Final Project: 3d Network Visualizer

Michael Yoshimura

CSCI4229/5229 Fall 2019

This is a simple framework that allows customized network rendering. Not a whole lot has changed since the project review, except a few tweaks and a few new examples to show what the visualizer can do.

Setup

This program is built with Glut in Python 3.4+. Requires GLSL 3.30+.

Modules you will probably need to install (via “pip install” or “pip3 install”):

PyOpenGL, PyOpenGL_accelerate, PyGLM, networkx.

Then, just run “python main.py”. You might have to run “python3 main.py” if your machine defaults to python2.

This visualizer has been tested extensively on Windows 10 and briefly on the Fall 2018 CU CS VM. If you do plan on testing on Windows, you might need a specific version of the PyOpenGL and accelerate packages. They are found here: https://www.lfd.uci.edu/~gohlke/pythonlibs/.

I will first outline what has been implemented, and what has been omitted from the original project proposal.

- Firstly, the network only has lines to represent the edges. It became immediately clear that cylinders would make most networks too cluttered and slow down performance.
- The flow animation shows the flow by having the lines have a moving striped pattern.
- Object Selection
    - For first person perspective, any node in the crosshair in the center of the screen will light up and the name of the node will show up on the bottom left of the screen. I didn't go any farther with UI because opengl and glut aren't designed for good UI.
    - For overhead, any node under the mouse will be highlighted
- The translucent surface to wrap around was too complicated for me to implement correctly (I think it requires convex hull algorithms)
- I created a material framework that allows me to specify the diffuse, specular, and ambient colors of nodes. I also implemented HDR, and a bloom effect that uses a gaussian blur shader.
- The performance is not amazing, but it does fine with networks up to around 1000 nodes.

Here is a brief outline of all the graphics stuff (the stuff that I hope will get me a good grade).

- Bloom post processing and HDR lighting are probably the most advanced features. My main render function (visualizer.py lines 354 - 430) uses three passes. The first pass does the rendering of the network. That gets captured in two duplicate frame buffers. The first is sent directly to the third HDR pass. The second frame buffer goes through the second pass, which

detects bright areas, and applies a gaussian blur to them. The blurred output gets combined with the original image in the third pass.

- The animated striped line is also fairly complicated because it uses a geometry shader to split the single line up into a line with multiple animated segments (See “network_line.vert”, “bloom_line.frag”, and “striped_line.geom”.
- Admittedly, there wasn’t very much advanced normal work in the project. However, since I no longer use the legacy pipeline, I had to implement lighting from the ground up. “spheres.vert” and “spheres.frag” implement Blinn-Phong lighting. The spheres themselves are subdivided icospheres from a previous homework.
    - Lighting only affects a few of the examples. Most networks have node colors bright enough to register as emission/bloom.
- Finding the node in the crosshair / underneath the mouse pointer is done on the cpu, which was much easier because I had access to the nodes in world coordinates. (visualizer.py lines 449 – 485).

How this application is used (if you want to play around with it)

- In main.py, there are a few examples of highlighters that basically specify how a network is rendered. You can specify the colors of every individual edge by passing in an array of tuples where each tuple is an RBGA value for one side of the edge (thus there will be two colors specified for each edge). The nodes colors are specified by an array of Materials (see highlighters.py) that specify the lighting and color of the node.
- The flow animations are automatically detected and applied when a network is directed.
- The light position is currently hardcoded, but the LightHighlighter allows for different colored lighting.

Controls:

Esc Exit the program

n Change the scene to another network

b Change the scene to a previous network

v Change view mode (overhead perspective or first-person perspective) [**Warning: first-person will capture your mouse**]

l Toggle lighting between moving test light or static light

scroll Change FOV

mouse Look around (in first-person perspective only), or hover over nodes to select (in overhead only)

arrows Look around (in overhead) or move (in first-person perspective)

Credit to these tutorials:

https://www.labri.fr/perso/nrougier/python-opengl/#about-this-book

http://pyopengl.sourceforge.net/context/tutorials/shader_1.html

https://paroj.github.io/gltut/index.html

https://schneide.blog/2016/07/15/generating-an-icosphere-in-c/

https://learnopengl.com/Getting-started/OpenGL

https://en.wikibooks.org/wiki/OpenGL_Programming/Glescraft_4

https://en.wikibooks.org/wiki/OpenGL_Programming/Object_selection