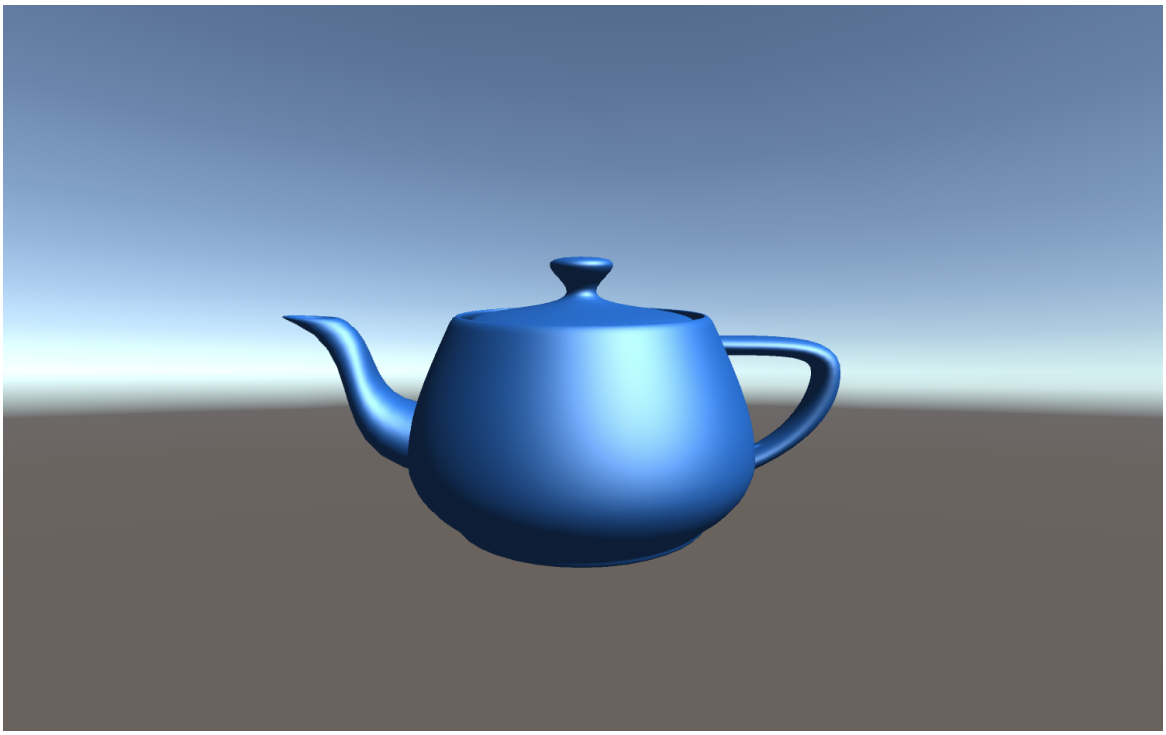
Exercise 2 - Meshes & Shading

In this exercise you will use OBJ files to draw and shade meshes.

The goal of this exercise is to learn about the rendering pipeline, meshes, shaders and lighting in 3D.

You **must** submit this exercise in pairs.



EX2 Guidelines

- In this exercise you may only edit the files MeshData.cs, BlinnPhongGouraud.shader, BlinnPhong.shader, according to the instructions.
- You are given a folder of different OBJ mesh files which you can use to check your code.
- Note that part 2 contains special guidelines for writing shaders.

General Guidelines

You may lose points for not following these guidelines.

- Make sure you are using **Unity 2020.1.6f1**
- Make sure that you understand the effect of each part of your code
- Make sure that your code does what it's supposed to do and that your results look the way they should
- Keep your code readable and clean! Avoid code duplication, comment non-trivial code and preserve coding conventions
- Keep your code efficient

Submission

Submit a single `.zip` file containing **only** the following files:

- **MeshData.cs**
- **BlinnPhongGouraud.shader**
- **BlinnPhong.shader**
- **`readme.txt`** that includes both partners' IDs and usernames. List the URLs of web pages that you used to complete this exercise, as well as the usernames of all students with whom you discussed this exercise

Deadline

Submit your solution via the course's moodle no later than **Thursday, December 8 at 23:55**.

Late submission will result in 2^{N+1} points deduction where N is the number of days between the deadline and your submission. The minimum grade is 0, saturday is excluded.

Part 0 / Setup

1. Download the exercise zip file from the course Moodle website and unzip it somewhere on your computer.

2. In Unity Hub, go to *Projects* and click the *Add* button on the top right. Select the folder that you have downloaded.

3. Open the project. Once Unity is open, double click the MainScene to open it.

Part 1 / On the CPU - Mesh Processing

In this part you will process the raw mesh data and calculate surface normals for the mesh. Additionally, you will implement a method that makes the mesh flat-shaded.

1. Click the MeshViewer GameObject in the scene hierarchy and take a look at the inspector. MeshViewer has MeshRenderer and MeshFilter components, as well as a material that visualizes surface normals at each pixel. It also has a MeshViewer.cs script component with a custom UI.

2. Enter play mode. In the MeshViewer.cs component, select an OBJ file and click “Show Mesh” to display it. The mesh will appear in a solid purplish color, because it contains no surface normal data!

3. Open MeshData.cs in the Scripts folder.

4. Implement the method `CalculateNormals` as seen in class. The method calculates surface normals for each vertex of the mesh, then saves them in the MeshData normals array.

   After implementing the method, Sphere.obj should look like this:

5. Implement the method `MakeFlatShaded`. The method edits the mesh data so that no two faces share any vertex. In other words, each face should have 3 unique vertices associated with it. The method does not change the location of the vertices.

After implementing the method and checking “Make Flat Shaded” in the MeshViewer inspector, Sphere.obj should look like this:

6. In your readme.txt file, explain why this function works - why do separate vertices for each face cause flat shading?

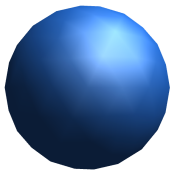
Part 2 / On the GPU - Shading

In this part you will write a shader file implementing the Blinn-Phong lighting model that you have seen in class.

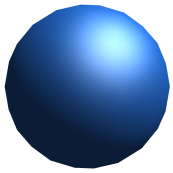
1. Open the Materials folder in the project view, and drag the BlinnPhongGouraud material onto the MeshViewer GameObject to apply it.
2. Note that this material is associated with the BlinnPhongGouraud.shader file. Open this file to edit it.

   Tip: when working on shaders you don't have to enter play mode each time to check your code. Unity automatically compiles and applies your `.shader` file after you save it. Just add some object to the scene, apply your material and you will see changes immediately!
3. Implement the vertex and fragment shaders to create Blinn-Phong lighting in the Gouraud shading model.
   - For our purposes, the light color will be the same for the specular, diffuse and ambient components. You can access this color using Unity’s `_LightColor0` variable.
   - You may not change the ShaderLab properties given in the file.
   - You may not change the vertex input structs given in the file, but you can add varyings as needed.
   - You may not use any function from `UnityCG.cginc` that you did not see in class, but you may use any built-in variable.

4. In the material inspector, play around with the material properties to test your shader and make sure it looks correct. To reset the material properties, click the cog icon ⚙ and then *reset*. When using the default values, Sphere.obj should look like this:



5. Now we will implement Blinn-Phong lighting in the Phong shading model. Drag the BlinnPhong material onto your chosen GameObject to apply it.
6. Implement the vertex and fragment shaders to create Blinn-Phong lighting in the Phong shading model.
    - The guidelines for this step are the same as in step 3.
7. When using the default values, Sphere.obj should look like this:

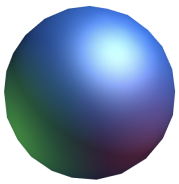


Because shaders run on the GPU, it can be hard to test your code properly. Make sure your code implements the lighting models correctly, and inspect the shaders visually!
In the Scene view, look at the object from different directions, move it, rotate it, change the light parameters, etc.

Part 3 / Additional Point Lights

In this part you will add support for multiple diffuse point lights to the Blinn-Phong shader you have written.

1. In the scene hierarchy, there are 2 point lights - a red and a green one. They currently do not affect objects in the scene, because our shaders only use the main (directional) light.
2. Open the BlinnPhong.shader file to edit it.
3. Implement the function `pointLights`. The function calculates the lighting of 4 secondary point lights, and returns the sum of their resulting colors.
    - Calculate only the diffuse part of the Phong lighting model, without the ambient and specular components.
    - You may always calculate the lighting for all 4 lights, even if there are fewer in the scene. Unused lights will have a color of 0 and will not affect the final outcome.
    - Relevant guidelines from step 3 apply here as well.
4. Add the result of `pointLights` to the final fragment color calculation. Sphere.obj should look something like this:



Good luck!