CO1301- Games Concepts
Worksheet 3 - 3D Engines: Control and Cameras

Introduction

In the previous two weeks we introduced the TL-Engine and used it to manipulate some simple 3D graphics. In this session you will create your own camera controls.

You have used a FPS camera. The FPS camera uses the cursor keys and the mouse. It would be better if you could set up the camera controls according to your own preferences. At the same time we can set up the camera so that movement and rotation speed can be adjusted so that they suit the machine you are working.

You will need to keep the code that you write. For example, you will be expected to re-use the camera control code you write today for use in later weeks. **You need to take a backup of your code.**

We would suggest that for convenience you buy a pendrive.

Using "if"

Last week you used the "if" statement. One of the exercises using "if" was to add the Escape key as a way of exiting from your program. Refresh your memory about this:

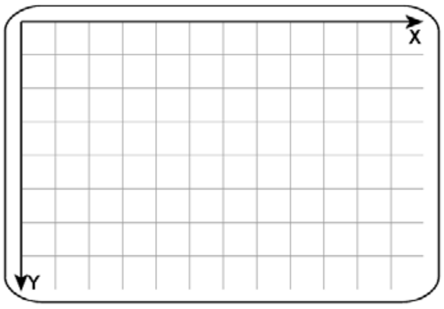
Type this code into the 'scene update' area:

```
if ( myEngine->KeyHit( Key_Escape ))
{
      myEngine->Stop();
}
```

This tests if the escape key has been pressed and quits the engine (and program) if it has. You may want to add this to all your projects so you don't need to use 'Alt-F4'.

Mouse Input

The 3D engine can provide the position and movement of the mouse cursor. The mouse cursor uses the X and Y axes of the screen, which are different than the axes in the 3D engine. The screen axes look like this:



To get mouse information from the engine use these methods:

```
// Returns the X position of the mouse (horizontal position relative to
// the top-left of the 3D engine window).
int GetMouseX();

// Returns the Y position of the mouse (vertical position relative to
// the top-left of the 3D engine window).
int GetMouseY();

// Returns the X mouse movement since the last call to this function
int GetMouseMovementX();

// Returns the Y mouse movement since the last call to this function
int GetMouseMovementY();
```

The first two methods return the X and Y position of the mouse cursor in the engine window. This can be used for clicking on buttons or icons - we won't be using these methods right now.

The second two methods tell you how much the mouse has been moved in the X and Y directions (since the last time you asked). This is a common way to control the rotation of 3D objects.

Mouse Control - Exercises

- We want to use the mouse input in our program now, but the FPS camera is using the mouse - so change the camera type from 'kFPS' to 'kManual'. This means we can't move the camera any more, so remove the "Grid.x" mesh again so we can see better.
- Remove the lines to rotate the arrow in your program including the lines to test if the keys 'Z' and 'X' are pressed. Replace them with two lines similar to these (but use your own model name):

```
int mouseMoveX = myEngine->GetMouseMovementX();
arrow->RotateY( mouseMoveX * 0.1 ); // the 0.1 reduces the rotation speed
```

This line is should not be inside an 'if' statement – it should always be executed.

- Run the program, you will be able to control the cube rotation with the mouse – but only when the mouse cursor is in the engine window, we will fix this later.
- Notice that we use the mouse X-axis movement to control rotation around the 3D engine Y-axis – this is confusing, but correct: Try using the mouse Y movement instead to see what happens (move the mouse up and down). It is often best to experiment in these cases.
- Now add a line to control the arrow's X rotation with the mouse Y movement.
- We now have complete control over the arrow using the mouse and keys. Save your project and start a new one.

Camerawork - Exercises

In a new project create a simple scene with models of your own choice. Set the camera type to 'kManual' again - you can position, move and rotate a manual camera using exactly the same methods as you have used with models.

- First, set up a value to control the speed of the camera. Declare a constant called "kCameraSpeed" at the top of your code and initialise it to a suitable value. For example:

```
       float kCameraSpeed = 0.001f;
```

- This constant will be used to control the camera speed in all of the directions of movement. Constants are declared outside the main function. Declare it at the top of your program just under the statement:

```
using namespace tle;
```

- Whenever the compiler sees the word "kCameraSpeed" it will use the numerical value that you specified.

- Now set up a constant called "kCameraRotation" and initialise it to a suitable value.
- Control the camera's movement with the same keys that you used for the cube: 'A' & 'D' = left and right, 'W' and 'S' = forward and backward, 'Q' & 'E' = up and down. You should use the local movement methods.
- Next use the mouse movement to control the camera rotation.
- Your final program should be very similar to the previous one.

You should have recreated a simple FPS camera, but we still have the problem that it only works when the mouse cursor is in the engine window.

Save your work. You need to reuse the camera setup in later program you write. The two constants which have declared mean that the code is flexible. You can use it in the Games Lab, in the University labs or at home and merely adjust the constants to suit the speed of the computer you are working on.

Add this line in the 'scene set up' area:

```
myEngine->StartMouseCapture();
```

This will switch off the mouse cursor and send all mouse information to the engine. You can restore the mouse with:

```
myEngine->StopMouseCapture();
```

- Create a toggle button to switch between using a captured mouse (cursor invisible) and using a non-captured mouse (cursor is visible). I would suggest using the TAB key.

Advanced Exercise

- Add another constant to control your FPS camera called "kMouseRotation". Use mouse input to control the rotation of the camera.