Proyecto Final-Manual de usuario

Universidad Nacional Autónoma de México

Ingeniería en Computación

Noveno semestre

Integrantes:
Amaro Cantoral Edgar

Número de Cuenta:
316625368

Materia: Computación Gráfica e Interacción Humana

Grupo: 5
Fecha de entrega: 10/01/23

Índice

Proyecto Final de Computación Gráfica e Interacción Humana

Cámara 3

Objetos estáticos 4

Objetos dinámicos 6

Información de las animaciones 6

Bibliografía 14

Cámara

Para llevar a cabo la utilización de la cámara sintética se importa la librería llamada "camera.h", de manera que el código principal del proyecto pueda utilizarla.

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

Posteriormente para utilizar todas sus funciones, se crea un objeto de dicha clase junto con sus variables, de manera que su sintaxis es la siguiente:

```
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;
```

Estos valores manejan un valor predeterminado que puede ser modificado dependiendo del resultado esperado en la ejecución. Por ejemplo, "MovementSpeed" representa la velocidad de la cámara, la cual realiza una traslación de acuerdo a la cámara y este valor mencionado anteriormente.

```
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);
```

Para que la cámara se mueva con las teclas del teclado, se declara una función llamada "my_input", la cuál permitirá detectar cuando se ingresa una tecla en el programa u código. Sin embargo, se debe realizar tanto una estructura de control IF para reconocer la tecla que se ha ingresado, como la librería de GLFW para comparar si la tecla se ha presionado, si esta condición se cumple se realiza la operación asignada a esta.

Para este caso de la cámara sintética, mediante las teclas "W", "S", "A" y "D" se especifica a cada tecla que dirección se moverá la cámara, como se muestra en la imagen anterior.

Objetos estáticos

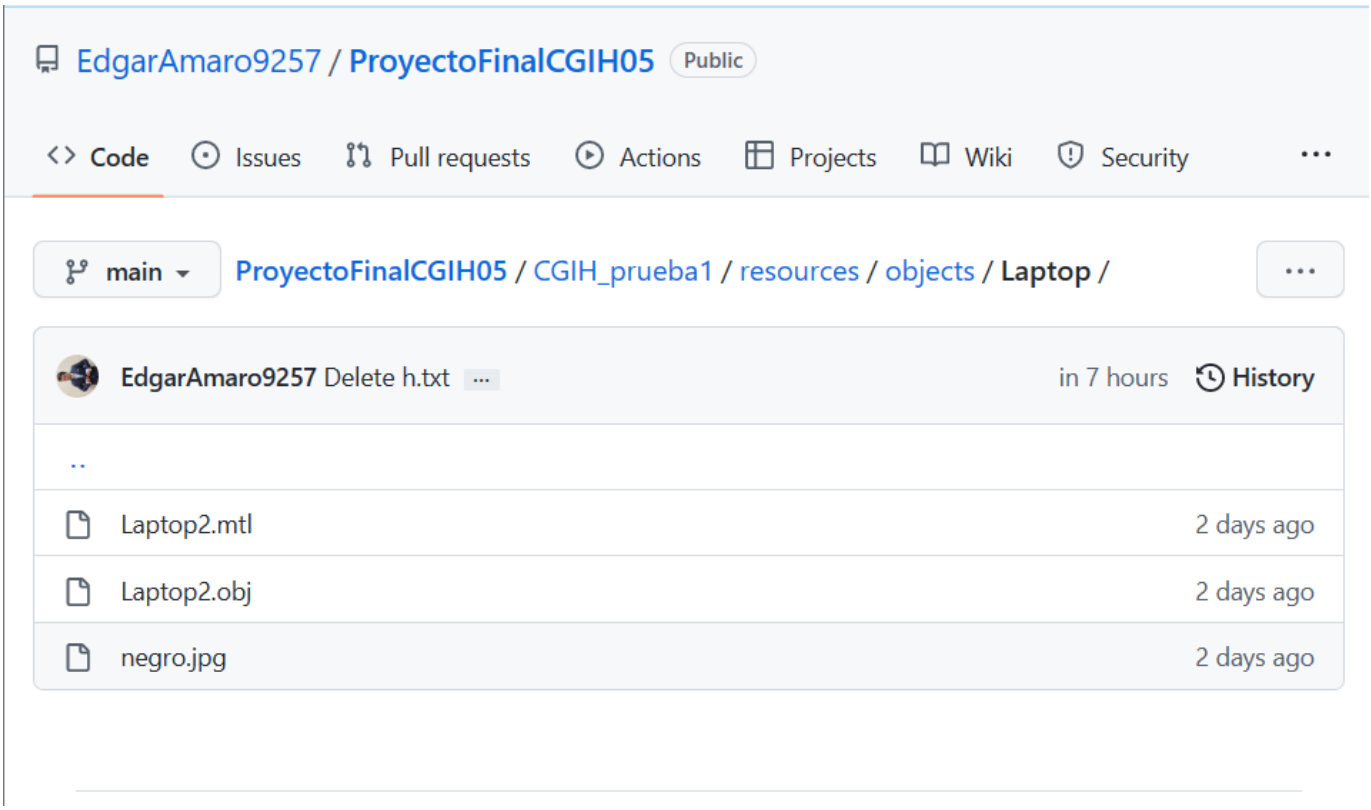
Para utilizar objetos estáticos en el proyecto es necesario importar los modelos con extensión “.obj”, ya que el programa está pensando para utilizar este tipo de archivo.

Para importar un archivo “.obj” al proyecto se requiere ingresar a la carpeta llamada “objects”, ésta se encuentra en la siguiente dirección:

“ProyectoFinalCGIH05/CGIH_prueba1/resources”

Una vez que se encuentra en la carpeta se debe crear una carpeta con el nombre del objeto, dentro de ésta misma se agregarán tanto las texturas como el archivo “.obj” y el “.mtl”, de esta manera se han importado correctamente el objeto. Cabe resaltar que el archivo “.mtl” debe tener asignada correctamente la dirección de las texturas porque de lo contrario, esta no las cargará en el programa ejecutable.

A continuación, verá un ejemplo de importación de un objeto estático:



Una vez que se ha importado correctamente el archivo con sus texturas correspondientes, se debe declarar dicho objeto. En este caso, hacemos uso de la función llamada “Static Shader”.

Se muestran los objetos declarados:

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");
```

Finalmente, para completar dicho proceso hay que llamar estos objetos y realizar sus transformaciones correspondientes, así como el Static Shader para que pueda mostrarse en el programa ejecutable.

Este es un ejemplo del proceso mencionado anteriormente:

```
//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);
```

Objetos dinámicos

Para este tipo de objetos se importan de la misma forma que los objetos estáticos, con la diferencia de que solo utilizan “archivos .dae” y sus respectivas texturas, ya que se utiliza una función llamada “Animate Shader.”

Primero se declara el objeto dinámico y posteriormente se llama dicho objeto mediante transformaciones correspondientes, así como también utilizar el “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);
```

Finalmente, para que dicho objeto realice una animación en el entorno, se requiere realizar las modificaciones en la función llamada “Animate”, las que se verán a continuación.

Información de las animaciones

1-Animación de carro amarillo Lamborghini:

Para generar dicha animación se crearon las siguientes variables globales:

```
// 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;
```

La animación está compuesta de diferentes objetos:

- Cuadro del carro Lamborghini
- Ruedas

Es importante resaltar que la animación del carro Lamborghini junto con su modelo 3D fueron creados previamente por mi profesor de Laboratorio Sergio Valencia para realizar algunas prácticas de animación. Esta animación que propuse y se verá a continuación lo modifique, es decir, recicle un modelo 3D con la diferencia de que modifique la clase de animación que realizará para el programa ejecutable.

Aun así, se explica a continuación la construcción del coche amarillo Lamborghini.

Primero lo dibujamos mediante las variables de movimiento, así como aplicando sus correspondientes transformaciones para que el programa muestre el coche amarillo junto son sus llantas correspondientes.

```
// -------------------------------------------------------------------------------------
// 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
```

Para la animación del carro se hizo una máquina de estados mediante una variable global que se declaró al principio del código llamado “avanza”. También se declaró otra variable importante para llevar a cabo ésta técnica, se llama “animation”.

```
79      int     avanza = 0;

73      bool    animacion = false,
```

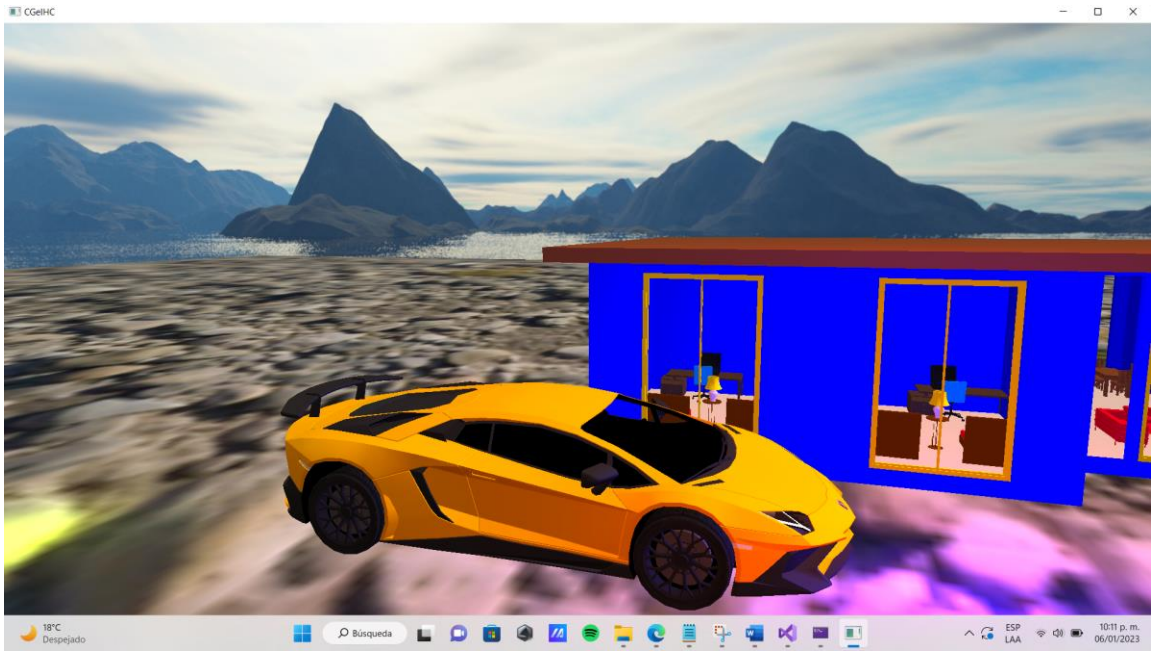
Por último, en la función “animate(void)”, creamos una estructura de condicional IF para llevar a cabo la secuencia de máquina de estados. En esta condicional solo se tienen 2 estados que son 0 y 1. Por ejemplo, si la variable “avanza” es 0, entonces el auto se moverá en reversa en la posición del eje z por la variable “movAuto_z” y si esta variable es menor igual a -180.0 entonces avanza es igual a uno y se moverá hacia adelante en el mismo eje z.

```
//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;
    }


}
```

Para reproducir dicha animación, se presiona la tecla “espacio” ya que es la que se asignó previamente y el coche comenzará a moverse de forma infinita hacia atrás y hacia adelante. Para pausar o detener la animación se vuelve a presionar la tecla de espacio.



2-Animación de Stewie con disfraz de Aquaman:

Para esta animación por cuadros claves se crearon las siguientes variables globales:

```
//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;
```

Este personaje también proviene de una práctica de laboratorio impartida por el profesor Sergio Valencia, se está reutilizando este personaje con su animación que explicare a continuación. El personaje está compuesto de 7 objetos que representan cada parte de su cuerpo como sus brazos y piernas y cada parte de su cuerpo contiene animaciones.

Para dibujar estos objetos con sus transformaciones se realizó de la siguiente forma:

```
// ------------------------------------------------------------------------------------------
// 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);
// ------------------------------------------------------------------------------------------
```

Para llevar cabo la animación de cada parte de su cuerpo se utilizaron todas las variables creadas anteriormente y se asignaron a cada función correspondiente para la animación mediante cuadros por clave.

Las funciones son las siguientes:

```
//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;
}
```

Una vez asignado sus funciones correspondientes para cada variable procedemos a ponerlos en acción con la función “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++;
        }
    }
```

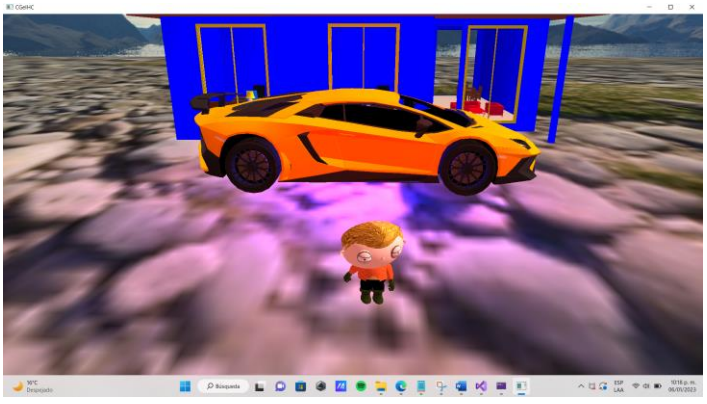
Finalmente, la animación queda compuesta por las siguientes teclas:

- Tecla L: Guardar cuadro
- Tecla P: Reproducir animación
- Tecla Y: Mover personaje hacia eje Z positivo (adelante)
- Tecla H: Mover personaje hacia eje Z negativo (atrás)
- Tecla J: Mover personaje hacia eje X positivo (izquierda)
- Tecla G: Mover personaje hacia eje X negativo (derecha)
- Tecla M: Mover personaje hacia eje Y positivo (arriba)
- Tecla N: Mover personaje hacia eje Y negativo (abajo)
- Tecla V: Mover personaje 360 grados hacia la izquierda
- Tecla B: Mover personaje 360 grados hacia la derecha
- Tecla 4: Mover ambos brazos hacia arriba
- Tecla 3: Mover ambos brazos hacia abajo
- Tecla 5: Mover ambas piernas hacia la derecha
- Tecla 6: Mover ambas piernas hacia la izquierda
- Tecla 7: Mover cabeza hacia la derecha
- Tecla 8: Mover cabeza hacia la izquierda

Para llevar a cabo la animación se hace lo siguiente:

1. En la posición original que se encuentra el personaje presionar la tecla L para guardar dicha posición. Si no se realiza esta acción no se podrá reproducir ninguna animación.
2. Mover el personaje junto con alguna parte de su cuerpo en la posición que usted quiera.
3. Después de mover el personaje vuelva a presionar la tecla L para guardar su posición en un cuadro o frame. *
4. Si termino de guardar los cuadros en las posiciones que quiera, presione la tecla P para reproducir la animación final mediante los cuadros que guardó.

*Debe guardar más de 1 cuadro para efectuar dicha animación.



Bibliografía

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