CSC418/2504 Computer Graphics Winter 2018: Assignment 2

15% of course grade
Due 11:59PM on February 26, 2018

Part B (60 marks)

In this assignment you will be writing several different shaders in a C-like language known as the OpenGL Shading Language, or GLSL for short. A shader is program that decides the colour of each pixel rendered for an object.

Your Programming Task. Your task is to implement the following shaders. Each type of shader will be discussed in either class or tutorial to provide you with what you need to start. Figure 1 shows how the results should look like.

a) [10 marks] Ambient, diffuse, and specular lighting using the Gouraud shading model.
`shaders/gouraud_vert.glsl, shaders/gouraud_frag.glsl`

b) [10 marks] Ambient, diffuse, and specular lighting using the Phong shading model.
`shaders/phong_vert.glsl, shaders/phong_frag.glsl`

c) [10 marks] Ambient, diffuse, and specular lighting using the *cel shading* model. *Cel shading* is not a well-defined term by itself, and may refer to various kinds of shading techniques emulating cartoons. You'll implement the most basic form of cel shading by giving a *posterized* look to the object.
`shaders/cel_vert.glsl, shaders/cel_frag.glsl`

d) [10 marks] *UV* texture mapping. With a given texture image and a model with defined texture coordinates, you need to write the shaders to display a textured model. Note that the provided texture will only look reasonable on the *sphere* model.
`shaders/texture_vert.glsl, shaders/texture_frag.glsl`

e) [10 marks] Halftone shading. The halftone technique simulates continuous tone using a grid of dots that vary in size to depict different tones.
`shaders/halftone_vert.glsl, shaders/halftone_frag.glsl`

f) [10 marks] X-Ray shading. The opacity of the rendered fragment decreases as you move further away from the edge of the object's silhouette, producing results that look like X-Ray images.
`shaders/xray_vert.glsl, shaders/xray_frag.glsl`

g) **Bonus** [10 marks] For bonus points, you have to implement one of the following shading/mapping techniques. Points will be awarded on the basis of technical competence, creativity, ingenuity, and aesthetic appeal of the result.

— Cook-Torrance reflectance model. The Phong model covered in class is not the only reflection model employed in computer graphics. The Cook-Torrance model is another common example.

— Hatching. Hatching is commonly used in pencil, charcoal, and pen and ink sketches to create tonal and shading effects. In your shader program, you can approximate tone by varying the density of *linear* hatching and/or the number of lines used for *crosshatching*. You can also try hatching in general contours, instead of straight lines.

— Painterly rendering. Implement a painterly rendering technique of your choice. You can try to mimic impressionist brush strokes, water colours, knife painting techniques, coloured pencils, etc.

```
shaders/bonus_vert.glsl, shaders/bonus_frag.glsl
```

Helper Code. You are provided with an environment for developing shaders that runs in a modern web browser. Start by opening up a terminal window, and navigating to the folder `A2`. Then, type out the following command

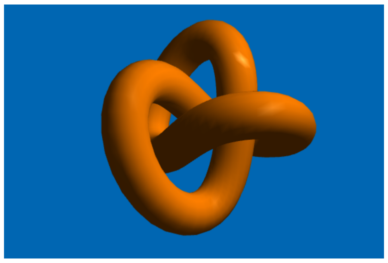
```
$ python server2.py
```

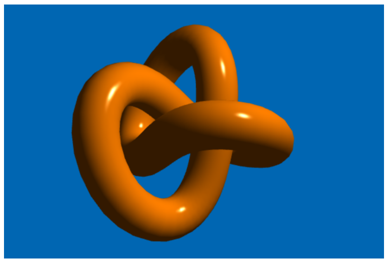
assuming the default python installation is python2 (this is true for CDF machines), or

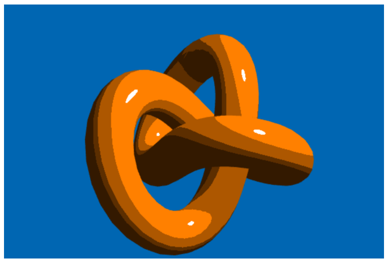
```
$ python server3.py
```

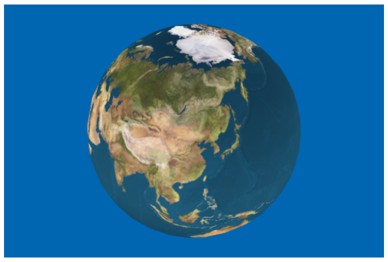
if the default python on your machine is python3.

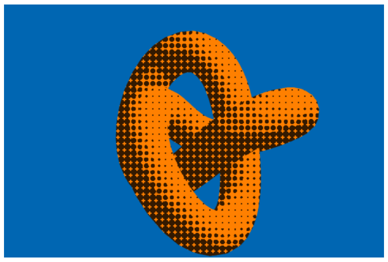
In any modern browser, then navigate to `http://localhost:8000`. You should see text boxes displaying the code of the currently selected shader, a viewer window, and UI tools to select the shader, material colours, light position, and the displayed shape (see Figure 2). The instructions are guaranteed to work on CDF machines. However, having Python and a modern browser (latest Chrome, Firefox, Edge, etc.) installed are the only requirements if you choose to run it on your own machine.

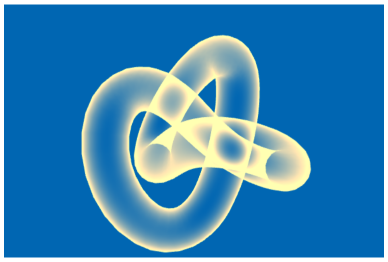

(a) Gouraud shading


(b) Phong shading


(c) *Cel* shading


(d) *UV* texture mapping


(e) Halftone shading


(f) X-ray shading

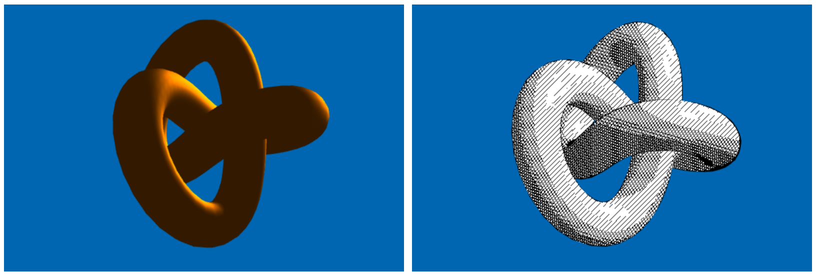
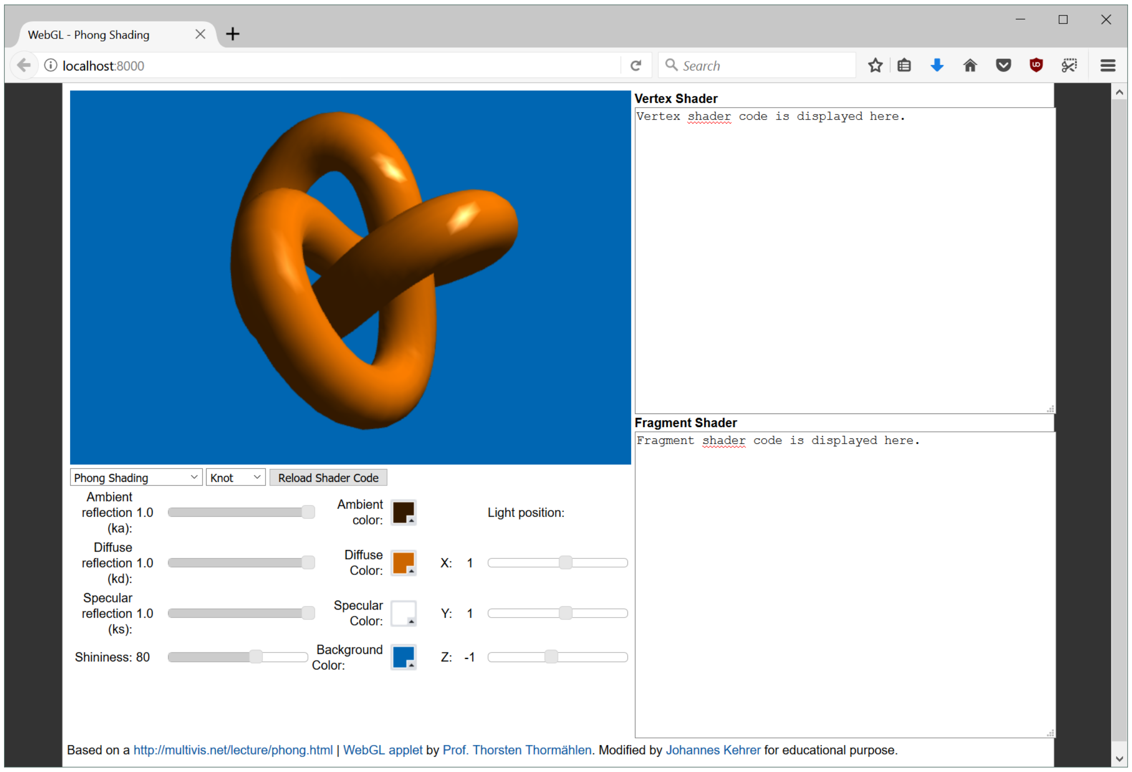

(g) Bonus shaders: Cook-Torrance model and cross-hatching

Figure 1: Sample results for the programming tasks. Note that your bonus shaders may look wildly different from the examples shown here and may still be deemed correct if you follow a reasonable algorithm.



Vertex shader code is displayed here.

Fragment shader code is displayed here.

Ambient reflection (ka): 1.0
Diffuse reflection (kd): 1.0
Specular reflection (ks): 1.0
Shininess: 80
Ambient color:
Diffuse Color:
Specular Color:
Background Color:
X: 1
Y: 1
Z: -1

Based on a http://multivis.net/lecture/phong.html | WebGL applet by Prof. Thorsten Thormählen. Modified by Johannes Kehrer for educational purpose.

Figure 2: Snapshot of the display window.

Note for Windows: The same commands will work for command prompt or PowerShell on Windows computers. Make sure that you've added the python installation folder to the system `PATH`.

You may edit the shaders by modifying the source code found in the `shaders/` directory. After you've made a change to a shader file, press the **Reload Shader Code** button to load and run the new code.

Note: You should not modify any of the provided code other than the files in the `shaders/` directory.

Turning in your Solution to Parts A and B. Your PDF file for part A and your submission for part B should go inside the same folder named `A2` in a `tgz` file. All your code should remain in the directory `A2/shaders`. Note that this file should not be thought of as a substitute for putting detailed comments in your code. Your code should be well commented if you want to receive full (or even partial) credit for it.

To pack and submit your solution, execute the following commands from the directory containing your code (i.e., `A2/shaders`).

```
$ cd ../../
$ tar cvfz a2-solution.tgz A2
```

Submit your assignment using the submission script on the course website.