COMS W4160— Computer Graphics Spring 2013

Programming Assignment 3

out: Mar 18, Monday
due: Apr 2, Tuesday midnight

After a relatively hard PA2, this should be an easy programming assignment. In this assignment, you will apply what you learned about graphics pipeline and implement several types of shading models using GLSL shading language. In particular, you need to implement the vertex and fragment processing stages to achieve a few different shading effects. Shader programming has been widely used in high-performance interactive graphics applications such as video games.

1 Principle of Operation

As discussed in class, the *graphics pipeline* is a sequence of processing stages that efficiently transforms a set of 3D *primitives* into a shaded rendering from a particular camera. The major stages of the pipeline include:

1. **Application** holds the scene being rendered in some appropriate data structure, and sends a series of primitives (only triangles, in our case) to the pipeline for rendering.
2. **Vertex processing** transforms the primitives into screen space, optionally doing other processing, such as lighting, along the way. The specific operations can be customized by vertex shaders.
3. **Rasterization** takes the screen-space triangles resulting from vertex processing and generates a fragment for every pixel that's covered by each triangle. Also interpolates parameter values, such as colors, normals, and texture coordinates, given by the vertex processing stage to create smoothly varying parameter values for the fragments. Depending on the design of the rasterizer, it may clip the primitives to the view volume.
4. **Fragment processing** processes the fragments to determine the final color for each, to perform z-buffering for hidden surface removal and to write the results into the *frame buffer*.
5. **Display** shows the contents of the frame buffer so the user can see them.

The detailed functionality of both vertex processing and fragment processing stages can be customized by providing the program your shader code. In this assignment, you are required to write your own shaders to satisfy the assignment requirements.

2 Getting Started

To help you start writing GLSL shader code, we prepared a starter code (see Figure 1) which illustrates a simple non-photorealistic cartoon-style shading effect. Please see the Appendix for the instruction of compiling the starter code. Because of the powerfulness and popularity of GLSL shading language, there are numerous online resources. I listed a few of them here for your reference:

- The Lighthouse3D tutorial gives a detailed introduction of the GLSL basics, and it also contains a few examples.

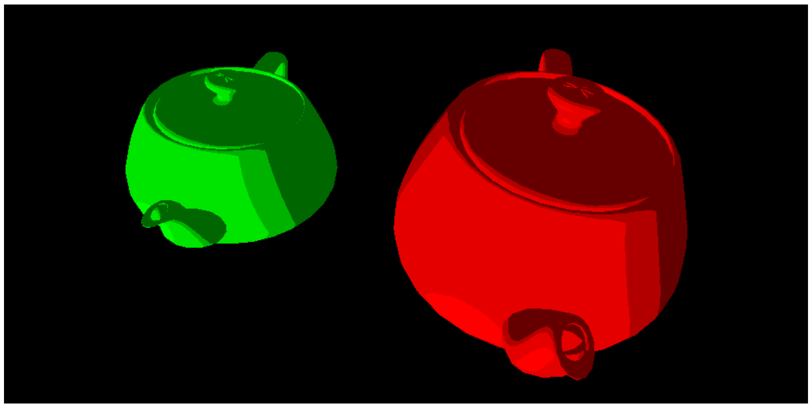

Figure 1: **Shading Effect:** A non-photorealistic cartoon-style effect is achieved using a simple vertex shader and a fragment shader. The graphics pipeline can be customized to display various interesting effects.

- NeHe tutorial
- `http://people.freedesktop.org/~idr/OpenGL_tutorials/`: A short yet illustrative tutorial of vertex and fragment shader.
- If you want to play with both vertex shader and fragment shader quickly, try out KickJS Shader Editor (in google Chrome or Firefox).
- Some nice animation effects can be found at `http://glsl.heroku.com/`, where you can also try and play with the shaders.

2.1 Required Library

The OpenGL Extension Wrangler Library is need to compile and "inject" the GLSL shader program into the graphics pipeline. The CLIC machines have already installed the library. On Ubuntu Linux, it can be installed easily by

```
sudo apt-get install libglew-dev
```

On MacOSX, you can install it using Mac Port (if you installed) by running the command

```
sudo port install glew
```

Otherwise, you can install it by downloading and compiling the glew source code from its Website.

2.2 OpenGL 2 v.s. OpenGL 3+

The specification of the GLSL shading language changed a little since OpenGL 3. On CLIC machines and in our starter code, it uses OpenGL 2. I suggest you use OpenGL 2 (GLSL 1.20).

3 Programming Requirements

3.1 Required Shaders

You will implement vertex and fragment shaders to provide one *required* shading effects and a *subset* of the following kinds of shading with support for multiple light sources:

1. **Gouraud shader (required):** you are required to implement a basic Gourand shader as discussed in class. The vertex color should be computed using a Phong reflection model also as discussed in class. The color inside a triangle should be interpolated.

2. **Blinn-Phong shading model (optional):** A Blinn-Phong is a simple modification to the Phong reflection model. You can find its details (including a pseudo code) on wikipedia.

3. **Texture-modulated Smooth Shader (optional):** Each triangle is shaded using a color value from either of the above shaders and multiplying the resulting color value by the current texture. The texture coordinate (given by gl_TexCoord[0].st in GLSL) for each vertex are used to index into the texture. The GLSLProgram.h in the starter code provides a method bind_texture to help you load a texture. Please look into the code for details.

4. **Checkerboard texture (optional):** In class, I illustrated a checkerboard pattern as a simple procedural texture. You can implement it in your shader and combine it with your Gourand or Blinn-Phong shading model to show checkboard patterns on object surfaces.

5. **Wireframe texture (optional):** You can implement a shader to highlight the mesh wireframes together with color shading. For example, it could look like a Blinn-Phong shader with the triangle edges highlighted by red color (hint: use the barycentric coordinates to determine how far a point is away from a triangle edge).

6. **Normal map (optional):** Load a image and use its 3 color channels to specify the geometry normals. In your shader, you can use the normals specified by textures to modulate shaded surface colors.

You are required to implement the first requirement above. For the rest 5 optional requirements, you should implement 2 of them. *However*, you *cannot* choose requirement #2 and requirement #3 simultaneously. In other words, you can use one from requirement #2 and #3, and one from requirement #4, #5 and #6; or, you can choose two from #4, #5 and #6.

3.2 Bonus Requirement

You are welcome to implement other cool shading effects. Use your imagination and creativity to develop any cool effects. You can find some effects online such as these pictures for inspiration. Make sure in the report to describe what your shader is doing.

3.3 Other Requirements

You are free to load any 3D models to visualize. However, in order to show the smoothly changing color effects produced by shaders, you probably need to use a smooth 3D geometry. A simple cube won't work. You should also enable a simple user interface which allows the user to change the camera view angle. The focus of this assignment is on shaders. Therefore, you are allowed to reuse your code from previous programming assignment.

4 Submission and FAQ

Submission Checklist: Submit your assignment as a zip file via courseworks. Your submission must consist of the following parts:

1. **Documented code:** Include all of your code and libraries, with usage instructions. Your code should be reasonably documented and be readable/understandable by the TAs. If the TAs are not able to run and use your program, they will be unable to grade it. Try to avoid using obscure packages or OS-dependent libraries, especially for C++ implementations. To ensure the TAs can grade your assignment, it is highly suggested to compile your code on CLIC machines, and include details of compiling and running your code on CLIC.

   In the starter code, we also include a CMakeLists.txt file which is used to automatically detect libraries and generate Makefile using the cmake building system. It is optional to modify the CMake file for your project, while it will probably make the compile more organized and less tedious.

2. **Brief report:** Include a description of what youve attempt, special features youve implemented, and any instructions on how to run/use your program. In compliance with Columbia's Code of Academic Integrity, please include references to any external sources or discussions that were used to achieve your results. *I highly suggest you put a few pictures in your report to illustrate your shading effects.*

3. **Video highlight!:** Include a video that highlights what youve achieved. The video footage should be in a resolution of 960×540, and be no longer than 10 seconds. We will concatenate some of the class videos together to highlight some of your work.

4. **Additional results (optional):** Please include additional information, pictures, videos, that you believe help the TAs understand what you have achieved.

Evaluation: Your work will be evaluated on how well you demonstrate proficiency with the requested shader code, and the quality of the submitted code and documentation, but also largely on how interesting and/or creative your overall effects is.

Please don't copy the shader code from somewhere else. We are going to run MOSS (see also `http://www.ics.uci.edu/~kay/checker.html`*) on your submitted code when we grade it. If there is a large similarity, we will have to ask you to explain.*

A C++ starter code

I attach the instructions of compiling the starter code here. I used CMake to compile the starter code, since I assume you knows CMake now after two assignments. On Ubuntu, you can install cmake by the command

```
sudo apt-get install cmake
```

The following command sequence will make code compiled on both Linux and MacOS.

1. `unzip pa3_starter.zip`
2. `cd pa3_starter && mkdir gcc-build && cd gcc-build`
3. `cmake ..`
4. `make`

Library dependence: Beside GLEW, the starter code depends only on the OpenGL library and GLUT. Both of them have been installed on CLIC machines. Most of the operation systems with graphics interface should have them already. If not, they can be manually installed. On Ubuntu Linux, you can install them by

```
sudo apt-get install libglu1-mesa-dev freeglut3-dev
```