CO1301- Games Concepts
Worksheet 2 - 3D Engines: Input and Control

Introduction

Last week we introduced the TL-Engine and used it to manipulate some simple 3D graphics. In this session you will use keyboard input to move objects within the game world.

It is useful for you to keep a copy of the code that you write. You can set up a working directory (folder) on the D drive to contain your work. However, remember that you cannot guarantee that material left on the hard drive will not be wiped. **You need to take a backup of your code.**

We would suggest that for convenience you buy a pendrive.

Keyboard Input

The 3D engine monitors the keyboard and mouse input for its own window. We can ask the engine if particular keys or mouse buttons have been pressed, the methods to do this are:

```
// Returns true when a given key or button is first pressed down.
bool KeyHit( EKeyCode );
// Example usage:
//     if (myEngine->KeyHit( Key_A )) { Do something }


// Returns true as long as a given key or button is held down.
bool KeyHeld( EKeyCode );
// Example usage:
//     if (myEngine->KeyHeld( Mouse_LButton )) { Do something }
```

Note: The above code snippets show the method's ***signatures*** – the return type of the method (bool in both cases), the method name (e.g. KeyHit), and the parameter type (EKeyCode). You should not copy the method signature into your program (otherwise the compiler will think you are trying to re-define the methods!).

When you **call** the methods, you can assign the result to a variable of the correct return type or, since the return type is bool, use the method call as the condition in an "if" statement, and you must pass in a parameter of the required type. The example usage (in the green comments above) show this later use of the methods.

You must supply a special key code to indicate which key (or button) you want to test (I used 'Key_A' and 'Mouse_Lbutton' in the examples). The full list of key codes can be found here:

http://www.gamesnorthwest.net/resources/tl_engine/downloads.html

Under *Documentation -> Appendix 2: Key Codes*

Notice that mouse buttons are considered to be 'keys' also.

There is a slight difference between these two methods. The first ('KeyHit') returns true once, when the key is first pressed, then it return false until you let go of the key and press it again.

The second method (‘KeyHeld’) will always return true as long as the key is held down. We will see how this difference is important soon.

Using "if"

Selection is the ability to execute code only if some condition is met. Selection is choice.

- A block of code may be executed (if the condition is true).
- A block of code may not be executed (if the condition is not true).

In C++ you exercise this control through the use of the word "if". The general form of an "if" statement is:

```
if (conditional)
{
      // conditional exectution
}
```

The *conditional* is a programming statement. It is any statement which equates to either true or false. Any expression which equates to a number can also be used.

Note that a conditional statement does not execute a command. It is used to enable decision making in programs.

Here is a simple example, 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’.

Keyboard Control - Exercises

- Add a cube to your scene (load the mesh “Cube.x”). Use the ‘KeyHeld’ method to move it right when you hold the ‘D’ key (i.e. move it along the X axis).
- Hints: you need to use another ‘if’ statement. The key code for ‘D’ is ‘Key_D’.
- Try using the ‘KeyHit’ method instead - can you see the difference? Now change it back.
- Next, make the cube move left when the key ‘A’ is held
- Now move the cube forwards and backward (Z axis) with the keys ‘W’ and ‘S’.
- Finally, move the cube up and down (Y axis) with ‘Q’ and ‘E’.

You can now fully control the movement of a model. Now we will see what happens when we control the rotation.

- Add a new “Grid.x” mesh to your scene and change the cube model from “Cube.x” to “Arrow.x”.
- Add code to rotate the arrow left and right when the keys ‘Z’ and ‘X’ are pressed (use the ‘RotateY’ method).

- Run the program and rotate the arrow right about 45°. Now move it left and right with ‘A’ and ‘D’. Rotate it some more and try again. What do you notice about the direction that it moves?

What you should notice is that the arrow always moves left and right in the same direction: in the X direction of the large grid. This is called the ‘world’ X direction. Look closely at the arrow model and you can see it has its own grid, and it has its own X direction. This is called the arrow’s ‘local’ X direction. Sometimes we want to move in the local directions, we can do this with these methods:

```
// Move in the local X direction by the given amount
void MoveLocalX(const float);
// E.g arrow->MoveLocalX( 0.1 );

// Move in the local Y direction by the given amount
void MoveLocalY(const float);

// Move in the local Z direction by the given amount
void MoveLocalZ(const float);
```

- Change all the ‘Move’ methods from the previous exercises to ‘MoveLocal’ methods. Now experiment with rotation and movement together. Can you see the difference?

PART 2 - Variables

The last section of last week’s worksheet (the bit called Further Positioning and Movement), required the use of a control variable, which was explained informally in the lab…

A variable is used to store numbers or other kinds of data. Each variable has a data type. The data type specifies the type of data that the variable can store. C++ uses a number of different data types:

```
float myNumber;         // floating point- a decimal number
int   anotherNumber;    // integer, a whole number
bool  check;            // a boolean - can only take the values true and false
```

You can assign a value to a variable. Assignment is done using the equals symbol:

```
int number;
number = 10; // number is assigned a value of 10
bool check;
check = true; // true and false are the two boolean states
```

Values are placed into a variable using assignment. C++ uses the equals sign to signify the assignment command. Assignment is not equals!

```
int a;
a = 10; // a is assigned the value 10
```

The contents of a variable can be changed. The ++ operation increments the value of an integer by 1, for example:

```
int i = 4;
i++; // i now has a value of 5
```

Whenever the variable is subsequently used *the value* it currently holds is referenced, e.g.

```
int firstVar = 7;
int secondVar;
secondVar = firstVar;   // secondVar is assigned the value held by firstVar,
                        // i.e. 7
```

Testing variables

The following code means "if the *firstVariable* is equal to the *secondVariable* " then do the code that is inside the curly brackets.

```
if ( a == b )
{
      // do something
}
```

The following code means "if the *firstVariable* is not equal to the *secondVariable* " then do the code that is inside the curly brackets.

```
if ( a != b )
{
      // do something
}
```

The following code means "if *firstVariable* is smaller than *secondVariable* " then do the code that is inside the curly brackets.

```
if ( a < b )
{
      // do something
}
```

The following code means "if *firstVariable* is greater than *secondVariable* " then do the code that is inside the curly brackets.

```
if ( a > b )
{
      // do something
}
```

- Load the cube mesh and then create a cube model.

```
      cubeMesh = myEngine->LoadMesh( "Cube.x" );
      cube = cubeMesh->CreateModel();
```

- Declare a variable of type *float*. The name of the variable should be *cubeSpeed*.
- Assign an appropriate value to *cubeSpeed*. For this exercise you should declare and initialise the variable before the game loop (not inside it).

- Using the variable which you have just created move the cube to the right.

```
cube->MoveX( cubeSpeed );
```

You can obtain the current location of a model. For instance, you can obtain the current x coordinate of the cube by using:

```
xCoordinate = cube->GetX(  );
```

Note that I have used a new variable to store the location of the cube. This new variable is called *xCoordinate*. The GetX() function *returns* a value. In other words, it passes a value back and this value can either be used directly or assigned to a variable.

- Now get the cube to stop moving when it reaches 40 along the x axis, i.e. when GetX( ) returns a value of 40.
- You will have to use the **if** statement in order to implement the check.

Using a variable to control movement

Last week you were challenged to write the following program:

- Bounce the cube left and right between X=-40 and X=40. You will need to use the properties of variables in order to implement this.
- Hint: the most elegant solution is to use a variable to determine the *direction* of movement.
- Can you make the cube roll while it bounces?

If you haven't completed this exercise from last week, do so now.
Expand your program so you can use the keyboard in the following way:

- Whenever the 'R' key is pressed, the cube changes direction
- The 'P' key acts as a pause button for your "game".
- Hint: You will need another variable to implement the pause button.