CAP4730 – Homework 2

Due February 24, 2014

1 Graphics Pipeline

`Smooth`[1] is an example of an OpenGL program that loads a mesh (given in a .OBJ file) and renders it with various features. Learn how `smooth` works so that you can modify the code. You can find meshes in .OBJ format from various online sources (when you use sources, properly cite and acknowledge where you obtained them from).

You are to modify the `smooth` program to implement a basic graphics pipeline for rendering triangular meshes. You may use all the functions in `smooth` for loading an .OBJ file and the data structures for maintaining and accessing vertices and faces of a triangular mesh.

1.1 Rasterization

Implement a rasterization algorithm that takes the coordinates of a 2-D triangle and draws the corresponding pixels on the screen. You can use the `glDrawPixels` method for drawing pixels/points.

- Modify the `smooth` program so that it will use your rasterization algorithm to draw simple 2-D triangles that are projected from the loaded 3-D model (look up `gluProject`).
- Use "data/cube.obj" as a starting object to test your implementation.
- Keep the functionality of the `smooth` program; allow the user to switch to your pipeline by pressing the key `'y'`. This key will toggle onto and off of your implementation of the pipeline.

`30pt`

1.2 Z-Buffer and Shading

In this section, you will complete your implementation of the graphics pipeline by incorporating the Z-buffer and Shading steps into the rasterization framework that you developed. The Z-buffer stage eliminates hidden surfaces from the rendered image, and the shading stage improves the visual realism by offering a local illumination which also helps perceiving the depth.

- As you recall, `gluProject` returns the depth values of projected vertices (i.e., z'). To determine the depth values of each pixel (that the scanline algorithm draws), you can simply use barycentric interpolation on the three depth values (that `gluProject` provides) for the projected vertices of each triangle. The interpolated depth values will provide a Z-buffer for each triangle which is being rendered. The final stage of the Z-buffer algorithm can be implemented as you blend each rendered triangle to the final image (i.e., framebuffer); simply overwrite a pixel in the framebuffer only if it is further to the viewer from the new pixel you're trying to draw. `25pt`

[1] `http://www.xmission.com/~nate/smooth.html`

- Implement flat shading for your pipeline. Query each triangle for its normal, and perform the local illumination (ambient, diffuse, specular) per triangle. Simply use the current lighting setup (i.e., direction, intensity,...) that is available in smooth for your local illumination calculations. Use the color of the illuminated triangle to draw the pixels that belong to that triangle. `25pt`

- *Bonus part* Modify your pipeline to include Gouraud shading. In the GPU language, you will implement a *vertex shader* that computes the shading formula per vertex and then interpolates the RGB colors across the triangle. `+20pt`

- Use "data/cube.obj" as a starting object to test your implementation. Once tested, examine your pipeline using more sophisticated models.

Submit a report (PDF document) that, along with images, documents the results for each step of the homework. Moreover, in the report explain any differences between the images rendered in your pipeline and the original pipeline implemented in `smooth`. Also comment on why your implementation is faster than or slower than the original implementation. `20pt`

2 Acknowledgments

If you are using resources for any part of the homework (including, ideas, source code and images), make sure to include proper citation and clear acknowledgments in your report.

Submission Guidelines
Submit a **single file** as a .zip or a .tar.gz bundle that contains all the files to be submitted. Include the source codes for your programs in the submission bundle. Please include a 'README' file that clearly explains how to run and test the program. Also include a 'Makefile' that compiles and links the program from the source files. Please make sure your program compiles/runs in the CISE linux machines.
Late submissions are penalized by 10% of the grade for each day past the due date.