CPSC 314
Assignment 3: Shaders

Due 11:59PM, November 9th, 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, Toon, PBR (Physically Based Rendering) as well as some other fun shaders. 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/cpsc314-2021w-t1/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 assigment, to get started.
- The file `A3.js` contains the JavaScript code used to set up the scene and the rendering environment. You will need to make minor in it to answer the questions.
- The folder `glsl` contains the vertex and fragment shaders for the armadillo and other geometry. This is where you will do the rest of your coding.
- 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.
- The folder `images` contains the texture images used.

1.3 Execution

As mentioned above, the assignment can be run by opening the file `A3.html` in any modern browser. However, most browsers will prevent pages from accessing local files on your computer. If you simply open `A3.html`, you may get a black screen and an error message on the console similar to this:

```
XMLHttpRequest cannot load... Cross origin requests are
  only supported for protocol schemes: http, data, https.
```

Please see this web page for options on how to run things locally:

https://threejs.org/docs/#manual/en/introduction/How-to-run-things-locally

We highly recommend that you run a local server, instead of changing browser security settings.

1. Follow the link https://nodejs.org/en/ to download and install Node.js, which comes packaged with npm.
2. Open the link https://www.npmjs.com/package/http-server and follow the instructions to download and install a local command-line http server.
3. Go to the command-line or terminal and run `http-server [path]` where [path] is the path to the assignment folder.
4. Open your preferred browser and copy and paste the URL of the local server specified by the http-server on your command-line.

2 Work to be done (100 pts)

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 four scenes, each corresponding to a different shader, you may toggle between them using the number keys 1, 2, 3, and 4 on your keyboard: 1 - Phong, 2 - Toon, 3 - Polka dots, 4 - Three.JS PBR.

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.TOON.key; for question 1b).

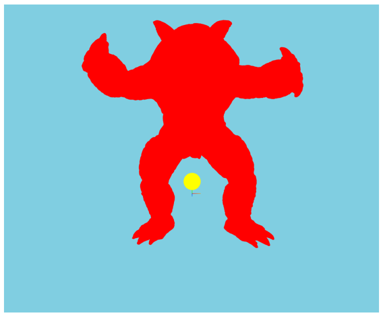
Figure 1: Initial configuration

Part 1: Required Features

(a) **25 pts** 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 except for part 1 (d). 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:

Hint 1: This part can be done entirely in `phong.fs.glsl` and `phong.vs.glsl`.

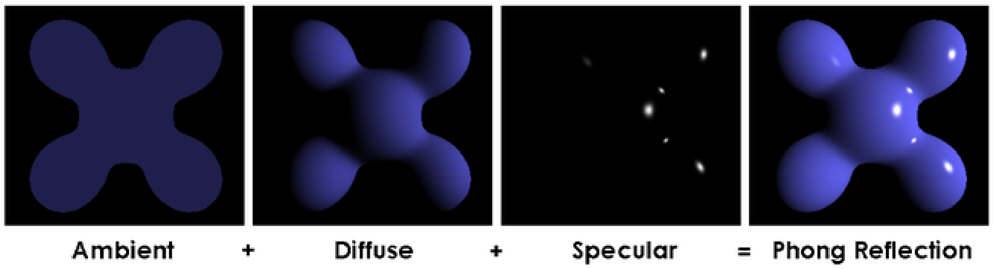

Figure 2: Phong reflection model.

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.

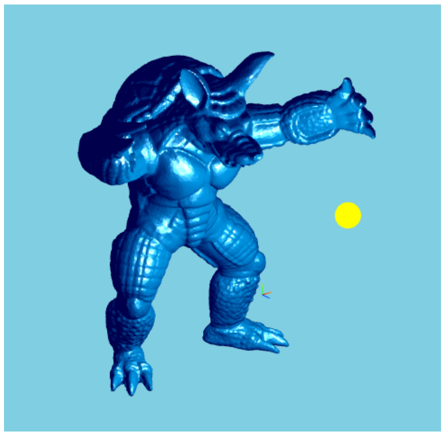

Figure 3: Question 1a). Phong armadillo

(b) **25 pts** Scene 2: 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 4.

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 silhouette of the armadillo.

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.

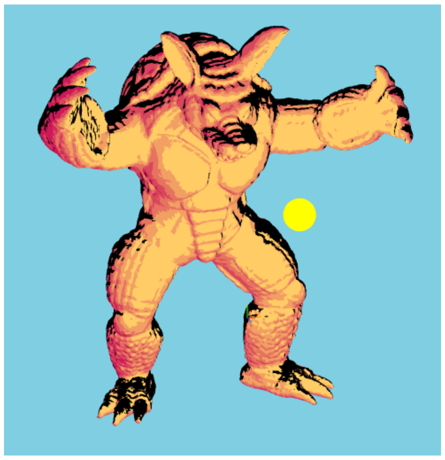

Figure 4: Question 1b). Toon armadillo

(c) **25 pts** Scene 3: Polkadots

The possibilities are endless! Here, we will (i) give the armadillo polka dots by checking whether a fragment is close enough to points on a regular grid in the 3D space around the armadillo; (ii) make the polka dots smoothly "roll" down the body over time, as in Figure 5, and (iii) make the dots smoothly change colour. To do so, you will need to write a function in the fragment shader that shades the fragment depending on a time "ticks" and the local vertex position, discarding fragments that you don't want to render. Notice the empty space between the dots. Complete the code in dots.vs.glsl and dots.fs.glsl.

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.

Hint 3: : The armadillo's model coordinates may be smaller than you think. If your armadillo disappears completely (not because of some syntax error) try using smaller numbers when computing regions to discard.

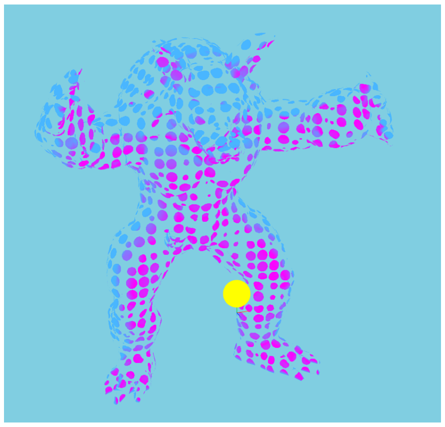

Figure 5: Question 1c). Polka-dotted armadillo

(d) **25 pts** Scene 4: Three.JS PBR and texturing

In this part, we'll get our first look at how to texture map an object using Three.JS and to use textures for PBR.

A modeled 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 use texture mapping. In basic texture mapping, UV coordinates are stored as vertex attributes in the vertex buffer (Three.js provides them in the `vec2 uv` attribute for default geometries such as planes and spheres). UV coordinates allow you to look up sampled data, such as colors or normals, stored in a texture image, as discussed in class. Figure 6 shows the textures we will use for rendering the chest.

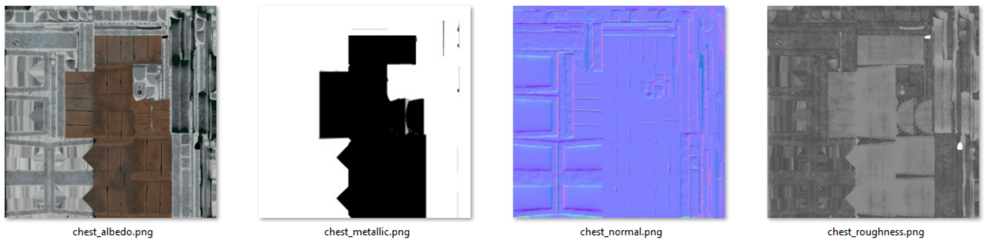

Figure 6: Textures used for the rendering the chest object .

PBR or Physically Based Rendering is a technique that combines different lighting phenomena to render surfaces more realistically. For PBR, each texture map contains different information about how a surface should interact with light. For more information, a great writeup can be found here: https://learnopengl.com/PBR/Theory.

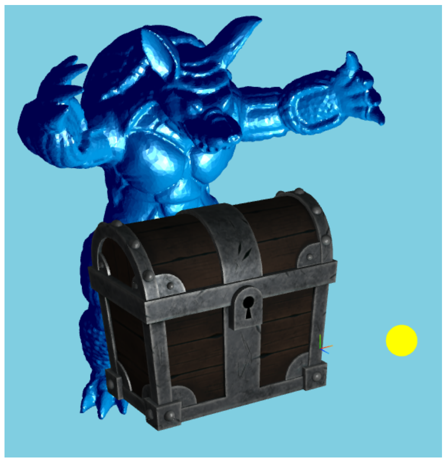

Figure 7: Question 1(d) PBR with `MeshStandardMaterial`

We will use Three.JS's built-in `MeshStandardMaterial` in this part. Edit `chestMaterialPbr` in `A3.js` to provide the material with the material textures available in the `texture` folder.

Textures in GLSL are specified as `sampler2D` uniforms, and the values can be looked up using the `texture()` function. Three.js built-in materials do this for you. We load the relevant textures using THREE.TextureLoader. You can consult the documentation for more information.

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 small number of exceptional contributions may be awarded bonus points. Some possible suggestions might be:

- Create a new interesting shader for the armadillo or chest. E.g., Make the armadillo's skin out of an anisotropic material, like brushed metal. e.g., see using Heidrich-Seidel Anisotropic distribution model
`https://en.wikipedia.org/wiki/Specular_highlight#Heidrich%E2%80%93Seidel_anisotropic_distribution`
- Add a new object that uses texture mapping.
- Generate new animated effects using the `discard` function in GLSL.
- Implement some aspect of PBR in your own shader.

3 Hand-in Instructions

3.1 Directory Structure

Under the root directory of your assignment, create two subdirectories named "part1" and "part2", and put all the source files, your makefile, and everything else required to run each part in the respective folder. Do not create more sub-directories than the ones already provided.
You must also 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. Place README.txt under the root directory of your assigment.

3.2 Submission Methods

Please compress everything under the root directory of your assignment into `a3.zip` and submit it on Canvas. You can make multiple submissions, but we will grade only the last one.