CPSC 314
Assignment 3: Shading, lighting and Textures

Due 11:59pm, Nov 6th, 2015

1 Introduction

The goal of this assignment is to explore lighting and shading of 3D models. You will be simulating four different styles of shading : Gouraud, Phong , Blinn-Phong and texturing. In this assignment you will be *adding* new pairs of shaders (both vertex and fragment shaders) to carry out these tasks. You will be writing a bit more code, but the shaders will be very similar to previous assignments.

2 Template

The template code is found in the main assignment directory. It is very similar to the previous template for assignment 1. You will need to edit the code and create some new shaders for this assignment. We provide you with default geomtery for all the objects you will need to shade, and have already defined the properties of a basic directional light in the scene (defined in the lightColor, lightDirection and ambientColor variables). We have also defined suggested material properties for the objects you will be shading (defined in kSpecular, kDiffuse and kAmbient). You will be responsible for updating the various shaders to implement the desired lighting schemes.

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 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. We have created template shaders similar to previous assignments for you to work in. For the texturing question we have used the three.js function THREE.ImageUtils.loadTexture to load the texture but you must pass it to the shaders and look up the texture color yourself.

4 Work to be done (100 pts)

(15 pts) Gouraud Shading In this part you will start with a simple and fast lighting model. Gouraud shading calculates the lighting of an object at each vertex (e.g. in the vertex shader). The standard graphics pipeline will then interpolate the color of the object between the vertices to get the individual fragment colors. An example of the result (taken from wikipedia) is shown below, note how models tend to look blocky or pixelated if they do not have a high polygon count. The calculation for the lighting at a vertex should be based on the standard Phong reflection model discussed in class. You can play around with the various components (diffuse, specular and ambient) to produce different effects and materials.

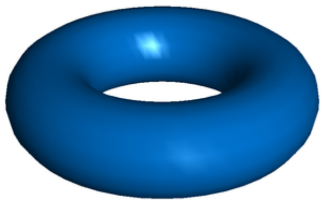

Figure 1: Gouraud shaded torus with phong reflection model.

(15 pts) Phong Reflection and Phong Shading. In the previous section a simplified lighting model was used to calculate the color of the object at any given vertex. This is fast and efficient but doesn't look very appealing or smooth. In this section you should implement the full Phong *reflection* model by passing the lighting parameters (such as the triangle normal and vertex position) as varying variables to the fragment shader. Then, you should calculate the lighting of the mesh at each fragment using the interpolated lioghting parameters. This allows the shading model to better approximate curved surfaces and produces a very smooth shading model. The following image, taken from the Wikipedia article on the Phong reflection model, shows how the different components look individually and summed together:

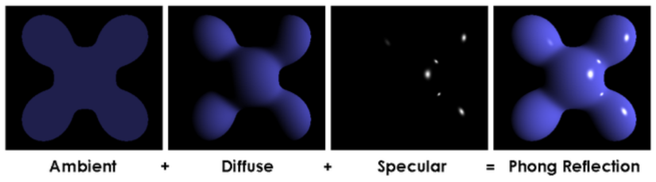

Figure 2: Phong Reflection Model

The main calculations should all be done in the fragment shader (in phong.fs.glsl). Note that you will still need a vertex shader (in phong.vs.glsl), to pass the appropriate information to your fragment shader.

(15 pts) Blinn-Phong Shading 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. This is the essence of the Blinn-phong shading model. Implement a shader which simulates the Blinn-Phong shading model. The textbook or lecture slides has more detail on this subject.

(25 pts) Basic Texturing A modelled object often has many very small details and facets of the surface (e.g. the grain of wood on a box, scratches on metal, freckles on skin). These are very difficult if not impossible to model as a single material lit by a phong like model. In order to efficiently simulate these materials we usually resort to texturing. In basic texturing, UV coordinates are taken from the vertex buffer (three.js provides them for free on default objects such as the sphere in our assignment). UV coordinates are a 2D index into an image file whoose RGB values can be used to help color the mesh.

While you can mix phong and texturing in practice, in this assignment we will ask you to use the texture data as the final color of the fragment. We provide you with a standard texture file called "gravel-rocks-texture.jpg" that you must apply to the fourth sphere using a custom shader. It is loaded into the variable "sphereTexture" in a3.js This should be done in the fragment shader. You are encouraged but not required to account for perspective distortion in the texture.

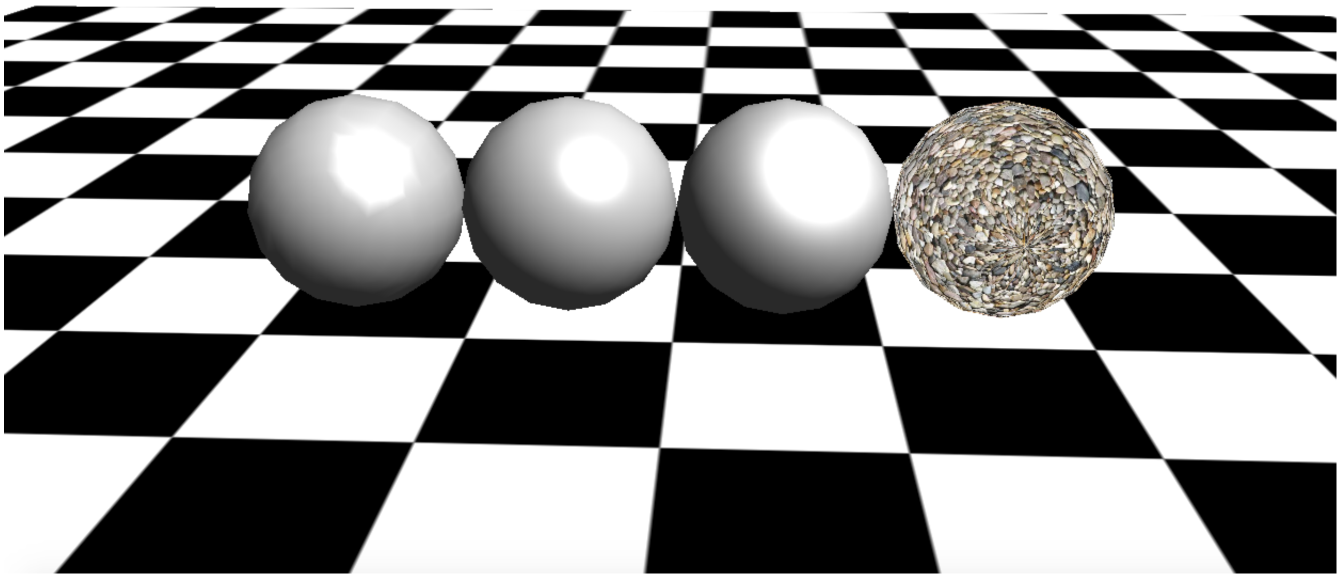

Figure 3: Example results for goraud, phong, blinn-phong and texturing, shown from left to right.

(30 pts) Creative License. This part is up to you. You should create a new shader or modify one of the above shaders to implement some interesting lighting, shading or texturing effects similar to the complexity of the ones above. Some recommendations are:

- Implement anisotropic material, such as brushed metal.
- Add spot lights and directional lights, perhaps animate them.
- Simulate fog.
- Implement normal mapping
- Implement a scheme to compute UV coordinates for objects without them (e.g. the armadillo)
- Implement a cube map to create a reflective object

Bonus marks may be given at the discretion of the marker for particularly noteworthy explorations.

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 login ID, and any information you would like to pass on to the marker. Create a folder called "a3" under your "cs314" directory. Put all the source files, your makefile, and your README.txt file for each part in the folder. Do not use further sub-directories. The assignment should be handed in with the exact command:

```
handin cs314 a3
```