# CS 174A Assignment 3

Illumination Models

**Out:** Mon March 5, 2018.
**Due:** Sat March 17, 2018, 11:59am.

**Value:** 10% of final grade
**Total Points:** 14 points

1. The objective of this coding question is to gain some further experience with using texture mapping, vertex shaders, and fragment shaders. You will be implementing the Phong illumination model and your own environment map using shaders. See the course web pages for the starting template.

   The code runs in the browser and can thus be run by simply opening `a3.html`. You will need to enable local file access, as you did for the previous assignments, i.e.,: `https://threejs.org/docs/#Manual/Getting_Started/How_to_run_things_locally`.

   You will be making changes to the javascript (`a3.js`), the vertex shaders,(`glsl/phong.vs.glsl`, `glsl/reflective.vs.glsl`), and the fragment shaders (`glsl/phong.fs.glsl`, `glsl/reflective.fs.glsl`).
   After making edits, a page reload on your browser will then run your code again. Error messages will be displayed on the javascript console.

   Complete the following modifications to the template code in `a3.js` and the specified shaders.

   (a) *(2 points) Uniform variables:* Run the template code in your browser, you should see three solid black spheres floating in the scene. Currently all three spheres are being rendered using the Blinn-Phong shaders `glsl/blinn_phong.vs.glsl` and `glsl/blinn_phong.fs.glsl`. The Blinn Phong shaders require no further modifications. However, there are no uniform variables being passed to them from the javascript file, which causes them to be rendered black. For this first part of the assignment, you need to bind the appropriate javascript values to the uniform variables of the Blinn-Phong shader material.

   - First, look at the uniform variables being used in the fragment shader: `glsl/blinn phong.fs.glsl`
   - Second, in `a3.js`, bind the correct values to these uniforms in the assignment to the `blinn_phong_material`.

   The `lightColorUniform` uniform variable has been done for you. You'll know you are finished this step when your spheres look like the reference solution, `ref_solutions/blinn_phong.jpg`.

   Note: In the current three.js version, uniform variables are attached to individual shader materials `THREE.ShaderMaterial`, then vertex and fragment shader pairs are attached to each of these shader materials, and finally these materials are applied to the geometries we add to our scene. An easy way to think of this is that a shader material holds the uniform variables relevant to the vertex and fragment shader pair attached to it, keeping everything organized in one location and accessible through the material.

(b) *(4 points) Phong illumination:* Now that you are familiar with shader materials, implement Phong shading and apply it to one of the spheres in your scene. You will have to attach the correct uniforms to the Phong shader material first, and implement the shader code in both `glsl/phong.vs.glsl` and `glsl/phong.fs.glsl`. The Blinn-Phong model that you have just worked on is a perfect template for developing your Phong shading model, which will only have minor differences. You'll know you are finished this step when your Phong illumination sphere look like the reference solutions:
`ref_solutions/phong_front.jpg`,
`ref_solutions/phong_back.jpg`, and
`ref_solutions/phong_back_side.jpg`.

(c) *(4 points) Understanding environment map implementation:*
`(4 points) Three.js` comes with an environment map lookup function, `textureCube()`, that makes it trivial to use a direction vector, $R$, to correctly access the correct face and texel-within-the-face for a cube map. The reflective shader material already has the correct uniforms attached, and the vertex shader `shaders/reflective.vs.glsl` is already completed for you. Note that the rays being used are in world coordinates. You only need to finish the fragment shader `shaders/reflective.fs.glsl`. Start by changing your last sphere to use the reflective shader material. You should see a the sphere reflect the scene around it as shown in `ref_solutions/env_map.jpg`. The fragment shader `shaders/reflective.fs.glsl` uses the convenient textureCube() lookup function. However, in this question you will be implementing your own texture map lookup for just the top face of the cube map using the sampler2D textureUniform uniform variable being passed from the javascript file. To finish your texture map lookup:

- First, test that you can correctly identify directions that would be captured by the top face by assigning those pixels a fixed colour, i.e., green. The details of this will be discussed in class.
- Second, compute the correct $(u, v)$ texture coordinates from the $(R_x, R_y, R_z)$, and then use those texture coordinates to do the appropriate texture lookup.

You'll know you are finished this step when your reflective shader sphere shows a planet on the top of it, as shown in the example solution image,
`ref_solutions/env_map_planet.jpg`.

(d) *(4 points) Creative component:* Develop an idea of your own for augmenting the scene. The expectations here are in line with the limited amount of time available to complete this assignment. Nevertheless, use this as a last chance to experiment! You might implement refraction; it is very similar to the reflection used during environment mapping and therefore requires very few lines of code in the fragment shader. Furthermore, GLSL comes with a `refract()` function, which computes the refraction direciton for an incident direction. Other ideas might relate to building a more interesting scene, which can be done by adding objects, textures, animation, and any feature of three.js that you might care to explore. Document what you did in a `README.txt` file. Note that we will not be using face-to-face grading, and so these written explanations are important.

Include a README.txt file that contains: (a) your name; (b) your student number; (c) any comments and explanations that you wish to include.