza == 0.0f)  //Con esto se mueve adela
    {
        movAuto_z -= 1.0f; //SE MUEVE EN REVERSA
        if (movAuto_z <= -180.0f)
            avanza = 1.0;
    }

    if (avanza == 1.0f)
    {
        movAuto_z += 1.0f; //Avanza
        if (movAuto_z >= 180.0f)
            avanza = 0.0f;
    }


}
```

To play that animation, press the "space" key because it is the one that was previously assigned and the car will start to move infinitely back and forth. To pause or stop the animation, press the space key again.



2- Animation of Stewie in Aquaman costume:

For this animation by keyboards the following global variables were created:

```
//Keyframes (Manipulación y dibujo)
float	posX = 0.0f,
		posY = 0.0f,
		posZ = 0.0f,
		rotRodIzq = 0.0f,
		giroMonito = 0.0f,
		movBrazoDer = 0.0f;
float	incX = 0.0f,
		incY = 0.0f,
		incZ = 0.0f,
		rotInc = 0.0f,
		giroMonitoInc = 0.0f,
		movBrazoDerInc = 0.0f,
		ambosBrazos = 0.0f,
		ambosBrazosInc = 0.0f,
		ambasPiernas = 0.0f,
		ambasPiernasInc = 0.0f,
		cabezonInc = 0.0f,
		cabezon = 0.0f;
```

This character also comes from a laboratory practice taught by Professor Sergio Valencia, this character is being reused with his animation that will explain below. The character is composed of 7 objects that represent each part of his body as his arms and legs and each part of his body contains animations.

To draw these objects with their transformations was done as follows:

```
// ------------------------------------------------------------------------------------------
// Personaje
// ------------------------------------------------------------------------------------------
model = glm::translate(glm::mat4(1.0f), glm::vec3(0, 0, 0));
model = glm::translate(model, glm::vec3(posX, posY, posZ));
tmp = model = glm::rotate(model, glm::radians(giroMonito), glm::vec3(0.0f, 1.0f, 0.0));
staticShader.setMat4("model", model);
torso.Draw(staticShader);

//Pierna Der
model = glm::translate(tmp, glm::vec3(-0.5f, 0.0f, -0.1f));
model = glm::rotate(model, glm::radians(ambasPiernas), glm::vec3(0.0f, 1.0f, 0.0));
model = glm::rotate(model, glm::radians(-rotRodIzq), glm::vec3(1.0f, 0.0f, 0.0f));
staticShader.setMat4("model", model);
piernaDer.Draw(staticShader);

//Pie Der
model = glm::translate(model, glm::vec3(0, -0.9f, -0.2f));
staticShader.setMat4("model", model);
botaDer.Draw(staticShader);

//Pierna Izq
model = glm::translate(tmp, glm::vec3(0.5f, 0.0f, -0.1f));
model = glm::rotate(model, glm::radians(ambasPiernas), glm::vec3(0.0f, 1.0f, 0.0f));
staticShader.setMat4("model", model);
piernaIzq.Draw(staticShader);

//Pie Iz
model = glm::translate(model, glm::vec3(0, -0.9f, -0.2f));
staticShader.setMat4("model", model);
botaDer.Draw(staticShader); //Izq trase


//Brazo derecho
model = glm::translate(tmp, glm::vec3(0.0f, -1.0f, 0.0f));
model = glm::translate(model, glm::vec3(-0.75f, 2.5f, 0));
model = glm::rotate(model, glm::radians(ambosBrazos), glm::vec3(0.0f, 0.0f, 1.0f));
staticShader.setMat4("model", model);
brazoDer.Draw(staticShader);

//Brazo izquierdo
model = glm::translate(tmp, glm::vec3(0.0f, -1.0f, 0.0f));
model = glm::translate(model, glm::vec3(0.75f, 2.5f, 0));
model = glm::rotate(model, glm::radians(ambosBrazos), glm::vec3(1.0f, 0.0f, 0.0f));
staticShader.setMat4("model", model);
brazoIzq.Draw(staticShader);

//Cabeza
model = glm::translate(tmp, glm::vec3(0.0f, -1.0f, 0.0f));
model = glm::rotate(model, glm::radians(cabezon), glm::vec3(0.0f, 1.0f, 0.0));
model = glm::translate(model, glm::vec3(0.0f, 2.5f, 0));
staticShader.setMat4("model", model);
cabeza.Draw(staticShader);
// ------------------------------------------------------------------------------------------
```

To perform the animation of each part of its body, all the variables created above were used and assigned to each corresponding function for animation using key boxes.

The function are as follows:

```
//Keyframes (Manipulación y dibujo)
float   posX = 0.0f,
        posY = 0.0f,
        posZ = 0.0f,
        rotRodIzq = 0.0f,
        giroMonito = 0.0f,
        movBrazoDer = 0.0f;
float   incX = 0.0f,
        incY = 0.0f,
        incZ = 0.0f,
        rotInc = 0.0f,
        giroMonitoInc = 0.0f,
        movBrazoDerInc = 0.0f,
        ambosBrazos = 0.0f,
        ambosBrazosInc = 0.0f,
        ambasPiernas = 0.0f,
        ambasPiernasInc = 0.0f,
        cabezonInc = 0.0f,
        cabezon = 0.0f;
```

```
#define MAX_FRAMES 9  //ESTA LIMITADO A 9 NUEVE CUADROS POR FRAME
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 rotRodIzq;
    float giroMonito;
    float movBrazoDer;
    float ambosBrazos;
    float ambasPiernas;
    float cabezon;

}FRAME;
```

```
FRAME KeyFrame[MAX_FRAMES];
int FrameIndex = 0;         //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].rotRodIzq = rotRodIzq;
    KeyFrame[FrameIndex].movBrazoDer = movBrazoDer;
    KeyFrame[FrameIndex].ambosBrazos = ambosBrazos;
    KeyFrame[FrameIndex].ambasPiernas = ambasPiernas;
    KeyFrame[FrameIndex].cabezon = cabezon;

    FrameIndex++;
}
```

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

    rotRodIzq = KeyFrame[0].rotRodIzq;
    giroMonito = KeyFrame[0].giroMonito;
    movBrazoDer = KeyFrame[0].movBrazoDer;
    ambosBrazos = KeyFrame[0].ambosBrazos;
    ambasPiernas = KeyFrame[0].ambasPiernas;
    cabezon = KeyFrame[0].cabezon;

}
```

```
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].rotRodIzq - KeyFrame[playIndex].rotRodIzq) / i_max_steps;
    giroMonitoInc = (KeyFrame[playIndex + 1].giroMonito - KeyFrame[playIndex].giroMonito) / i_max_steps;
    ambosBrazosInc = (KeyFrame[playIndex + 1].ambosBrazos - KeyFrame[playIndex].ambosBrazos) / i_max_steps;
    ambasPiernasInc = (KeyFrame[playIndex + 1].ambasPiernas - KeyFrame[playIndex].ambasPiernas) / i_max_steps;
    cabezonInc = (KeyFrame[playIndex + 1].cabezon - KeyFrame[playIndex].cabezon) / i_max_steps;

}
```

```
for (int i = 0; i < MAX_FRAMES; i++)
{
    KeyFrame[i].posX = 0;
    KeyFrame[i].posY = 0;
    KeyFrame[i].posZ = 0;
    KeyFrame[i].rotRodIzq = 0;
    KeyFrame[i].giroMonito = 0;
    KeyFrame[i].movBrazoDer = 0;
    KeyFrame[i].ambosBrazos = 0;
    KeyFrame[i].ambasPiernas = 0;
    KeyFrame[i].cabezon = 0;
}
```

Once assigned their corresponding functions for each variable we proceed to put them into action with the function "animate (void)."

```
void animate(void)            //Se ejecuta cada ciclo de programa
{
    //ANIMANDO una trayectoria DE UN CIRCULO, el de ELIPE es SOLO MODIFICAR
    //lightPosition.x = 300.0f * cos(tiempo);
    //lightPosition.y = 300.0f * cos(tiempo);
    //lightPosition.z = 300.0f * sin(tiempo);
    //tiempo += 0.1f;

    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;

            rotRodIzq += rotInc;
            giroMonito += giroMonitoInc;
            movBrazoDer += movBrazoDerInc;
            ambosBrazos += ambosBrazosInc;
            ambasPiernas += ambasPiernasInc;
            cabezon += cabezonInc;

            i_curr_steps++;
        }
    }
```

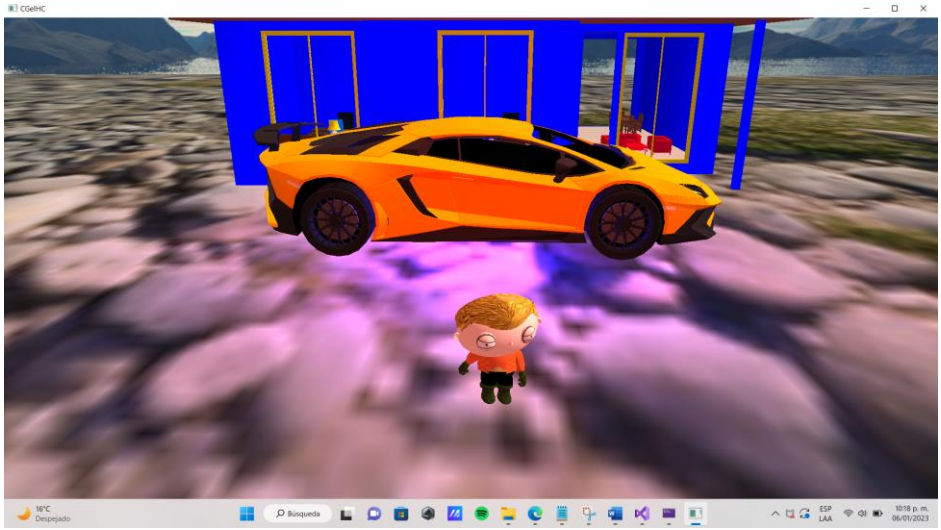
Finally, the animation is composed of the following keys:

- L Key: Save Frame
- P Key: Play animation
- Y Key: Move character towards positive Z-axis (forward)
- H Key: Move character towards negative Z-axis (backward)
- J Key: Move character towards positive X-axis (left)
- G Key: Move character towards negative X-axis (right)
- M Key: Move character towards positive Y-axis (up)
- N Key: Move character towards negative Y-axis (down)
- V Key: Move character 360 degrees towards left
- B Key: Move character 360 degrees towards right
- 4 Key: Move both arms up
- 3 Key: Move both arms down
- 5 Key: Move both legs towards right
- 6 Key: Move both legs towards left
- 7 Key: Move head towards right
- 8 Key: Move head towards left

To perform the animation, the following is done:

1. At the original position of the character press the L key to save that position. If this action is not performed, no animation can be played.
2. Move the character along with some part of your body in the position you want.
3. After moving the character press the L key again to save its position in a frame. *
4. If I finish saving the boxes to the desired positions, press the P key to play the final animation using the boxes you saved.

* You must save more than 1 frame to perform the animation.



Bibliography

- 3D Models for Professionals: (s. f.). TurboSquid. https://www.turbosquid.com/
- CGTrader - 3D Model Store. (s. f.). CGTrader. https://www.cgtrader.com/