CO1301: Games Concepts
Lab 3 - Textures,Cameras and Control

1. Introduction

1. In the previous two weeks we introduced the TL-Engine and used it to manipulate some simple 3D graphics. We have also used a FPS camera which is controlled using the cursor keys and the mouse.

2. It would be better if you could set up the camera controls, speed and rotation based on your own preferences and the machine you are working on. Therefore in this session you will create your own camera controls.

3. But before creating custom cameras you will first learn to manipulate the texture of models.

4. Remember 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. Therefore, as always, make sure you save copies of your code and projects in secure accessible locations.

2. Texture Manipulation

1. Create a new porject named **Lab3_TextureManipulation_Project**. Open the project.

2. It is possible to change the skin on a model (the image that is drawn on the geometry). To do so, you will need to use this method on the model:

   ```
   void SetSkin( const string& sFileName );
   // Example usage: sphere->SetSkin( "Marine2.jpg" );
   //only works for models that use a single texture.
   ```

3. The TL-Engine sets a default location for all of its media files. This location is set in the initialisation section as part of the template:

   ```
   myEngine->AddMediaFolder( "C\\..." );
   ```

   The path provided as a paramter to the function above is the location of the media files. This will probably be a folder in C on your home computer. If you look inside the directory you can examine the media files installed with TL-Engine.

4. Now add and display a sphere in your scene so that you can change its skin. Here are some examples of textures designed to be used with the sphere:

   - Clouds.jpg
   - EarthHi.jpg
   - EarthNight.jpg
   - EarthPlain.jpg
   - Jupiter.jpg
   - Mars.jpg
   - Saturn.jpg
   - Pluto.jpg

5. Use the SetSkin() method to change the skin of the sphere. For example to use "Jupiter.jpg" as the skin, add the code below after creating the sphere model:

   ```
   sphere->SetSkin( "Jupiter.jpg" );
   ```

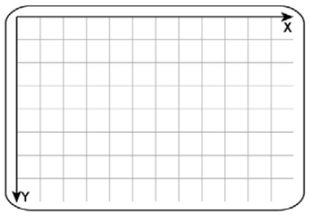
6. Add code to move the sphere left and right between X = -40 and X = 40. Make the sphere roll while it moves and change the texture of the sphere every time that it changes direction. You can use textures of your choice.

7. Save your project.

3. Mouse Control

1. Last week, we used 'if' statements to check whether mouse buttons are held or pressed.

2. The 3D engine can also provide the position and movement of the mouse cursor using the X and Y axes of the screen, which are different than the axes in the 3D engine. The screen axes look like the image below:



3. 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.

4. Exercises - Mouse Control

1. Open last week's lab project, Lab2_Controls_Project, so that we can add code to handle mouse input for control.

2. We want to use the mouse input in **Lab2_Controls_Project**, but the FPS camera is using the mouse - so change the camera type from 'kFPS' to 'kManual'. Remeber from last week's lab that this means we can't move the camera any more, so remove the "Grid.x" mesh again so we can see better.

3. Remove the lines of code that 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 code should not be inside an 'if' statement – it should always be executed.

4. 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.
5. 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.
6. Now add a line of code to control the arrow's X rotation with the mouse Y movement.
7. We now have complete control over the arrow using the mouse and keys. Save your project.

5. Exercises - Camera Control

1. Create a new TL_Engine project named **Lab3_CameraControl_Project**
2. Create and display models of your own choice. Set the camera type to 'kManual' again, although this camera is fixed by default you can add code to position, move and rotate a it using exactly the same methods as you have used with models.
3. First, set up a value to control the speed of the camera. Declare a constant called "kCameraSpeed" outside the main function just under the statement `using namespace tle;` (constants are declared outside the main function). Initialise "kCameraSpeed" to a suitable value using the **const** keyword, for example:

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

   This specifies that the variable's value is constant and tells the compiler to prevent the programmer from modifying it
4. Whenever the compiler sees the word "kCameraSpeed" it will use the numerical value that you specified..
5. Now set up another constant called "kCameraRotation" and initialise it to a suitable value.
6. 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.
7. Next use the mouse movement to control the camera rotation.
8. Your final program should be very similar to the previous one.
9. 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.
10. Add this line in the initialisation section within the 'int main' function:

    ```
    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();
    ```
11. 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.

Hint: You will need to use an 'if' statement to check for the appropriate key press inside the game loop.

12. You will need to reuse the camera setup in later programs you write. The two constants which you have declared mean that the code is flexible. You can use it in new projects and merely adjust the constants to suit the speed of the computer you are working on.

6. Exercises - Controlled Rotation

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