Project Report - OpenGL CSG - Owen Hellum

Accomplishments

The project has been very successful in accomplishing what it set out to do. With the half exception of one of the objectives, all goals have been met. Furthermore, a few minor enhancements have been made to make the experience more user-friendly. I was able to create a fully-functional system for performing boolean operations (union, subtraction, intersection, and difference) between 3D models to produce a technique known as Constructive Solid Geometry (*aka* CSG). It takes OBJ files as input (with some that come default with the project), and performs and number of the above operations upon them. Colours, scale, and offset of the objects can also be customized, and they can be rotated and zoomed in on in a 3D space.

Here are the accomplishments regarding the objectives:

Allow the loading and editing of primitive and arbitrary obj files: Any obj file (given that it is a triangular mesh) can easily be loaded into the project. They will function in conjunction with any others.

Implement union, subtraction, difference, and intersection boolean operations of structures: All four of these operations have been implemented to great success.

Properly handle custom shading/texturing of resulting geometry from the above operations: Custom colours can be applied to objects and operations within the scene, however custom texturing and operation texture handling required more time than was possible.

Generate obj files of resulting geometry: I was pleasantly surprised how easy it was to quickly generate resulting OBJ models from my OpenGL operations. The results are flawless and generate quickly.

Allow for primitives and operations to be invoked via the command line: I extended this objective by allowing arbitrary objects to be created and operations to be performed using JSON files. This allows for much more customization and for multiple operations to be invoked at once.

Regarding JSON reading, the following parameters are available:

For geometry:

- `name`: String name of the object (to be used in the operation later on)
- `filename`: String name of the OBJ file to be read from for this object (minus the folder path and `.obj` extension)
- `scale`: Float value to resize the object by
- `offset`: 3D vector to reposition the object by
- `colour`: The RGB colour of the object when shown
- `show`: Whether to show the original object in the 3D scene

For operations:

- `name`: String name of the operation (used when generating the resulting OBJ)
- `model1`: String name of the model to have the operation performed on
- `model2`: String name of the model to perform the operation with
- `operation`: String for type of operation to perform (`+` = union, `-` = subtraction, `|` = intersection, and `/` = difference)
- `colour`: The RGB colour of the operation when it generates
- `save`: Whether to save the resulting geometry to a new OBJ file

Failures

Due to time restrictions, a few extra goals could not be accomplished. These included:

- Generating operations that consisted of more than 2 objects at a time
- Integrating a UI for real-time customization
- A script to automatically subdivide models for more fine-tuned results
- Custom texturing as well as colouring
- Speed improvements when performing operations on models with high vertex counts (though some initial speed-up measures have already been implemented)

In particular one performance issue does stand out. Due to the large size of the `capsule03` Lab capsule that the project was built upon, operations that require significant time to calculate will often result in strange vertex-based artifacts in the 3D viewer. However, the resulting saved OBJ files do not include these errors. Thus, for certain cases this bug is not terribly serious.

Difficulties

The main difficulty faced when getting proper CSG working was being able to check if vertices were contained within meshes of arbitrary geometry. I used a very elegant solution, which comprised shooting rays in random directions from the point being checked towards the inside of the mesh being checked. All mesh hits along those trajectories were then checked. Rays with an odd number of hits were deemed to be inside the mesh, and those with even hits were outside (0 is even). Using this method, I could find out which vertices of certain objects were inside of others, and perform my operations. It took a long time to get working effectively, but it functions flawlessly now (outside of calculation time of course). Other tasks like implementing JSON reading and OBJ exporting also presented a few challenges, but were ultimately fruitful, and are integral parts to the project's functioning.