Tutorial for Week 1 in Further Games And Graphics Concepts COSE50581

1. Download the Framework in the link provided and examine the code.
2. Your tutor will describe the main sections of the framework we are using in this module.
3. Be sure to make notes on the different sections described.
4. Compile and run the framework using Visual Studio 2017
5. If you are unsure of the purpose of any code in the framework please ask your tutor.
6. Task 1:
7. Extend the framework given to draw a spinning cube
8. Task 2:
9. Draw multiple cubes at different positions and rotate them differently in the scene

Much of the code will be discussed in future lectures, however here are some pointers.

The file DX11 Framework.fx is where the code is contained for the pixel and vertex shaders. The function VS is the vertex shader, and PS is the pixel shader. You can see in the vertex shader where the vertex position is transformed by the World, View and Projection matrices, as discussed in the first lecture.

The pixel shader simply outputs the colour of a pixel. You could try hard-coding this to see how it changes the colour. For example: red would be

return float4(1,0,0,1);

green would be:

return float4(0,0,1,1);

What do you think blue would be? What about yellow?

Hint: each element in the float4 equates to (red, green, blue, alpha). Leave alpha for 1 for now, but you will learn more about this later.

In the ‘Application.cpp’ – this contains much of the code for rendering the square.

Application::Initialise

You can see code here for setting the view matrix.

Look up ‘XMMatrixLookAtLH’ in the directX documentation and see how it works.

Application::InitShadersAndInputLayout()

This function contains some code for compiling the pixel and vertex shaders – they need to be compiled by the C++ code, the resulting code is then used by directx for rendering meshes.

Application::InitVertexBuffer()

This function initialises the vertex buffer – a vertex buffer contains the vertices and colours for the mesh:

SimpleVertex vertices[] =

{

{ XMFLOAT3( -1.0f, 1.0f, 0.0f ), XMFLOAT4( 0.0f, 0.0f, 1.0f, 1.0f ) },

You can see that the first ‘float3’ is the position, and the second ‘float4’ is the colour. Try changing these and see what happens to the mesh (if you have changed the pixel shader to output a constant colour you will need to change it back).

Application::InitIndexBuffer()

While the vertex buffer contains the vertex information (vertex and colour data, in this instance), the index buffer tells directx which vertices to use to render two triangles to form a square. So, in this instance, it says to use vertices 0,1,2 for the first triangle, and 2,1,3 for the second triangle.

Try adding a third triangle: you will need to change the vertex buffer (if adding any vertices), plus also the ByteWidth variable below.

You will need to change the index buffer (and the ByteWidth variable again).

Finally you will need to change the DrawIndexed function in the ‘Draw()’ method. If you don’t understand how to change this, look in the directx documentation.

Finally, to draw a cube you will almost certainly need a pen and paper, unless you are very good at visualising 3D positions. Note the ‘winding order’ of triangles is important – check the existing triangles: it will need to clockwise for each triangle.

Once you have done this, go through the code and anything you don’t understand – look it up in the directx documentation or the course text books.