COMP30019 – Graphics and Interaction
Lab 1

The University of Melbourne

Introduction:

In this lab you will be introduced to the two key tools we will be using in this subject; namely, Unity and Blender. Unity is a powerful games and simulation engine which has been used in many major titles. Blender is an open-source 3D modelling tool which facilitates the construction of 3D assets, often used in conjunction with Unity to create rich interactive scenes.

You will need to work with three files to start off with:

- *MainScene.unity* – A Unity scene containing a partially formed cube entity. You will be working with this scene.
- *CubeScript.cs* – The C# script which constructs and renders the cube geometry, attached to the cube entity.
- *VertexColorShader.shader* – A Cg/HLSL shader used to render the cube. Note that you do not need to worry about the workings of shaders for this lab. We will briefly modify the file for the purposes of some demonstrations, but you do not need to understand how the code in it works.

Tasks:

1) Open *MainScene.unity* in Unity. Press the ‘Play’ button and you should see a partially rendered cube. Switch to the ‘Scene’ tab so you can navigate the view using the mouse: The middle button/scroll wheel allows you to drag/zoom the view, and dragging with the right mouse button allows you to rotate the view.
   a. Navigate the view to look at the cube from different angles. It may help to right-click the ‘scene gizmo’ located in the top-right of the viewport in order to select different angles to view the cube from. Which two faces are missing?
   b. Open *VertexColorShader.shader* and remove the code “Cull Off”, and save the file. Make sure you keep this code removed for the remainder of the lab. Examine the cube again in the Unity editor. What did “Cull Off” do?
2) Add vertex definitions to *CubeScript.cs* to complete the cube (add the missing faces). Make sure you use the correct vertex winding order such that the triangles you define are visible from the outside of the cube.
3) Create a new script component called *PyramidScript.cs* to define an upward facing square pyramid. The tip of the pyramid should be located at (0.0, 1.0, 0.0). You may wish to copy the cube script as a starting point. Give each face a unique colour. Remember to create a new empty game object to attach the script to, and make sure you set its transform component so it does not overlap the cube in any way.
4) Navigate the view to look from the inside of the cube. Since back-face culling is on, you won’t be able to see the inside faces of the cube. Modify the index generation code in *CubeScript.cs* (near the end of the script) so that you can see the interior of the cube instead of the exterior.

Constructing 3D meshes by programmatically defining vertices is tedious, however the ability to do so is sometimes useful if we wish to procedurally generate a mesh based on some sort of algorithm. On the other hand, modelling tools such as Blender exist that allow you to construct 3D objects using a graphical user interface. In the industry, graphic artists usually use tools like Blender to construct "3D models" which are then exported to a game engine like Unity for programmers to utilize. You will now work with Blender to create and export a 3D model to Unity.

5) Launch Blender. You should see a default cube at the centre of the scene. Practice navigating the view like you did in Unity. In Blender, the middle mouse button/wheel lets you rotate and zoom the view, and holding shift with the middle mouse button down allows you to drag the view (by default).
    a. Select the cube. Press 'tab' to go into "Edit Mode". All vertices will be selected by default, so press 'a' to deselect them.
    b. Right click on any vertex to select it, and then move it. Change the shape of the cube to your liking.
    c. Press 'tab' again to return to "Object Mode". Use the menu on the toolbar just under the view which currently reads "Object Mode" to change the mode to "Vertex Paint Mode". Use the colour wheel on the toolshelf panel left to the view so select colours, and then paint the vertices of the cube to your liking (press 't' if the panel isn't visible).
    d. Save the Blender project, such that the *.blend* file is in the "/assets" directory of the Unity project.
6) Return to Unity. You should notice the blender project file is now visible in the 'Project' panel; a little arrow to the left of it should expand to show its contents (the actual mesh should be visible under here). Unity is automatically treating the entire blender scene like a *prefab,* as indicated by the little blue cube icon.
    a. Drag the model from the "Project" panel onto the scene to place it in the scene.
    b. The 'standard' shader currently applied to it does not support vertex colours. Select the entity, and then change the shader property under the 'Material' section of the 'Inspector' panel to use the *VertexColorShader.cs* shader. The cube should now be coloured as you painted it in Blender.
    c. Notice how the "cube" you imported from Blender is always visible in the scene, even when Unity isn't in play mode. Why is this the case? (As opposed to the cube and pyramid we defined, which are both only visible during play mode)
    d. Examine the transform component of the imported cube. You'll notice that Unity has automatically set a transformation of -90 degrees about the x-axis. Why does Unity do this by default? Could it pose any potential issues for non-static game objects which use the model? If so, is there an alternative way to achieve the same outcome without Unity needing to apply the default transform?
7) ***Extension.*** Experiment with creating more complex models in Blender and exporting them to Unity. There are many resources online which explain Blender modelling in detail, probably worth exploring.
8) ***Extension (hard).*** Write a script to procedurally generate a sphere. It will probably be worth researching various techniques for generating them before starting.