Escuela Técnica superior de ingeniería

Universidad de Huelva

Trabajo

Entrenamiento de Rocket League

Asignatura

Realidad Virtual

Alumno

Jaime José Romero Molina

Curso

2023-2024

Índice

1. Introducción

Esta es la memoria del trabajo final de la asignatura de Realidad virtual. Este trabajo consiste en desarrollar "a mano" un juego inspirado en el popular videojuego Rocket League. Se trata de desarrollar un modo de juego de un único jugador en el que el objetivo es marcar gol con un único coche a modo de entrenamiento del juego original.

Rocket League es un videojuego que combina el fútbol con los vehículos. Fue desarrollado por Psyonix, dicho juego se lanzó por primera vez para PlayStation 4 y Windows en julio de 2015, y más tarde se lanzaron ports para Xbox One y Nintendo Switch.

En junio de 2016, 505 Games comenzó a distribuir una versión física minorista para PlayStation 4 y Xbox One, y Warner Bros. Interactive Entertainment se hizo cargo de esas funciones a finales de 2017. El juego pasó a ser free to play en septiembre de 2020, cuando Epic Games tomó posesión.

2. Controles

En este trabajo, tenemos 3 tipos de modo de cámara, el modo 1 en el que la vista está situada en el coche, el modo 2 que será una vista desde arriba para poder ver todo el campo de juego y el modo 3 será para desplazar la cámara libremente a la vez que puedes mover el coche. A continuación, se muestran los controles del juego:

2.1 Modo Cámara 1

Flecha Arriba	Aumenta velocidad del coche
Flecha Abajo	Disminuye velocidad del coche
Flecha Izquierda	Gira el coche hacia la izquierda
Flecha Derecha	Gira el coche hacia la derecha
Tecla S	Para el coche
Numero 2	Cambia a la cámara 2
Numero 3	Cambia a la cámara 3

2.2 Modo Cámara 2

Flecha Arriba	Aumenta velocidad del coche
Flecha Abajo	Disminuye velocidad del coche
Flecha Izquierda	Gira el coche hacia la izquierda
Flecha Derecha	Gira el coche hacia la derecha
Tecla S	Para el coche
Numero 1	Cambia a la cámara 1
Numero 3	Cambia a la cámara 3

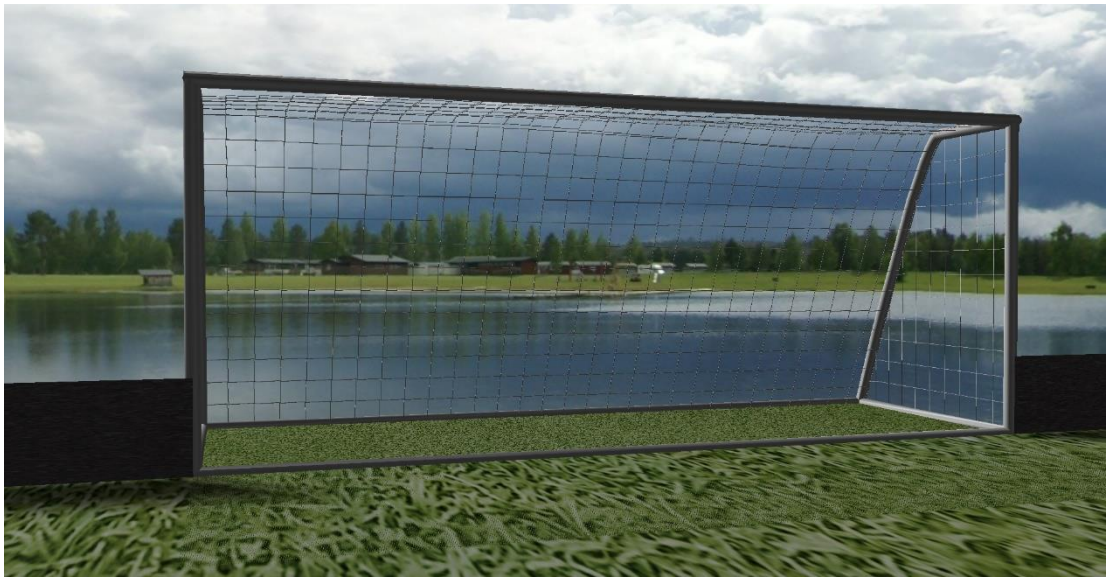
2.3 Modo Cámara 3

Flecha Arriba	Gira hacia abajo
Flecha Abajo	Gira hacia arriba
Flecha Izquierda	Gira en sentido contrario a un reloj
Flecha Derecha	Gira en sentido del reloj
Tecla S	Para la cámara
Numero 1	Cambia a la cámara 1
Numero 2	Cambia a la cámara 2
Tecla +	Aumenta la velocidad de la cámara
Tecla Delete	Reduce la velocidad de la cámara
Tecla Q	Mueve la cámara hacia arriba
Tecla A	Mueve la cámara hacia abajo
Tecla O	Mueve la cámara hacia la izquierda
Tecla P	Mueve la cámara hacia la derecha
Tecla K	Gira la cámara hacia la izquierda
Tecla L	Gira la cámara hacia la derecha
Tecla R	Aumenta la velocidad del coche
Tecla F	Disminuye la velocidad del coche
Tecla D	Gira el coche hacia la izquierda
Tecla G	Gira el coche hacia la derecha

3. Descripción de Objetos

3.1 Porterías

Las porterías se han implementado creando una nueva clase llamada footballgoal, a partir de un modelo descargado de la página free3D.com y a la que se le ha aplicado una textura de color blanco.



3.2 Paredes

Las paredes que delimitan el terreno de juego se han creado usando como base la clase CGGround proporcionada en las prácticas. Hemos creado 6 “Suelos” que hemos desplazado y rotado de forma que se sitúen en las posiciones límites del campo y se les ha aplicado la textura “concrete.jpg” proporcionada en los archivos auxiliares.

4. Clases que desarrollan la escena

4.1 CGScene

La clase CGScene es la que se encarga de crear los objetos que componen nuestro terreno de juego.

En ella se inicializan, el suelo y las paredes usando la clase CGGround, se inicializa el coche con la clase Mercedes_G_Class, las porterías con football_goal y la luz con CGLight.

4.2 CGSkybox

La clase CGSkybox se encarga de generar el skybox que rodea la escena.

4.3 CGLight

La clase CGLight se encarga de representar la luz del juego.

4.4 CGModel

La clase CGModel es la clase principal y encargada de inicializar la escena, la cámara y el skybox, en ella se encuentra la mayor parte de la lógica del proyecto, como la forma en la que se calcula el movimiento del balón, el movimiento del coche, los impactos del coche al balón, los choques con las paredes y el gol.

5. Posición de la cámara

5.1 Cámara 1

Colocamos la cámara en la misma posición que el coche (salvo en la altura que la aumentamos 1.5f) y la orientamos para que mire hacia donde apunta la parte delantera del coche, finalmente la hacemos que apunte un poco hacia abajo para que la vista quede mejor, esta cámara va actualizando su posición constantemente por lo que es la que se encarga de seguir al coche.

5.2 Cámara 2

Posicionamos la cámara en el centro, salvo el eje Y que tendrá el valor de 220, y la hacemos mirar hacia abajo para poder ver el terreno de juego desde arriba.

5.3 Cámara 3

Es igual a la implementada en las prácticas, es una cámara libre, a excepción de que también se puede controlar el coche en este modo.

6. Descripción de movimiento y choque

6.1 Balón

El movimiento del balón corresponde al de un objeto que sufre la aceleración de la gravedad (9,8 m/s2). Esto genera una trayectoria parabólica. Como podemos ver en la siguiente imagen, si hay choque, asignamos a la pelota la dirección y velocidad del coche antes del impacto.

```
if (distance < cocheRadius + ballRadius) {

    GLfloat moveStep = mercedes->GetMoveStep();
    vy = 9.8/60.f;
    locball[2][0] = loc[2][0];
    locball[2][1] = loc[2][1];
    locball[2][2] = loc[2][2];
    locball[0][0] = loc[0][0];
    locball[0][1] = loc[0][1];
    locball[0][2] = loc[0][2];

    bola->location = locball;

    //Velocidad X
        bola->SetMoveStep(moveStep);
    //Velocidad Y
        bola->Move(vy*0.5f);
```

Y la velocidad tanto en el eje X como en el eje Y se van actualizando en el método update y en cameraconstraints.

Eje Y en método Update:

```
bola->Move(vy);

CameraConstraints();

//Actualizamos velocidad eje Y
if (vy != 0)
   vy = vy - (0.098f/60.0f);
```

Eje X en cameraconstraints: Cada rebote disminuye un 20% la velocidad.

```
if (ballpos.y < 1.0f) { ballpos.y = 1.0f; balllimit = 1; vy = vy * -0.6; bola->SetMoveStep(moveStep * 0.8f);    }
if (ballpos.y > 100.0f) {  balllimit = 1; vy = vy * -0.6;   bola->SetMoveStep(moveStep*0.8f);    }
if (ballpos.x > 99.0f && (ballpos.z > 10.0f || ballpos.z < -10.0f || ballpos.y > 6.5f)) { ballpos.x = 99.0f;
ballDir = glm::reflect(-ballDir, glm::vec3(-1.0f, 0.0f, 0.0f)); balllimit = 2; bola->SetMoveStep(moveStep * 0.8f); }
if (ballpos.x < -99.0f && (ballpos.z > 10.0f || ballpos.z < -10.0f || ballpos.y > 6.5f)) { ballpos.x = -99.0f;
ballDir= glm::reflect(-ballDir, glm::vec3(1.0f, 0.0f, 0.0f)); balllimit = 2; bola->SetMoveStep(moveStep * 0.8f); }
if (ballpos.z > 49.0f) { ballpos.z = 49.0f;
ballDir = glm::reflect(-ballDir, glm::vec3(0.0f, 0.0f, 1.0f)); balllimit = 3;  bola->SetMoveStep(moveStep * 0.8f); }
if (ballpos.z < -49.0f) { ballpos.z = -49.0f;
ballDir = glm::reflect(-ballDir, glm::vec3(0.0f, 0.0f, -1.0f)); balllimit = 3; bola->SetMoveStep(moveStep * 0.8f);}
if (balllimit == 1)
{
   locball[3][0] = ballpos.x;
   locball[3][1] = ballpos.y;
   locball[3][2] = ballpos.z;
   scene->fig4->location = locball;
}
if (balllimit == 2)
{
   locball[3][0] = ballpos.x;
   locball[3][1] = ballpos.y;
   locball[3][2] = ballpos.z;
   locball[2][0] = ballDir.x;
   locball[2][1] = ballDir.y;
```

6.2 Coche

El movimiento del coche sigue una lógica similar a la clase CGCamera implementada en las prácticas. A la clase Mercedes_G_Class se le ha añadido:

```
GLfloat moveStep;
void SetMoveStep(GLfloat step);
void SetTurnStep(GLfloat step);
GLfloat GetMoveStep();
GLfloat GetTurnStep();
void MoveFront();
void MoveBack();
void TurnRight();
void TurnLeft();
void TurnCW();
void TurnCCW();
```

```
GLfloat turnStep;
GLfloat cosAngle;
GLfloat sinAngle;
```

Por lo que con las teclas podemos aumentar y reducir su velocidad cambiando la variable moveStep al igual que con CGCamera, y girar el coche con las flechas del teclado, gracias a la siguiente implementación de métodos

```
void Mercedes_G_Class::MoveFront() {
    Translate(glm::vec3(-moveStep, 0.0f, 0.0f));
}

void Mercedes_G_Class::MoveBack() {
    Translate(glm::vec3(moveStep, 0.0f, 0.0f));
}

void Mercedes_G_Class::TurnRight() {
    Rotate(-turnStep, glm::vec3(0.0f, 1.0f, 0.0f));
}

void Mercedes_G_Class::TurnLeft() {
    Rotate(turnStep, glm::vec3(0.0f, 1.0f, 0.0f));
}
```

6.3 Impacto coche con balón

El impacto del coche con el balón lo calculamos de la siguiente forma en el método CameraConstraints:

Tenemos dos constantes que serán los radios de la pelota y el coche

```
const float cocheRadius = 2.0f;
const float ballRadius = 1.0f;
```

Obtenemos la posición del coche y de la pelota

```
glm::vec3 cochepos = glm::vec3(loc[3][0], loc[3][1], loc[3][2]); // Posición del co
```

```
glm::vec3 ballpos = glm::vec3(locball[3][0], locball[3][1], locball[3][2]); // Posición de
```

Con la funcion distance calculamos la distancia entre el coche y la pelota

```
float distance = glm::distance(cochepos, ballpos);
```

Y si la distancia es menor a la suma del radio del coche más el radio de la pelota quiere decir que hay impacto, y asignamos a la pelota la dirección y velocidad del coche antes del impacto.

```
if (distance < cocheRadius + ballRadius) {

    GLfloat moveStep = mercedes->GetMoveStep();
    vy = 9.8/60.f;
    locball[2][0] = loc[2][0];
    locball[2][1] = loc[2][1];
    locball[2][2] = loc[2][2];
    locball[0][0] = loc[0][0];
    locball[0][1] = loc[0][1];
    locball[0][2] = loc[0][2];

    bola->location = locball;

    //Velocidad X
        bola->SetMoveStep(moveStep);
    //Velocidad Y
        bola->Move(vy*0.5f);


}
```

7. Descripción de detección

7.1 Choque con paredes

Como conocemos las posiciones límite del terreno de juego, solo tenemos que comprobar en el método CameraConstraints que la posición del coche, pelota o cámara no se pasa del valor máximo o mínimo en ninguno de los ejes. Un ejemplo con el coche:

```
glm::mat4 loc = scene->coche->GetLocation();
glm::vec3 cochepos = glm::vec3(loc[3][0], loc[3][1], loc[3][2]); // Posición del coche
glm::vec3 cocheDir = glm::vec3(loc[2][0], loc[2][1], loc[2][2]); // Dirección del coche
glm::vec3 cocheRight = glm::vec3(loc[0][0], loc[0][1], loc[0][2]); // Vector Right del coche
glm::vec3 cocheup = glm::vec3(loc[1][0], loc[1][1], loc[1][2]); // Vector Right del coche

int res = 0;

if (cochepos.x > 96.0f) { cochepos.x = 96.0f;  res = 1; cocheDir = glm::reflect(-cocheDir, glm::vec3(-1.0f, 0.0f, 0.0f)); }
if (cochepos.x < -96.0f) { cochepos.x = -96.0f;  res = 1; cocheDir = glm::reflect(-cocheDir, glm::vec3(1.0f, 0.0f, 0.0f)); }
if (cochepos.z > 46.0f) { cochepos.z = 46.0f;  res = 1; cocheDir = glm::reflect(-cocheDir, glm::vec3(0.0f, 0.0f, 1.0f)); }
if (cochepos.z < -46.0f) { cochepos.z = -46.0f; res = 1; cocheDir = glm::reflect(-cocheDir, glm::vec3(0.0f, 0.0f, 1.0f)); }
if (res == 1)
{
    loc[2][0] = cocheDir.x;
    loc[2][1] = cocheDir.y;
    loc[2][2] = cocheDir.z;
    cocheRight = glm::cross(cocheup, cocheDir);
    loc[0][0] = cocheRight.x;
    loc[0][1] = cocheRight.y;
    loc[0][2] = cocheRight.z;
    scene->coche->SetLocation(loc);
}
```

En nuestro caso, si detecta que el coche choca con una pared, hace que “rebote” asignando como nueva dirección, el resultado de usar la función reflect para la dirección del coche y la normal de la pared.

7.2 Gol

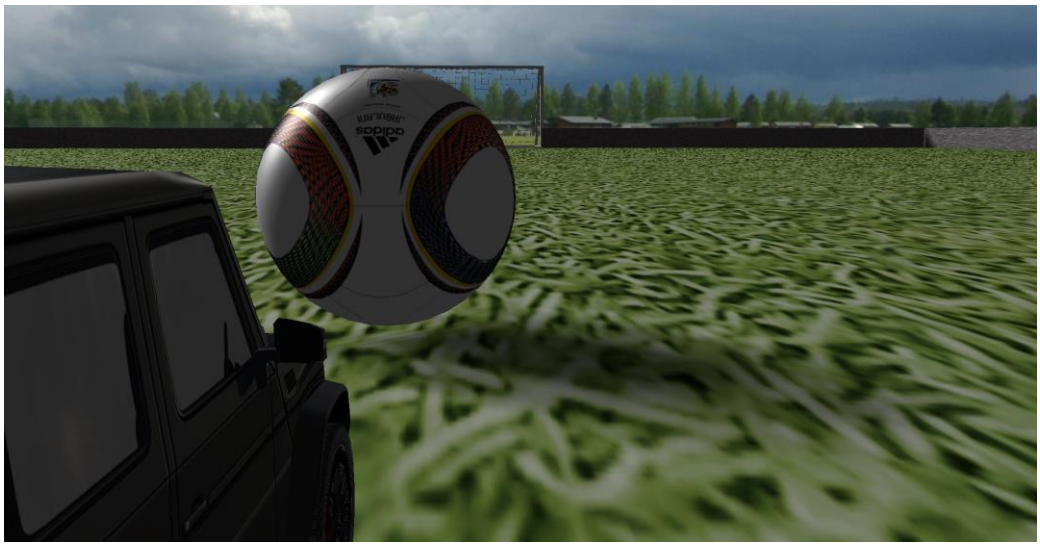
Al igual que antes se trata de calcular constantemente si la posición de la pelota se encuentra dentro de alguna de las porterías.

```
    glm::mat4 locball = scene->fig4->location;
    glm::vec3 ballpos = glm::vec3(locball[3][0], locball[3][1], locball[3][2]); // Posición de la pelota
    glm::vec3 ballDir = glm::vec3(locball[2][0], locball[2][1], locball[2][2]); // Dirección de la pelota
    glm::vec3 ballright = glm::vec3(locball[0][0], locball[0][1], locball[0][2]); // Vector Right de la pelota
    int balllimit = 0;


    if (ballpos.x > 101.0f &&( ballpos.z < 10.0f && ballpos.z > -10.0f && ballpos.y < 6.5f)) { balllimit = 4; }
    if (ballpos.x < -101.0f && (ballpos.z < 10.0f && ballpos.z > -10.0f && ballpos.y < 6.5f)) { balllimit = 4; }


if (balllimit == 4)
    {

    CGApplication terminar;
    terminar.cleanup();

    }
```

Si detecta que la pelota se encuentra en alguna de las porterías, cambia el valor de la variable balllimit a 4 y finaliza la aplicación llamando al método cleanup de la clase CGApplication.

8. Pruebas de funcionamiento

Modo cámara 1



Modo cámara 2

Modo cámara 3

9. Referencias

Francisco José Moreno Velo “Programa teórico”
https://www.uhu.es/francisco.moreno/gii_rv/

Francisco José Moreno Velo “Programa de laboratorio”
https://www.uhu.es/francisco.moreno/gii_rv/

Francisco José Moreno Velo “Entrenamiento de Rocket League”
https://www.uhu.es/francisco.moreno/gii_rv/trabajo/trabajo.htm

Wikipedia “Rocket League” https://es.wikipedia.org/wiki/Rocket_League

Modelo Portería https://free3d.com/3d-model/football-goal-473670.html

Marcador https://depositphotos.com/es/photo/soccer-scoreboard-score-1-0-48120341.html