Camera

1. Introduction

This tutorial introduces the camera in the framework and how to use it.

2. Design of the Camera

We use a camera to act as eyes for the user in the game. In the camera we can hold the position and orientation of the camera so that we can calculate the View matrix, but we can also hold the matrix used in defining the projection of the 3D world onto the 2D screen.

Therefore, the camera is a useful container for relevant information pertaining to the View and Projection.

The camera is also very useful for calculating audio, collision detection and even has some uses in AI – more on these in the lectures and lab tutorials.

3. Exercises

Please attempt these exercises, but if you get stuck or you are confused then ask for help during the scheduled lab times.

1. Look at the **Camera** class until you understand what it is doing and how it works.

2. You will notice that when you select a cursor key the camera rotates/moves. However, it does this in a jerky manner. This is because the OS sends the key commands in intervals rather than continuously.

   Add code to your game so that the key method in **GameScene** records the key presses. Your game can use the record of key presses. It is suggested that you create an array of 255 bools to hold the key states as below.

```
bool [] keysPressed = new bool[255];
```

Move the existing code inside of **Keyboard_KeyDown()** into the **Update()** method of **GameScene** – we will edit this code to work shortly.

Change the **Keyboard_KeyDown()** method to the following.

```
public void Keyboard_KeyDown(KeyboardKeyEventArgs e)
{
    keysPressed[(char)e.Key] = true;
}
```

You will need another method in **GameScene** that responds to key releases so that you can stop recording a key being pressed as below. You also need to add this (and remove it in the Close() method) from the **sceneManager.keyboardUpDelegate** delegate in **GameScene**.

```
public void Keyboard_KeyUp(KeyboardKeyEventArgs e)
{
    keysPressed[(char)e.Key] = false;
}
```

Edit the key code in **Update()** that you copied from **Keyboard_KeyDown()**. Test the bool values in **keysPressed** for each desired key, for eg:

```
if (keysPressed[(char)Keys.Up])
{
    camera.MoveForward(0.1f);
}
```

3. You will notice that we are reliant on the delegate keyboard updates. These delegates are called in the SceneManager that in turn get notified from the OpenTK event handlers that in turn are notified from the OS.

   If we want to have similar keyboard controls in another Scene, we will have to copy the keyboard handling code from GameScene. This suggests that there is probably a better design for handling Input.

   In previous lectures we discussed that you can access the state of the Keyboard (and other input devices) directly without using the event handlers. Consider what an InputManager may look like and how it would handle input for all Scenes.