CPSC 314
Assignment 3: Shading

Due 11:59PM, March 12th, 2021

1 Introduction

In this Assignment, you will utilize your knowledge of lighting and shading on 3D models. Here we will study how to implement various shading algorithms: Phong, Blinn-Phong, Anisotropic, Toon, and a fun "rolling stripes" shader. This is a rather interesting assignment, so we hope you will have fun with this one.

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/cpsc314-2020w-t2/a3-release.git
```

1.2 Template

- The file `A3.html` is the launcher of the assignment. Open it in your preferred browser to run the assignment, to get started.
- The file `A3.js` contains the JavaScript code used to set up the scene and the rendering environment. You will do most of your work here for this assignment.
- The folder `glsl` contains the vertex and fragment shaders for the sphere and the five different shading algorithms.
- The folder `js` contains the required JavaScript libraries. You do not need to change anything here.
- The folder `obj` contains the geometric models loaded in the scene.

2 Work to be done (100 points)

First, ensure that you can run the template code in your browser. See instructions in Assignment 1. The initial scene should look as in Figure 1. Study the template to get a sense of what and how values are passed to each shader file. There are five scenes, each corresponding to a different shader, you may toggle between them using the number keys 1, 2, 3, 4, and 5 on your keyboard: 1 - Phong, 2 - Blinn-Phong, 3 - Anisotropic, 4 - Toon, and 5 - Rolling stripes.

The default scene is set to 1. See `let mode = shaders.PHONG.key;` in `A3.js`. You may find it convenient during your development to change this default value to the scene containing the shader that you are currently working on (e.g. `let mode = shaders.ANISOTROPIC.key;` for question 1c).

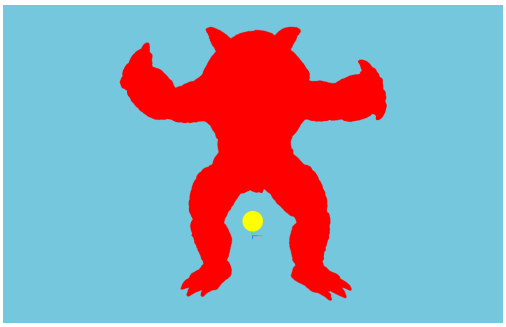

Figure 1: Initial configuration

1. **Part 1: Required Features**

   (a) **(15 points)** Scene 1: Phong Reflection.
   First of all, note that the Phong reflection model is a different type of thing than the Phong shading model; they just happen to be named after the same person. The latter improves on the Gouraud shading by computing the lighting per fragment, rather than per vertex. This is done by using the interpolated values of the fragment's position and normal. We'll be using Phong shading throughout this assignment. In this scene, you'll implement the Phong *reflection* model. The following image, taken from the Wikipedia article on this model, shows how the different components look individually and summed together:

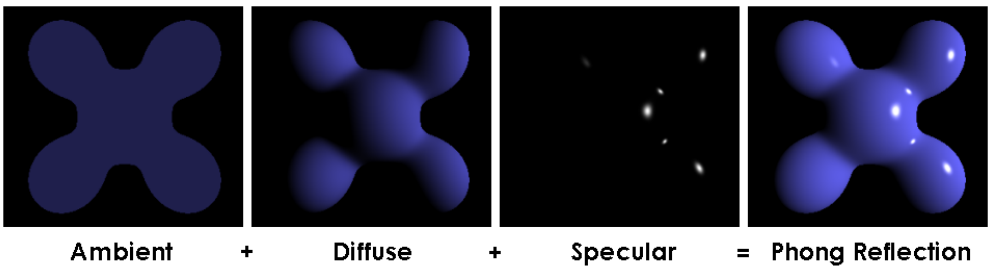

Figure 2: Phong reflection model.

Your task here is to complete the code in `phong.vs.glsl` and `phong.fs.glsl` to shade the Armadillo in Scene 1 using the Phong reflection algorithm. The main calculations should all go in the fragment shader, but you will still need a vertex shader to pass the appropriate information to your fragment shader. Your resulting armadillo should look something like Figure 3.

Hint 1: Remember that dot products should be clamped between 0 and 1, since we don't want to see reflections from the wrong side of the object.

Figure 3: Question 1a). Phong armadillo.

(b) **(15 points)** Scene 2: Blinn-Phong Reflection.
The Blinn-Phong model is similar to the Phong model, with one key difference in how specular reflection is computed. In Phong reflection, specular reflection is computed via the dot product between the reflection vector and the view vector. In Blinn-Phong shading, this is replaced with a dot product between the surface normal and the halfway vector of the light direction and the view vector. Complete the implementation in `blinnphong.vs.glsl` and `blinnphong.fs.glsl` to apply the Blinn-Phong reflection model, per fragment, to the armadillo in Scene 2. Your armadillo should look like Figure 4. Notice that the light looks more spread out here than with the Phong reflection.

Figure 4: Question 1b). Blinn-Phong armadillo.

(c) **(25 points)** Scene 3: Anisotropic.

The Heidrich-Seidel's Anisotropic distribution model is a simple anisotropic model based on the Phong model (see `https://en.wikipedia.org/wiki/Specular_highlight#Heidrich%E2%80%93Seidel_anisotropic_distribution`).

Your task is to shade the Armadillo in scene 3 using this model, as in Figure 5. In anisotropic shading, light scatters in a preferred tangent direction. You can see this effect in common real-world materials such as hair/fur, brushed metal (e.g. stainless steel cooking pots), or certain 3D printed materials. We define the preferred direction as the uniform value `tanDirection`. Complete the implementation in `anisotropic.vs.glsl` and `anisotropic.fs.glsl`.

Hint 1: This shading model is similar to Phong reflection: it uses the same three components. However, the diffuse and specular components are computed differently (refer to the link above).

Hint 2: Highlights should not affect vertices that the light cannot reach — pay attention to the "conditions" section specified in the Wikipedia article.

Figure 5: Question 1b). Anisotropic armadillo.

(d) **(25 points)** Scene 4: Toon.
Send the armadillo to the realm of action and superheros! Unlike the smooth, realistic shading in the previous questions, Toon shading gives a non-photorealistic result. It 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 two tones to colour the armadillo; red for the darker areas (lower light intensity) and yellow for the lighter areas (higher light intensity), as seen in Figure 6. This is most easily done by interpolating between two predefined colours. Lastly, draw dark silhouette outlines on the armadillo. You can use the cosine of the normal and the viewing direction to compute whether the fragment should be an outline: fragments that are "edgy" enough should be outlines, and recall how you can obtain the cosine for two vectors.

If you need some inspiration, the following movies and video games were rendered with toon shading (also called cell shading) techniques:
`http://en.wikipedia.org/wiki/List_of_cel-shaded_video_games`

Hint 1: Use the surface normal and the viewing direction of a fragment to determine whether it is on the edge of the armadillo to draw the silhouette.

Hint 2: Since the silhouette is determined using the surface normal and the viewing direction, and does not depend the light direction, moving the light source will not affect its location.

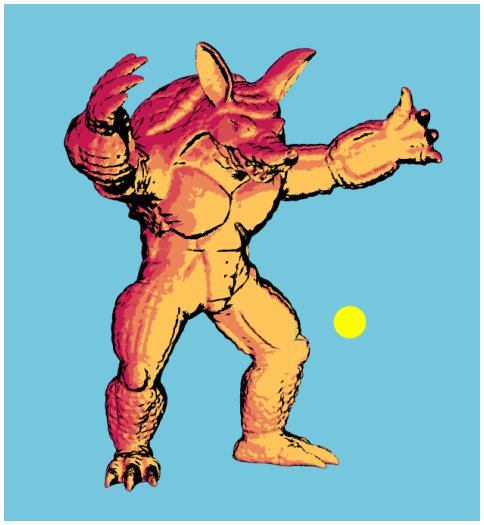

Figure 6: Question 1d). Toon armadillo

(e) **(20 points)** Scene 5: Rolling Stripes.
The possibilities are endless! Here, we will give the armadillo horizontal stripes that smoothly "roll" down the body over time, as in Figure 7. To do so, you will need to write a logical function in the fragment shader that shades the fragment depending on a time component `ticks` and the local vertex position. Complete the code in `stripe.vs.glsl` and `stripe.fs.glsl`. Notice the empty space between the stripes.

Hint 1: What mathematical function oscillates values between negative and positive values?

Hint 2: The `discard` statement in GLSL throws away the current fragment, so it is not rendered.

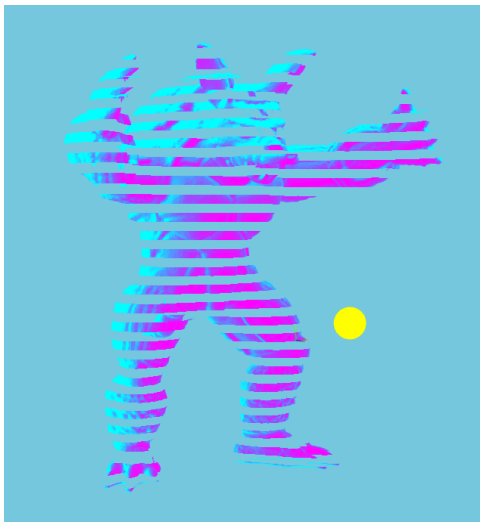

Figure 7: Question 1e). Rolling stripes

2. **Part 2: Creative License (Optional)**

   You have many opportunities to unleash your creativity in computer graphics! In this **optional** section, and you are invited to extend the assignment in fun and creative ways. We'll highlight some of the best work in class. A number of exceptional contributions may be awarded bonus points. Some possible suggestions might be:

   - Explore other non-photo realistic shaders, such as Gooch shading for technical illustration.
   - Implement a fun, new, and creative shader. Try making it procedural so that the shading changes over time, or spatial so that the shading depends on the vertex position!
   - Animate the objects in toon shader to create an interesting demo.

2.1 Hand-in Instructions

You must write a clear `README.txt` file that includes your name, student number, and CWL username, instructions on how to use the program (keyboard actions, etc.) and any information you would like to pass on to the marker. Create a folder called "a3" inside your "cs-314" directory. Within this directory have two subdirectories named "part1," and "part2", and put all the source files and your `README.txt` file for each part in their respective folder. Do not use further sub-directories. The assignment should be handed in with the exact command:

```
handin cs-314 a3
```

on a department computer, which you can SSH into.
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`.