COSE50581 Tutorial Week 3

Before starting this tutorial ensure that you have finished the work from last week. You should now have:

* A cube solar system, with a number of objects rotating around each other. This should demonstrate your understanding of composite matrices in DirectX 11.
* Depth buffer enabled, and a key to switch your rendering into wireframe mode.

If you have not finished these tasks, please ask your tutor for help fixing any problems you are having.

## Adding a new object

## Construct the vertex and index list of a pyramid, as shown below, and draw it. Colour the base vertices green and the tip vertex red.

## Macintosh HD:Users:Steve:Desktop:Screen Shot 2015-10-08 at 23.55.05.png

1. First you will need to add a new vertex and index buffer containing the new vertex and index data. You will still use the SimpleVertex structure for your vertex data, but should now have **separate** vertex and index buffers for each object. E.g. VB and IB for the cube, and VB and IB for the pyramid. Do not use the same buffer for both objects.

HINT: Remember to have the pyramid centred around the origin of its local coordinate system.

1. After you have created your buffers, now find the lines of code that set the Input Assembler to use the cube VB and IBs. Change these lines to now use your new pyramid, and also alter the number of indices used by DrawIndexed accordingly. This should change all your drawn objects to be the pyramid.
2. Now move the code to set the Input Assembler buffers to your Draw function, drawing some of your objects and cubes and others as pyramids.

## Altering data in the vertex shader

Modify the your code by applying the following transformation to each vertex in the vertex shader prior to transforming to world space. This will animate the vertices as a function of time by distorting them periodically with the sine function.

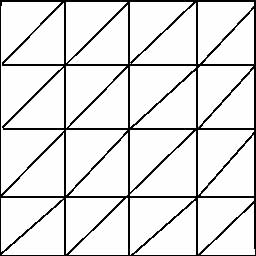
Pos.xy += 0.5f \* sin(Pos.x) \* sin(3.0f \* gTime);

Pos.z \*= 0.6f + 0.4f \* sin(2.0f \* gTime);

1. The “gTime” variable needs to be an increasing time value, this will allow the vertices to oscillate with a sin wave over time. To do this we will need to pass a time value from our application to the shader. Add a new float variable to your C++ constant buffer, and also your shader constant buffer. Be sure to add it at the end of each of the buffers.
2. Now assign the ‘t’ variable in your Update method to your C++ constant buffer. This will allow it to be passed across to the shader and used in the calculations.
3. Add the above code to your vertex shader before the world space transformation. On compilation you should now see your vertices moving with a sin wave.
4. Experiment with the formula above to achieve different effects.

## Generating a flat vertex grid

1. Use the knowledge you gained above to create a new object (this will require new Vertex and Index buffers).
2. This object should be a flat grid that is 4x4 triangles wide and high. The diagram below shows the structure:



1. Render the object as a large floor plane below your other objects in the scene. You should now have three separate objects being rendered, all using different vertex and index buffers.

## Additional Task

Consider how you would generate a flat triangle grid of any width and height, can you create a function that would set up a vertex and index buffer with the correct data given only the dimensions of the grid?