



# **Universidad Nacional Autónoma de México.**

## **Facultad de ingeniería.**

### ***Proyecto Final - Manual Técnico.***

**Materia:** Computación grafica e interacción humano-computadora.

**Profesor:** Carlos Aldair Roman Balbuena.

**Alumno:** Vaquero Barajas Alexis.

**Semestre:** 2022-2.

**Fecha de entrega:** 27 de Mayo de 2022.

## **Proyecto final.**

### **Contenido.**

<b>Cámara.</b>	<b>3</b>
<b>Objetos estáticos.</b>	<b>4</b>
<b>Objetos dinámicos.</b>	<b>6</b>
<b>Información de las animaciones.</b>	<b>7</b>
<b>Referencias.</b>	<b>17</b>

## Cámara.

Para poder utilizar una cámara sintética se importa la librería **camera.h**, de manera que el programa principal pueda utilizarla.

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

Posteriormente para utilizar las funciones de esta librería, se crea un objeto de esta clase, de manera que la sintaxis es la siguiente:

```
// camera
Camera camera(glm::vec3(0.0f, 50.0f, 200.0f));
```

Dentro de esta librería se encuentran las siguientes constantes:

```
18 // Default camera values
19 const float YAW      = -90.0f;
20 const float PITCH    =  0.0f;
21 const float SPEED    =  0.3f;
22 const float SENSITIVITY = 0.7f;
23 const float ZOOM     =  45.0f;
```

Estas constantes manejan un valor predeterminado que puede cambiarse dependiendo el resultado esperado, por ejemplo, modificar **SPEED** involucra la velocidad en cómo se mueve la cámara, en otras palabras, realiza una traslación de acuerdo con el valor de **SPEED**.

```
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);
```

Para que la cámara se mueva con las teclas del teclado, se declara la función **my\_input**, la cual permitirá detectar cuando se ingresa una tecla en el programa. Para que este reconozca que tecla se ha ingresado usamos una estructura de control IF y mediante la librería de GLFW comparamos si dicha tecla se ha presionado, si esta condición se cumple se realiza la operación asignada a esta.

Para el caso de la cámara mediante **FORWARD**, **BACKWARD**, **LEFT** y **RIGHT** se especifica a cada tecla que dirección se moverá la cámara, como se muestra a continuación.

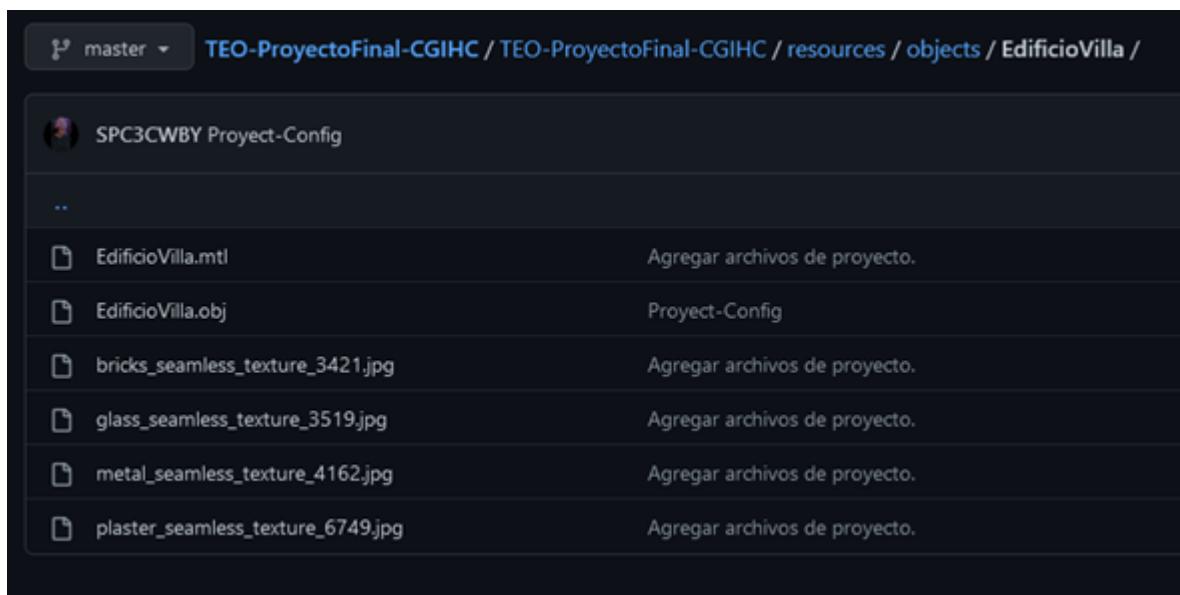
### Objetos estáticos.

Para utilizar objetos estáticos en el proyecto es necesario importar los modelos como **.obj**, ya que el programa está pensado para utilizar este tipo de archivo.

Para importar un archivo **.obj** al proyecto se requiere ingresar a la carpeta **objects**, esta se encuentra en la siguiente dirección: **TEO-ProyectoFinal-CGIHC/TEO-ProyectoFinal-CGIHC/resources**.

Una vez se encuentra en la carpeta se debe crear una carpeta con el nombre del objeto, dentro de esta misma se agregarán las **texturas**, el **archivo .obj** y el **.mtl**, de esta forma se han importado correctamente el objeto. Cabe aclarar que el archivo **.mtl** debe tener bien la dirección de la texturas, ya que de no ser así, esta no las cargará en el programa.

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



Una vez se ha importado correctamente el archivo, es necesario declarar dicho objeto. En este caso, al tratarse de un objeto estático, se utiliza el **Static Shader**.

A continuación, se muestran los objetos declarados:

```
// Modelos Estaticos
// -----
Model oxxo("resources/objects/oxxo/oxxo.obj"); // Oxxo
Model entrada("resources/objects/Entrada/Entrada.obj"); //entrada a la unidad
Model Carro("resources/objects/Bocho/Bocho.obj"); //Carroceria del bocho
Model llanta("resources/objects/Bocho/Rueda.obj"); //ruedas
Model plano("resources/objects/Plano/planoVilla.obj"); //plano de la unidad
Model edificiosVilla("resources/objects/EdificioVilla/EdificioVilla.obj"); //edificios de la unidad
Model iglesia("resources/objects/Iglesia/iglesia.obj"); //iglesia
Model arbusto("resources/objects/arbusto/arbusto.obj"); //arbusto
Model pedales("resources/objects/bicicleta/pedales.obj"); //Bici
Model cuadro("resources/objects/bicicleta/cuadro.obj");
Model rueda("resources/objects/bicicleta/rueda.obj");
Model courtBasket("resources/objects/CanchaBasquet/cancha.obj"); //Cancha Basquetbol
Model cuarto1("resources/objects/Cuarto_1/cuarto1.obj"); //Cuarto 1
Model cuarto2("resources/objects/Cuarto_2/cuarto2.obj"); //Cuarto 2
```

Finalmente, solo queda llamar a estos objetos y realizar sus transformaciones correspondientes, así como el **Static Shader** para que pueda mostrarse en el programa.

Un ejemplo es el siguiente:

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

model = glm::translate(glm::mat4(1.0f), glm::vec3(0.0f, 0.0f, -70.0f)); //proporciones de todo con un plano
model = glm::scale(model, glm::vec3(12.0f));
staticShader.setMat4("model", model);
plano.Draw(staticShader);
```

## Objetos dinámicos.

Para estos objetos se importan de la misma manera que los objetos estáticos solo que ahora estos utilizan **archivos .dae** y sus respectivas texturas, ya que se utiliza un **Animate Shader**.

Para declarar un objeto dinámico se realiza de la siguiente manera:

```
// Dog
ModelAnim dog("resources/objects/Dog/doggo.dae");
dog.initShaders(animShader.ID);
```

Posteriormente solo se llama al objeto y de igual manera se le realiza sus transformaciones correspondientes, así como también utilizar el **Animate Shader**.

```
// -----
// Animaciones (AnimShader)
// -----
//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", 0.0f, 0.0f, 0.0f);
animShader.setVec3("light.direction", lightDirection);
animShader.setVec3("viewPos", camera.Position);

//-----Bicicleta-----
model = glm::translate(glm::mat4(1.0f), glm::vec3(movBici_x, 0.0f, movBici_z));
model = glm::rotate(model, glm::radians(orientaBici), glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::scale(model, glm::vec3(0.09));
animShader.setMat4("model", model);
manBici.Draw(animShader);
```

Para que este objeto realice una animación en el entorno, se requiere realizar las modificaciones en la función **Animate**, como las que se verán a continuación.

## Información de las animaciones.

### Animación de hombre en bici:

Para generar la animación se crearon tres variables globales:

```
float movBici_z = -5.0f, // posición inicial en z  
      movBici_x = -500.0f, // posición inicial en x  
      orientaBici = 0.0f; // orientación incial
```

La animación está dividida en varios objetos:

- Hombre pedaleando
- Cuadro de la bicicleta
- Pedales
- Ruedas

La animación del hombre pedaleando fue creado con 3ds Max utilizando la herramienta de Auto key. al momento de dibujar al hombre en el programa le asignamos las variables de movimiento en x y z de la bicicleta y de rotación para que al momento que la bicicleta se mueva, el hombre también lo haga.

```
//-----Bicicleta-----  
model = glm::translate(glm::mat4(1.0f), glm::vec3(movBici_x, 0.0f, movBici_z));  
model = glm::rotate(model, glm::radians(orientaBici), glm::vec3(0.0f, 1.0f, 0.0f));  
model = glm::scale(model, glm::vec3(0.09));  
animShader.setMat4("model", model);  
manBici.Draw(animShader);
```

Dibujamos la bicicleta con sus valores iniciales y creamos un modelo temporal. Para la animación de los pedales y las ruedas agregamos una rotación del eje x:

```

// -----
// Persona en bici
// -----
tmp = model = glm::translate(glm::mat4(1.0f), glm::vec3(movBici_x, -0.5f, movBici_z));
tmp = model = glm::rotate(model, glm::radians(orientaBici), glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::scale(model, glm::vec3(0.09f));
staticShader.setMat4("model", model);
cuadro.Draw(staticShader);

model = glm::translate(tmp, glm::vec3(0.0f, 2.75f, 0.5f));
model = model = glm::rotate(model, glm::radians(giroPedales), glm::vec3(1.0f, 0.0f, 0.0f));
model = glm::scale(model, glm::vec3(0.09f));
staticShader.setMat4("model", model);
pedales.Draw(staticShader);

model = glm::translate(tmp, glm::vec3(-0.2f, 2.85f, 5.85f));
model = model = glm::rotate(model, glm::radians(giroPedales), glm::vec3(1.0f, 0.0f, 0.0f));
model = glm::scale(model, glm::vec3(0.09f));
staticShader.setMat4("model", model);
rueda.Draw(staticShader); // Adelante

model = glm::translate(tmp, glm::vec3(-0.2f, 2.85f, -3.25f));
model = model = glm::rotate(model, glm::radians(giroPedales), glm::vec3(1.0f, 0.0f, 0.0f));
model = glm::scale(model, glm::vec3(0.09f));
staticShader.setMat4("model", model);
staticShader.setMat4("model", model);
rueda.Draw(staticShader); // Atras

```

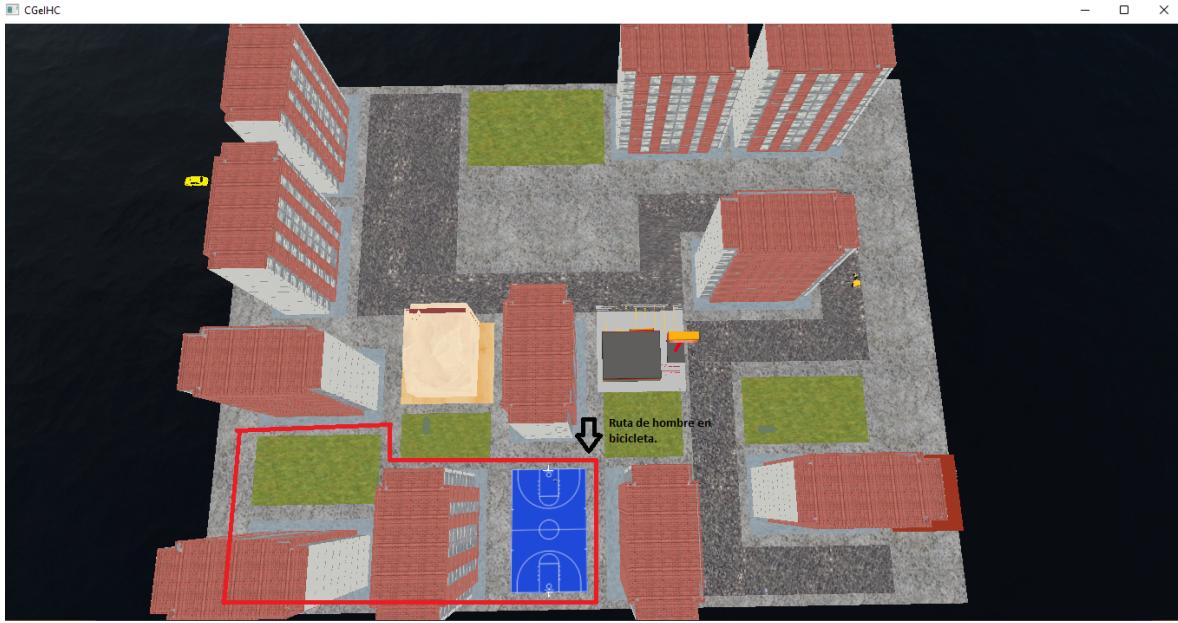
Para la animación de la bici se asignó una ruta dentro del mapa en donde se utilizaran 6 estados del 0 al 5 utilizando la estructura de control Switch/Case, la bicicleta va a cambiar su posición en el eje x y el eje z.

Esta animación se repite infinitamente ya que al llegar al estado 5 la bicicleta se encuentra en las coordenadas de origen y regresa al estado 0 tampoco se necesita presionar alguna tecla para que comience, la animación inicia junto con el programa.

```

switch (estado_bici)
{
    case 0:
        if (movBici_z <= 255.0f) {
            movBici_z += 2.0f;
            orientaBici = 0.0;
        }
        else {
            estado_bici = 1;
        }
        break;
    case 1:
        if (movBici_x <= 55.0f) {
            orientaBici = 90.0f;
            movBici_x += 2.0;
        }
        else {
            estado_bici = 2;
        }
        break;
    case 2:
        if (movBici_z >= 50.0f)
        {
            orientaBici = 180.0f;
            movBici_z -= 2.0f;
        }
        else {
            estado_bici = 3;
        }
        break;
    case 3:
        if (movBici_x >= -250.0f) {
            orientaBici = 270.0f;
            movBici_x -= 2.0f;
        }
        else {
            estado_bici = 4;
        }
        break;
    case 4:
        if (movBici_z >= -5.0f) {
            orientaBici = 180.0f;
            movBici_z -= 2.0f;
        }
        else {
            estado_bici = 5;
        }
        break;
    case 5:
        if (movBici_x >= -500.0f) {
            orientaBici = 270.0f;
            movBici_x -= 2.0f;
        }
        else
        {
            estado_bici = 0;
        }
        break;
    default:
        break;
}

```



### Animación de hombre haciendo deporte en la cancha:

Para generar la animación se crearon cuatro variables globales:

```
//para mov de deportista
float movShan = 0.0f, //con esto se moverá el deportista
    IdaRegresoShan = 1.0f; //para que avance y regrese
int estadoShan = 1.0f;
float orientShan = 0.0f; // para que gire a diferentes posiciones
```

La animación está formada por un único objeto, el personaje Shannon con la animación Running, descargada de la página Miamamo. Al momento de descargar se indicó que la animación debía ser en un solo lugar, para que a base de código se generará el desplazamiento.

Para poder dibujar el objeto se debió colocar la variable que le permitirá desplazarse en el eje de las Z, esto tomando ventaja de que no se moverá en los ejes X y Y; en conjunto con una variable que permita su rotación sobre el eje Y para que simule su cambio de dirección.

```

//Deportista
model = glm::translate(glm::mat4(1.0f), glm::vec3(0.0f, 0.0f, 70.0f + movShan));
model = glm::rotate(model, glm::radians(orientShan), glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::scale(model, glm::vec3(0.09f));//escala
animShader.setMat4("model", model);
shannon.Draw(animShader);

```

Ocupando un SWITCH se escoge entre los diferentes estados, que representan hasta qué distancia de la cancha debe llegar el objeto, dentro de cada CASE se encuentra una sentencia IF que verifica si el objeto se encuentra avanzando (hacia valores mayores) o regresando (hacia valores menores).

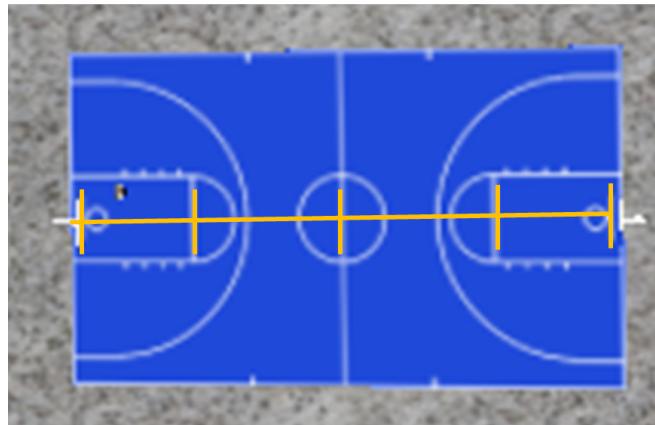
En caso de que se cumpla la condición de que llegue hasta cierta distancia, el objeto realizará un giro de 180° y realizará el movimiento siguiente cambiando de estado.

```

switch (estadoShan){//deportista: va a hacer Suicidios
case 1://llega a 20
    if (IdaRegresoShan == 1)
    {
        movShan += 0.5f;//con esto el deportista sale hacia adelante
        if (movShan >= 20) {
            IdaRegresoShan = 0;//cambio de estado
            orientShan = 180.0f;//giro para el regreso
        }
    }
    else {
        movShan -= 0.5f;//con esto el deportista sale hacia atras
        if (movShan <= 0) {
            orientShan = 0.0f;//giro hacia enfrente
            IdaRegresoShan = 1;//cambio de estado
            estadoShan = 2;//cambio de estado del case
        }
    }
    break;
case 2://llega a 80
    if (IdaRegresoShan == 1 && estadoShan == 2)
    {
        movShan += 0.5f;//con esto el deportista sale hacia adelante
        if (movShan >= 80) {
            IdaRegresoShan = 0;//cambio de estado
            orientShan = 180.0f;//giro para el regreso
        }
    }
    else {
        movShan -= 0.5f;//con esto el deportista sale hacia atras
        if (movShan <= 0) {
            orientShan = 0.0f;//giro hacia enfrente
            IdaRegresoShan = 1;//cambio de estado
            estadoShan = 3;//cambio de estado del case
        }
    }
}
case 3://llega a 120
if (IdaRegresoShan == 1)
{
    movShan += 0.5f;//con esto el deportista sale hacia adelante
    if (movShan >= 120) {
        IdaRegresoShan = 0;//cambio de estado
        orientShan = 180.0f;//giro para el regreso
    }
}
else {
    movShan -= 0.5f;//con esto el deportista sale hacia atras
    if (movShan <= 0) {
        orientShan = 0.0f;//giro hacia enfrente
        IdaRegresoShan = 1;//cambio de estado
        estadoShan = 4;//cambio de estado del case
    }
}
break;
case 4://llega a 160
if (IdaRegresoShan == 1)
{
    movShan += 0.5f;//con esto el deportista sale hacia adelante
    if (movShan >= 160) {
        IdaRegresoShan = 0;//cambio de estado
        orientShan = 180.0f;//giro para el regreso
    }
}
else {
    movShan -= 0.5f;//con esto el deportista sale hacia atras
    if (movShan <= 0) {
        orientShan = 0.0f;//giro hacia enfrente
        IdaRegresoShan = 1;//cambio de estado
        estadoShan = 1;//cambio de estado del case (regreso)
    }
}
}

```

Para que resulte la animación de que el personaje corra hasta cada marca y regrese hasta el origen una y otra vez, ya que no se condicionó la animación a una entrada del usuario al presionar una tecla.



### **Animación de Carro:**

Para generar la animación se crearon cuatro variables globales:

```
//para el movimiento del automovil
float    movAuto_x = 0.0f,
          movAuto_z = 0.0f,
          orienta = 180.0f,
          girollantas = 0.0f,
          estadoAuto = 1.0f;
```

La animación está formada por cinco objetos, la carrocería y cuatro ruedas, considero que lo más complicado de colocar los objetos fue las proporciones y colocar las ruedas en la posición correcta para que no se vean más grandes de lo que deberían.

En la carrocería se colocaron las variables que permiten su movimiento en X y Z y una para realizar el giro sobre Y; para las ruedas solo se ocupa una variable para su giro sobre X.

```

// -
// 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(-225.0f + movAuto_x, 1.5f, 500.0f + movAuto_z));
tmp = model = glm::rotate(model, glm::radians(orienta), glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::scale(model, glm::vec3(0.005f, 0.005f, 0.005f));
staticShader.setMat4("model", model);
Carro.Draw(staticShader);

model = glm::translate(tmp, glm::vec3(6.7f, 1.0f, 15.0f));
model = glm::rotate(model, glm::radians(girollantas), glm::vec3(1.0f, 0.0f, 0.0f)); //giro de las llantas
model = glm::scale(model, glm::vec3(0.0044f, 0.0044f, 0.0044f));
model = glm::rotate(model, glm::radians(180.0f), glm::vec3(0.0f, 1.0f, 0.0f));
staticShader.setMat4("model", model);
llanta.Draw(staticShader); //Izq delantera

model = glm::translate(tmp, glm::vec3(-6.7f, 1.0f, 15.0f));
model = glm::rotate(model, glm::radians(girollantas), glm::vec3(1.0f, 0.0f, 0.0f));
model = glm::scale(model, glm::vec3(0.0044f, 0.0044f, 0.0044f));
staticShader.setMat4("model", model);
llanta.Draw(staticShader); //Der delantera

model = glm::translate(tmp, glm::vec3(-6.7f, 1.0f, -10.0f));
model = glm::rotate(model, glm::radians(girollantas), glm::vec3(1.0f, 0.0f, 0.0f));
model = glm::scale(model, glm::vec3(0.0044f, 0.0044f, 0.0044f));
model = glm::rotate(model, glm::radians(180.0f), glm::vec3(0.0f, 1.0f, 0.0f));
staticShader.setMat4("model", model);
llanta.Draw(staticShader); //Izq trasera

model = glm::translate(tmp, glm::vec3(6.7f, 1.0f, -10.0f));
model = glm::rotate(model, glm::radians(girollantas), glm::vec3(1.0f, 0.0f, 0.0f));
model = glm::scale(model, glm::vec3(0.0044f, 0.0044f, 0.0044f));
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

```

Para que el auto realice sus movimientos se necesita que el usuario presione la tecla **SPACE** para que avance o se detenga. Para el recorrido se decidió que empiece al lado del objeto de entrada y termine estacionado.

Lo primero que se debe hacer es verificar en qué estado se encuentra el auto para que avance y giren las llantas hacia una dirección, después se pregunta si ya llegó hasta una posición que seleccionamos para que realice un giro, cambie de estado y siga avanzando; las posiciones dependen de la distribución del asfalto.

Considero que eso fue lo más complicado de la animación fue ver hasta qué punto debe girar y no atraviesa objetos que no deseamos.

```

//Vehículo (se estaciona)
if (animacion) {
    if (estadoAuto == 1) {
        movAuto_z -= 3.0f;
        girollantas += 3.0f;
        if (movAuto_z <= -260) {
            orienta = 90.0f;
            estadoAuto = 2.0f;
            //animacion = FALSE;//no debe pararse
        }
    }
    if (estadoAuto == 2) {
        movAuto_x += 3.0f;
        girollantas += 3.0f;
        if (movAuto_x >= 360) {
            orienta = 0.0f;
            estadoAuto = 3.0f;
            //animacion = FALSE;//no debe pararse
        }
    }
    if (estadoAuto == 3) {
        movAuto_z += 3.0f;
        girollantas += 3.0f;
        if (movAuto_z >= -30) {
            orienta = 90.0f;
            estadoAuto = 4.0f;
            //animacion = FALSE;//no debe pararse
        }
    }
}
if (girollantas == 0) {
    if (movAuto_x >= 580) {
        orienta = 180.0f;
        estadoAuto = 5.0f;
        //animacion = FALSE;//no debe pararse
    }
}
if (girollantas == 3.0f) {
    if (movAuto_x >= 580) {
        orienta = 180.0f;
        estadoAuto = 5.0f;
        //animacion = FALSE;//no debe pararse
    }
}
if (girollantas == 6.0f) {
    if (movAuto_x >= 580) {
        orienta = 180.0f;
        estadoAuto = 5.0f;
        //animacion = FALSE;//no debe pararse
    }
}
if (girollantas == 9.0f) {
    if (movAuto_x >= 650) {
        orienta = 180.0f;
        estadoAuto = 9.0f;
        //animacion = FALSE;//no debe pararse
    }
}
if (girollantas == 12.0f) {
    if (movAuto_x >= 650) {
        orienta = 180.0f;
        estadoAuto = 9.0f;
        //animacion = FALSE;//no debe pararse
    }
}
if (girollantas == 15.0f) {
    if (movAuto_x >= 650) {
        orienta = 180.0f;
        estadoAuto = 9.0f;
        //animacion = FALSE;//no debe pararse
    }
}
if (girollantas == 18.0f) {
    if (movAuto_x >= 650) {
        orienta = 180.0f;
        estadoAuto = 9.0f;
        //animacion = FALSE;//no debe pararse
    }
}
if (girollantas == 21.0f) {
    if (movAuto_x >= 650) {
        orienta = 180.0f;
        estadoAuto = 9.0f;
        //animacion = FALSE;//no debe pararse
    }
}
if (girollantas == 24.0f) {
    if (movAuto_x >= 650) {
        orienta = 180.0f;
        estadoAuto = 9.0f;
        //animacion = FALSE;//no debe pararse
    }
}
if (girollantas == 27.0f) {
    if (movAuto_x >= 650) {
        orienta = 180.0f;
        estadoAuto = 9.0f;
        //animacion = FALSE;//no debe pararse
    }
}
if (girollantas == 30.0f) {
    if (movAuto_x >= 650) {
        orienta = 180.0f;
        estadoAuto = 9.0f;
        //animacion = FALSE;//no debe pararse
    }
}
if (girollantas == 33.0f) {
    if (movAuto_x >= 650) {
        orienta = 180.0f;
        estadoAuto = 9.0f;
        //animacion = FALSE;//no debe pararse
    }
}
if (girollantas == 36.0f) {
    if (movAuto_x >= 650) {
        orienta = 180.0f;
        estadoAuto = 9.0f;
        //animacion = FALSE;//no debe pararse
    }
}
if (girollantas == 39.0f) {
    if (movAuto_x >= 650) {
        orienta = 180.0f;
        estadoAuto = 9.0f;
        //animacion = FALSE;//no debe pararse
    }
}
if (girollantas == 42.0f) {
    if (movAuto_x >= 650) {
        orienta = 180.0f;
        estadoAuto = 9.0f;
        //animacion = FALSE;//no debe pararse
    }
}
if (girollantas == 45.0f) {
    if (movAuto_x >= 650) {
        orienta = 180.0f;
        estadoAuto = 9.0f;
        //animacion = FALSE;//no debe pararse
    }
}
if (girollantas == 48.0f) {
    if (movAuto_x >= 650) {
        orienta = 180.0f;
        estadoAuto = 9.0f;
        //animacion = FALSE;//no debe pararse
    }
}
if (girollantas == 51.0f) {
    if (movAuto_x >= 650) {
        orienta = 180.0f;
        estadoAuto = 9.0f;
        //animacion = FALSE;//no debe pararse
    }
}
if (girollantas == 54.0f) {
    if (movAuto_x >= 650) {
        orienta = 180.0f;
        estadoAuto = 9.0f;
        //animacion = FALSE;//no debe pararse
    }
}
if (girollantas == 57.0f) {
    if (movAuto_x >= 650) {
        orienta = 180.0f;
        estadoAuto = 9.0f;
        //animacion = FALSE;//no debe pararse
    }
}
if (girollantas == 60.0f) {
    if (movAuto_x >= 650) {
        orienta = 180.0f;
        estadoAuto = 9.0f;
        //animacion = FALSE;//no debe pararse
    }
}
if (girollantas == 63.0f) {
    if (movAuto_x >= 650) {
        orienta = 180.0f;
        estadoAuto = 9.0f;
        //animacion = FALSE;//no debe pararse
    }
}
if (girollantas == 66.0f) {
    if (movAuto_x >= 650) {
        orienta = 180.0f;
        estadoAuto = 9.0f;
        //animacion = FALSE;//no debe pararse
    }
}
if (girollantas == 69.0f) {
    if (movAuto_x >= 650) {
        orienta = 180.0f;
        estadoAuto = 9.0f;
        //animacion = FALSE;//no debe pararse
    }
}
if (girollantas == 72.0f) {
    if (movAuto_x >= 650) {
        orienta = 180.0f;
        estadoAuto = 9.0f;
        //animacion = FALSE;//no debe pararse
    }
}
if (girollantas == 75.0f) {
    if (movAuto_x >= 650) {
        orienta = 180.0f;
        estadoAuto = 9.0f;
        //animacion = FALSE;//no debe pararse
    }
}
if (girollantas == 78.0f) {
    if (movAuto_x >= 650) {
        orienta = 180.0f;
        estadoAuto = 9.0f;
        //animacion = FALSE;//no debe pararse
    }
}
if (girollantas == 81.0f) {
    if (movAuto_x >= 650) {
        orienta = 180.0f;
        estadoAuto = 9.0f;
        //animacion = FALSE;//no debe pararse
    }
}
if (girollantas == 84.0f) {
    if (movAuto_x >= 650) {
        orienta = 180.0f;
        estadoAuto = 9.0f;
        //animacion = FALSE;//no debe pararse
    }
}
if (girollantas == 87.0f) {
    if (movAuto_x >= 650) {
        orienta = 180.0f;
        estadoAuto = 9.0f;
        //animacion = FALSE;//no debe pararse
    }
}
if (girollantas == 90.0f) {
    if (movAuto_x >= 650) {
        orienta = 180.0f;
        estadoAuto = 9.0f;
        //animacion = FALSE;//no debe pararse
    }
}
}

```

## Animación Hombre que camina:

Para generar la animación se crearon cuatro variables globales:

```

//para el movimiento del constructor
float movConstX = 0.0f,
      movConstZ = 0.0f,
      OrientaConst = 180.0f,
      estadoConst = 1.0f;

```

Para esta animación solo se ocupó un objeto descargado de Mixamo. El personaje se colocó debajo de la entrada y con dos variables para su movimiento en X y Z se desplazará hasta llegar al OXXO y con una variable en su rotación girara sobre el eje Y.

```

// Persona Caminando
model = glm::translate(glm::mat4(1.0f), glm::vec3(580.0f + movConstX, 0.0f, 120.0f + movConstZ));
model = glm::scale(model, glm::vec3(0.09f));
model = glm::rotate(model, glm::radians(OrientaConst), glm::vec3(0.0f, 1.0f, 0.0f));
animShader.setMat4("model", model);
manWalk.Draw(animShader);

```

Dentro de la función `animate`, se colocó un IF para que el usuario presione la tecla 1 y así el personaje empiece su recorrido. Posteriormente se verifica en qué estado se encuentra para que avance en el sentido correcto y al llegar hasta cierta ubicación, éste cambia de orientación y de estado.

Cabe aclarar que en el estado cinco, el personaje no avanzará si es que el automóvil va a cruzar por el mismo sitio, esto verificando que el estado del auto sea diferente a dos; a su vez el personaje solo atraviesa el objeto OXXO, no se abren las puertas para que entre.

```

if (animacionConst) { //presionar 1 para que el constructor camine
    if (estadoConst == 1) {
        movConstX -= 0.6f;
        if (movConstX <= -90) {
            OrientaConst = 90.0f;
            estadoConst = 2.0f;
            //animacionConst = FALSE; //no debe pararse
        }
    }
    if (estadoConst == 2) {
        movConstZ -= 0.6f;
        if (movConstZ <= -85) {
            OrientaConst = 180.0f;
            estadoConst = 3.0f;
            //animacionConst = FALSE; //no debe pararse
        }
    }
    if (estadoConst == 3) {
        movConstX -= 0.6f;
        if (movConstX <= -290) {
            OrientaConst = 90.0f;
            estadoConst = 4.0f;
            //animacionConst = FALSE; //no debe pararse
        }
    }
    if (estadoConst == 4) {
        movConstZ -= 0.6f;
        if (movConstZ <= -200) {
            OrientaConst = 180.0f;
            estadoConst = 5.0f;
            //animacionConst = FALSE; //no debe pararse
        }
    }
}

if (estadoConst == 5 && estadoAuto != 2) { //solo cruza si es que el carro no esta
    movConstX -= 0.6f;
    if (movConstX <= -370) {
        OrientaConst = 90.0f;
        estadoConst = 6.0f;
        //animacionConst = FALSE; //no debe pararse
    }
}
if (estadoConst == 6) {
    movConstZ -= 0.6f;
    if (movConstZ <= -275) {
        OrientaConst = 180.0f;
        estadoConst = 7.0f;
        //animacionConst = FALSE; //no debe pararse
    }
}
if (estadoConst == 7) {
    movConstX -= 0.6f;
    if (movConstX <= -440) {
        OrientaConst = -90.0f;
        estadoConst = 8.0f;
        //animacionConst = FALSE; //no debe pararse
    }
}
if (estadoConst == 8) {
    movConstZ += 0.6f;
    if (movConstZ >= -260) {
        animacionConst = FALSE;
    }
}

```

### Animación Persona paseando perro.

Para realizar la animación correspondiente, primero se definió una constante, la cuál indicará la velocidad en la que los personajes además de 5 variables flotantes, donde estas nos ayudaran a trasladar los objetos en el eje X, Z y finalmente rotar estos mismos.

```

1 #define SPEED_MOV 0.2f
2 float    rotDogPerson = 180.0f,
3         movPersonX = 0.0f,
4         movPersonZ = 0.0f,
5         movDogX = 0.0f,
6         movDogZ = 0.0f;

```

Otra variable que también se declara, es la del estado de la animación, la cuál será la que definirá el cambio de estado por medio de un switch-case.

```

int estado_trici = 0,
estado_dogPerson = 0,
estado_bici = 0;

```

Para que los objetos puedan realizar la animación, es importante que donde dibujamos estos mismos, agreguemos las variables correspondientes en la traslación (X, Z), donde estas se sumarán con su posición definida, de manera que la posición inicial no será modificada. De igual manera, para que los objetos puedan realizar una rotación, se agrega la variable correspondiente en esta.

```

// DOG
model = glm::translate(glm::mat4(1.0f), glm::vec3(290.0f + movDogX, 0.0f, 0.0f + movDogZ));
model = glm::scale(model, glm::vec3(0.3f));
model = glm::rotate(model, glm::radians(rotDogPerson), glm::vec3(0.0f, 1.0f, 0.0f));
animShader.setMat4("model", model);
dog.Draw(animShader);

// Persona Paseando
model = glm::translate(glm::mat4(1.0f), glm::vec3(285.0 + movPersonX, 0.0f, 0.0f + movPersonZ));
model = glm::scale(model, glm::vec3(0.09f));
model = glm::rotate(model, glm::radians(rotDogPerson), glm::vec3(0.0f, 1.0f, 0.0f));
animShader.setMat4("model", model);
womanWalk.Draw(animShader);

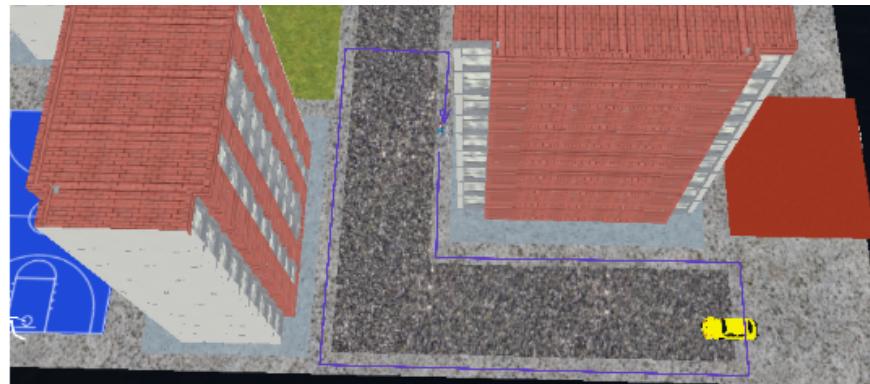
```

Para realizar los estados de la animación se utiliza la estructura de control Switch-Case, de manera que cuando el resultado esperado se cumpla, pase a otro caso y continúe la animación.

Para cada caso se encuentra un if-else el cuál va comprobar que el movimiento en determinado eje no supere al que determinó el desarrollador, lo que consecuencia los objetos aumentarán o disminuirán su posición hasta este límite además de rotar los grados establecidos. Finalmente una vez la condición no se cumpla, el perro aumentará o disminuirá su posición de manera que este logre que el perro esté al lado izquierdo de la persona y se pase al siguiente caso (estado de la animación con un máximo de 5 estados).

```

1  switch (estado_dogPerson) {
2      case 0:
3          if (movPersonZ <= 170.0f) {
4              rotDogPerson = 0.0f;
5              movDogZ += SPEED_MOV;
6              movPersonZ += SPEED_MOV;
7          } else {
8              movDogZ = movPersonZ - 5.0f;
9              estado_dogPerson = 1;
10         }
11         break;
12     case 1:
13         if (movPersonX <= 220.0f) {
14             rotDogPerson = 90.0f;
15             movDogX += SPEED_MOV;
16             movPersonX += SPEED_MOV;
17         } else {
18             movDogX = movPersonX + 5.0f;
19             estado_dogPerson = 2;
20         }
21         break;
22     case 2:
23         if (movPersonZ <= 250.0f) {
24             rotDogPerson = 0.0f;
25             movDogZ += SPEED_MOV;
26             movPersonZ += SPEED_MOV;
27         }
28         else {
29             movDogZ = movPersonZ + 10.0f;
30             estado_dogPerson = 3;
31         }
32         break;
33     case 3:
34         if (movPersonX >= -80.0f) {
35             rotDogPerson = 270.0f;
36             movDogX -= SPEED_MOV;
37             movPersonX -= SPEED_MOV;
38         }
39         else {
40             movDogX = movPersonX - 10.0f;
41             estado_dogPerson = 4;
42         }
43         break;
44     case 4:
45         if (movPersonX >= 0.0f) {
46             rotDogPerson = 180.0f;
47             movDogZ -= SPEED_MOV;
48             movPersonZ -= SPEED_MOV;
49         }
50         else {
51             movDogZ = movPersonZ - 5.0f;
52             estado_dogPerson = 5;
53         }
54         break;
55     case 5:
56         if (movPersonX <= 5.0f) {
57             rotDogPerson = 90.0f;
58             movDogX += SPEED_MOV;
59             movPersonX += SPEED_MOV;
60         }
61         else {
62             movDogZ = movPersonX + 5.0f;
63             estado_dogPerson = 0;
64             movPersonZ = movDogZ = 0.0f;
65             movPersonX = movDogX = 0.0f;
66         }
67         break;
68     }
69 }
```



## **Referencias.**

- Edificios de México (s.f.). Villa Olímpica Miguel Hidalgo [mensaje en un blog]. Recuperado de <https://www.edemx.com/site/villa-olimpica-miguel-hidalgo/>
- MIXAMO. (2022). Recuperado de <https://www.mixamo.com/#/>
- Open3DModel. (2022). Recuperado de <https://open3dmodel.com/es/>