Jordan Stein

CSCI 4800-002

5/09/2016

Shaders/GPU Final Project Report

Motivation/Scope:

I decided to implement an interactive model viewer for my final project in CSCI 4800. In the past, I have used model viewers for many different video games, so I thought it would be very interesting to have the opportunity to design and implement my own model viewer using the skills I have gained throughout the class. I knew that the scope of this project would be very suitable for me because the project description stated that it was oriented towards 4800 students who have not taken computer graphics.

Related Work:

I tried relating my model viewer based off the example Dr. Choi provided us with at https://www.youtube.com/watch?v=Ui0PNjnvQPs . I wanted to mimic the model viewer for that 3D Graphics Engine while adding my own design to it.

Main Method:

My main objective was to use the Qt5 graphical user interface to allow the user to easily modify and update the models being viewed in real-time. This included updating the models, textures, lighting, and position. I have successfully implemented all of the major requirements for my model viewer, while adding some additional features (such as viewing surface normal/color mapping).

Below is a list of everything that is functionally implemented:

- Update the model being shown in real-time
- Update the textures applied to the models in real-time
- Show a sample of the texture selected on the GUI
- Toggle the lighting between fixed and rotating
- Alter the speed of the rotating light
- Update the model position (x,y,z)
- Toggle between Phong/Gourand Shading
- View surface normal in real time
- Alter normal scale (length of surface normal lines)
- View color mapping for each model
- Change the background color

The minor issues I ran into was implementing spotlights and allowing the user to import their own shaders into the viewer. I struggled to nicely implement spotlights into the viewer so I had decided to drop that from my requirements. I had also decided to drop the ability to import and use custom shaders because of how complex it is to hard-code an implementation for every custom shader since unknown variables needed to be passed depending on the custom shader.

Implementation:

To implement the model viewer, I begun by working off of the SGPU_HW3_InteractiveParticleSimulation files we had used for homework 3. I deleted everything off of the files pertaining to the particle simulation and edited the form on Qt to better suit a model viewer. Once I had all of my variables passing from the GUI to the QViewport, I was able to begin working on implementing the model viewer.

I first initialized the default scene on the model viewer (sphere with “bark” texture) using the QViewport::initializeGL() function. To do this I simply loaded the default model and default texture inside that function.

I found that QViewport had a function named paintGL() that was consistently looping throughout the duration of the program. I used this function to update and render everything based off the variables passed through the GUI. In my implementation, you will find many conditional statements that update the features to the mesh being rendered. It would first check if the model choice had been updated in the GUI. If it had, it would re-initialize the mesh and load the appropriate model into it. It would then apply the selected texture to the model. Afterword’s, it checks if the user had selected phong shading or not, and loaded the correct shader accordingly. It would then apply lighting to the model. If the user had checked rotating lighting, a rotationLightPhi variable would be incremented and a rotation of the light direction would be calculated and updated in real-time. Otherwise the lighting will be fixed. Finally, the position of the model is set to wherever the user had chosen in the GUI and the model is rendered.

To account for additional features, I would check if the user had selected to view the surface normal or color mapping before I flushed the gl using glFlush(). If the user had selected either of the additional features, different shaders would be applied to the model accordingly before updating the model to the screen.

Conclusion:

I have successfully implemented a model viewer that includes all of the major requirements I had initially planned for. While a few minor requirements were dropped, I had added additional features to compensate. I learned a lot about how to implement and modify shaders throughout programming this assignment. My favorite thing that I had taken away from this assignment was learning how to use and implement the Qt GUI effectively to update the aspects of the model being viewed. I had a lot of fun programming this assignment and I am very pleased with the result.

A video demonstration of my implementation can be viewed at:
https://www.youtube.com/watch?v=1fiY3ucT32Q