CPSC 314
Assignment 3 : Shading

Due 11:59PM, February 28th, 2020

1 Introduction

The goal of this assignment is to explore lighting and shading of 3D models. You will be simulating seven different models: Blinn-Phong, Anisotropic, Fog (the closer, the 'clearer'), Spotlight, Toon (cartoon-like), Gooch (cool-to-warm), and hatching. In this assignment you will be completing the seven supplied pairs of shaders (both vertex and fragment shaders) to carry out these tasks, which includes editing necessary parts of the A3.js file.

1.1 Getting the Code

Assignment code is hosted on the UBC Students Github. To retrieve it onto your local machine navigate to the folder on your machine where you intend to keep your assignment code, and run the following command from the terminal or command line:

```
git clone https://github.students.cs.ubc.ca/cpsc314-2019w-t2/a3-release.git
```

1.2 Template

There are five scenes in the assignment, accessible with the number keys 1, 2, 3, 4, 5, and 6 on your keyboard. Scene 1 is the default per-vertex shader ('Gouraud Shading'). You will be implementing the reflection models Blinn-Phong in Scene 2, Anisotropic in Scene 3, Fog in Scene 4, Spotlight in Scene 5 and the three different Non-photorealistic models in Scene 6. The template code is found in the main assignment directory, shader files can be found in the glsl folder, and the obj folder contains different objects at your disposal for the last scene.
We have already defined the properties of a basic directional light in `A3.js` for the scene (defined by `lightColor`, `lightFogColor`, `lightDirection`, `spotlightPosition` and `ambientColor`). We have also defined suggested material properties for the objects you will be shading (defined in `kSpecular`, `kDiffuse`, `kAmbient`, `shininess`, `tangentDirection`, `fogDensity`, and cool/warm parameters for Gooch).

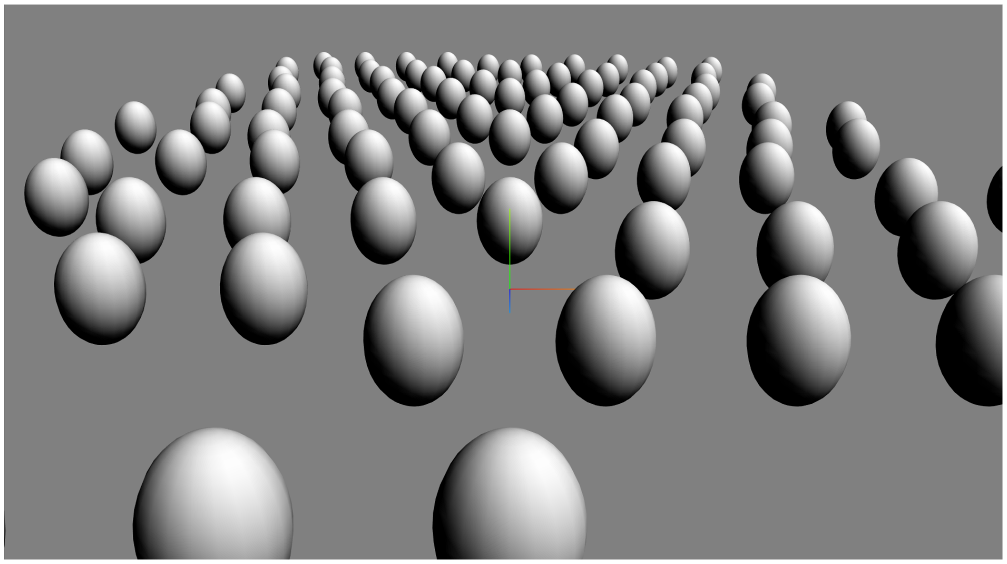

Figure 1: Initial scene with gouraud shading.

1.3 Important Rules

The lighting models you will be implementing in this assignment are very common in graphics applications. Thus three.js has already implemented most of them in a number of the default materials provided with three.js. These can be used to test if you are getting reasonable results, but you must implement these shading models yourself in your own custom shaders.

2 Work to be done (100 points)

First ensure that you can run the template code in your browser. If you have any issues, see instructions in Assignment 1. The initial scene should show an array of eggs shaded with the Gouraud model.

Part 1: (55 points) Photorealistic Shaders.

1. **(15 points)** Blinn-Phong Reflection.
   The Blinn-Phong *reflection* model builds on the Phong reflection model (and is used in physically based rendering as well). See Textbook Chapter 14.3. Instead of computing the dot product between the reflected light vector and viewer, a dot product between the halfway vector between light and viewing direction, and the surface normal can be used.

   Complete the supplied Blinn-Phong shaders (`phong_blinn.fs.glsl` and `phong_blinn.vs.glsl`) to apply the Blinn-Phong reflection model, per fragment, to the eggs in Scene 2. Once Scene 2 is complete, it should look similar to Figure 2. You can also use the built in THREE.js Phong materials to test whether your shader works correctly.

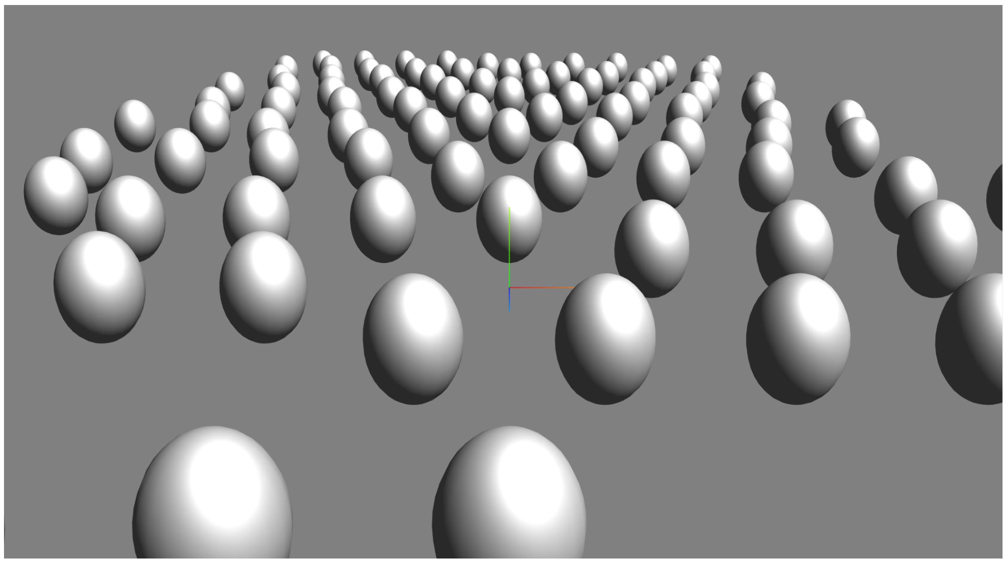

Figure 2: Example of completed Scene 2 (Blinn-Phong).

Hint 1: use the `normalMatrix` *to transform* `normal` *from World to Eye frame. See Chapter 3.6 of Textbook.*
Hint 2: Specular lighting shouldn't be static as you move the camera around the scene.

2. **(10 points)** Anisotropic Material.
   The materials used for Part 1.1 is *isotropic*, meaning there is no preferred direction of light reflection. In this task, you will implement an anisotropic material, so that the light will scatter in a preferred tangent direction, which is already defined by

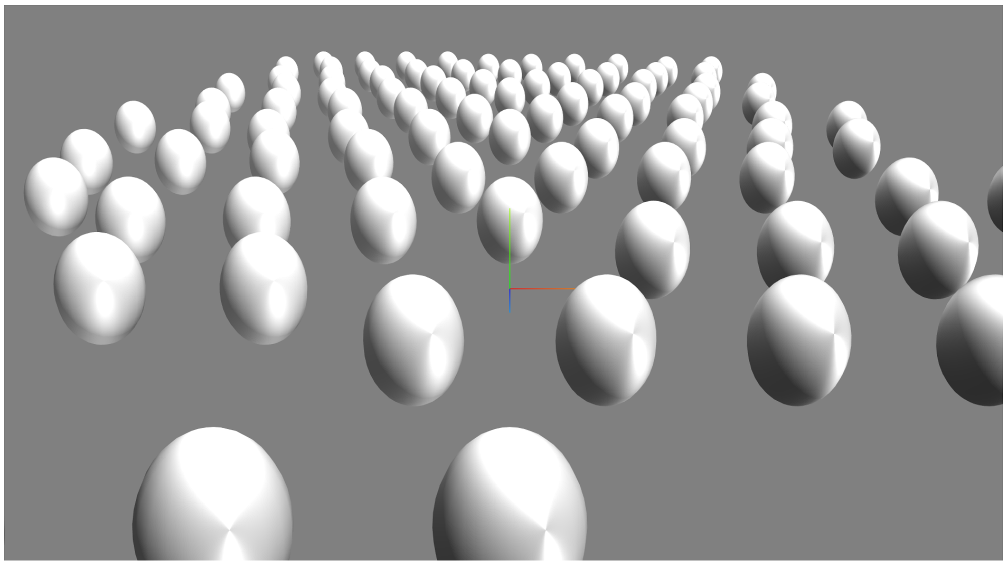

Figure 3: Example of completed Scene 3 (Anisotropic).

`tangentDirection`. Complete the anisotropic shaders (`anisotropic.fs.glsl` and `anisotropic.vs.glsl`) described in the textbook Chapter 14.4, but using the same ambient and diffuse model as Blinn-Phong. The results should look similar to Figure 3.

Hint: In the textbook example, `shininessUniform` *is* 16.0 *and* `kSpecularUniform` *is* 0.3*.*

3. **(15 points)** Fog Simulation.
In this part, you would modify the shader to simulate fog, i.e. the objects that are closer to the camera are clearer while the further ones are foggier. Complete the supplied Fog shaders(`fog.vs.glsl` and `fog.fs.glsl`). You should compute the distance between the vertex and the camera. Then you're required to apply exponential fog equation to get FogLevel for interpolation.

$$\text{FogLevel} = 1/e^{\text{distance * fogDensity}} \tag{1}$$

The final fragment color will be an interpolated value between the fog color(`lightFogColor`) and the simple diffuse color(you should calculate it by yourself). You should get a similar scene as Figure 4.

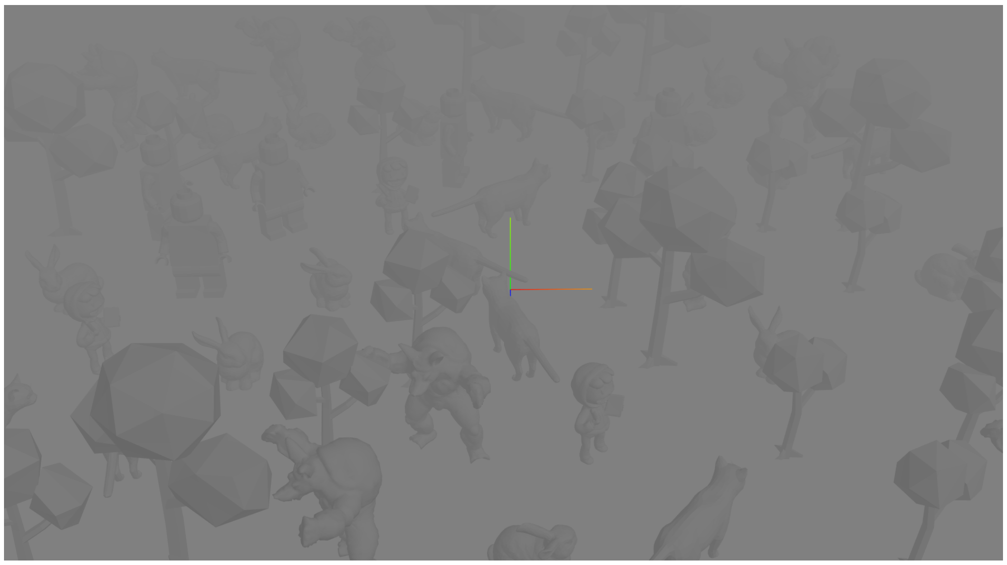

Figure 4: Example of completed Scene 4 (Fog).

4. **(15 points)** Moving Spotlight.
A spotlight is a kind of point light, which emits only a cone of light. The vertex of the cone is located at the position of the light(`spotlightPosition`). The cone points in some direction, called the spot direction, is specified as a vector. The intensity of light is dependent on the angle between the spot direction and the light ray. Your goal for this part is to create a spotlight with `spotlightPosition` fixed and spot direction moving by keyboard control. In `spotlight.fs.glsl`, you will need to judge

whether the fragments can be lit by the spotlight or not using the angle(threshold set by yourself). Then the fragments will be colored with the intensity Ic^s. (I is the `SpotColor`, c is the cosine of the angle and s is `spotExponent`). After modifying the shaders, you will need to apply this material to the Pixel model and the floor to see how it works. Your completed scene should be similar to Figure 5.

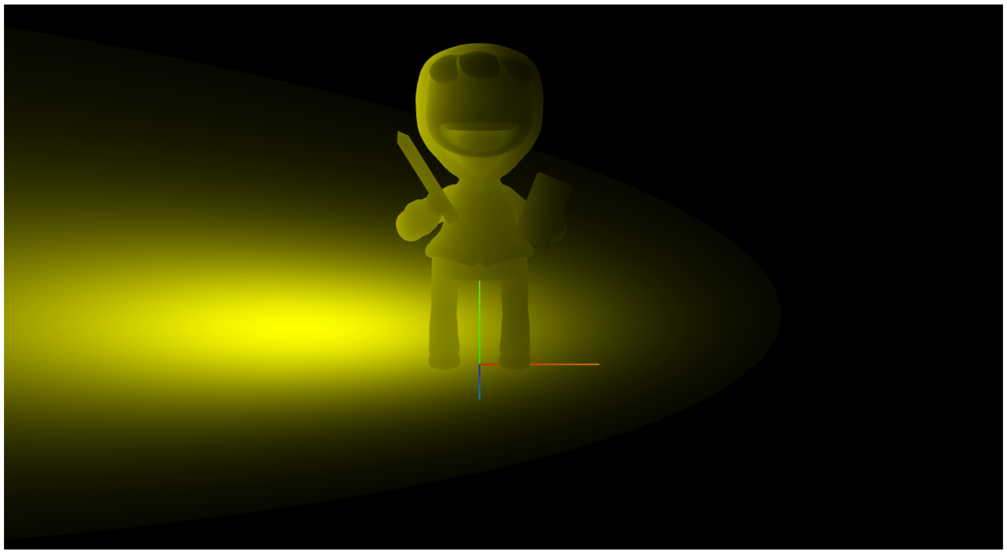

Figure 5: Example of completed Scene 5 (Spotlight).

Part 2: (45 points) Non-Photorealistic Shaders.
In Part 2, you will be implementing "non-photorealistic" (or NPR) shaders. These are shaders inspired not by "photorealistic" light scattering and reflection models, but by art styles such as cartoons, technical illustrations, and paintings. You will be implementing three different NPR shaders and apply them to each Pixel model in Scene 6.

1. **(15 points)** Toon Shading.
   Toon Shading (or Cel Shading) emulates the way cartoons use very few colors for shading, and the color changes abruptly, while still providing a sense of 3D for the model. This can be implemented by quantizing the light intensity across the surface of the object. Instead of making the intensity vary smoothly, you quantize this variation into a number of steps for each "layer" of toon shading. Use at least 3 of such "layers" based on light intensity.

   Also, draw approximate silhouette edges of the object by detecting fragments on the silhouette and coloring these fragments with a dark color (think about the relationship between the surface normal and the viewing direction for these fragments).

   Implement the Toon shaders (`toon.fs.glsl` and `toon.vs.glsl`) for the leftmost Pixel model. The completed shader should look similar to the leftmost model on Figure 6.

2. **(15 points)** Gooch Shading.
   Gooch Shading, also known as cool-to-warm shading, is a shading model inspired by

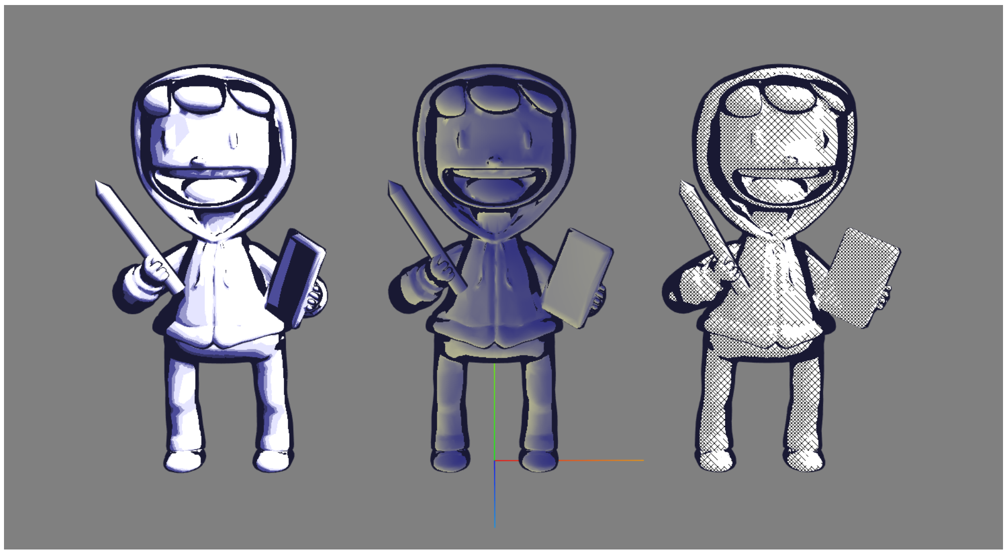

Figure 6: Example of completed Scene 6 (NPR).

technical illustrations. Essentially, the edges are shaded with a dark color (as done in the Part 2.1), then the rest of the material is shaded based on an blending of a cool and a warm color depending on the light intensity. The blending is done by the following equation,

$$I = \left(\frac{1 + \text{lightDirection} * \text{normal}}{2}\right) k_{\text{cool}} + \left(1 - \frac{1 + \text{lightDirection} * \text{normal}}{2}\right) k_{\text{warm}},$$

where,

$$k_{\text{cool}} = k_{\text{blue}} + \alpha k_{\text{diffuse}}$$
$$k_{\text{warm}} = k_{\text{yellow}} + \beta k_{\text{diffuse}}$$

The blue and yellow colors k_{blue} and k_{yellow} (`blueUniform` and `yellowUniform`) and the corresponding "prominence" parameters α and β (`alphaUniform` and `betaUniform`) is already provided in the code.

Implement the Gooch shader (`gooch.fs.glsl` and `gooch.vs.glsl`) and apply it to the second Pixel object. Your result should look similar to the one on Figure 6.

3. **(15 points)** Cross-hatching.
Hatching is an artistic technique to emulate shading effects by drawing closely spaced parallel lines. When there are parallel lines at an angle with each other, it is called cross-hatching. You will be implementing this technique as the final NPR shader (`hatch.fs.glsl` and `hatch.vs.glsl`).

Similarly to Toon shading, in the fragment shader find how closely the parallel lines should be drawn based on the total light intensity (sum of diffuse and specular). You can use `gl_fragCoord` to get the position of the fragment with respect to the window coordinates. When the shader is completed, apply the shader to the third Pixel model. Your results should look similar to Figure 6.

Hint: Use the modulo operator (`mod`) in the fragment shader to find where the hatch lines are drawn.

2.1 Hand-in Instructions

You do not have to hand in any printed code. Create a README.txt file that includes your name, student number, and CWL username, and any information you would like to pass on to the marker. Create a folder called "A3" under your "cs-314" directory, and put all the source files, your makefile, and your README.txt file for each part in the respective folder. The assignment should run without any changes directly from your submission folders. The assignment can be handed in on a department computer, which you can SSH into, with the exact command:

```
handin cs-314 A3
```

You may also use Web-Handin by following this link
`https://my.cs.ubc.ca/docs/hand-in,`

logging in with your CWL credentials, and writing cs-314 for the course, A3 for the assignment name, and zipping your assignment folder for submission. It is always in your best interest to make sure your assignment was successfully handed in. To do this, you may either use the Check submissions button in Web-Handin, or using the -c flag on the command line `handin -c cs-314 A3`.