Matthew Gillatt
CS464
12/18/2015

Final Project Complete Report

The planned features up to but before "draw polygons from closest to farthest without overdraw" have been implemented successfully. This program loads a list of wall segments from a separate map script, automatically constructs a BSP tree using it, and then draws the polygons from farthest to nearest by traversing the BSP tree instead of using WebGL's automatic depth testing. Back-face culling is easily enabled by setting a variable to 1.

Segments may have a color defined, and are drawn using Phong shading. Textures can be enabled, but would stretch over the entire segment. The segment splitting done by the tree builder will make the textures begin and end at strange places. Phong shading without textures makes the segmentation seamless. Textured walls are not a finished feature in this final submission. This would require changing the Segment object to include a texture offset. Segment splitting during the tree building would need to compute new offsets. Also, a Segment would need to remember its texture name if I add support for multiple textures in one map. I would like to draw the wall textures in a tiled format instead of stretching over the whole segment, so this would require clever manipulation of s and t in the fragment shader.

There are other features that wouldn't be difficult to add given more days to work on the project. A second shader could be used to ignore lighting on the floor and ceiling, so that the "flashlight" effect wouldn't make their color so dark. On the 2D graphics layer, I could try drawing a mini-map, using the 2D Segment objects I already have. Even DOOM did this automatically! Also, moving objects inside the map would take some serious refactoring (to be described below), but static Things that decorate the map could be possible, by making a very short kind of segment that's always drawn facing the camera, and using a sprite with alpha instead of drawing a wall from floor to ceiling.

From the way I understand it, DOOM does not use trees that put single segments into nodes.

Rather, the nodes of the tree contain subsectors, each with a list of subsegments along its perimeter and list of Things currently inside it. Having a node contain a list of segments easily fixes the case of collinear segments during tree building, though in my code, I consider a collinear segment to be behind, as it typically doesn't matter which order the two are drawn. Even though DOOM has special tricks to draw map geometry from front-to-back, the sprites are actually drawn from back-to-front like the Painter's algorithm. This means that enormous levels can be handles easily, but the system will have problems if there are too many visible sprites!

I haven't tested my code against WebGL's automatic depth testing yet. It's possible that GPUs are still efficient enough to outperform my linear tree traversal. I'm certain my code would be better if I used front-to-back drawing without overdraw, but the only code examples I saw require ray tracing! I believe this is beyond the scope of this project. A test could be constructed by making enormous amounts of segments using loops. The order they're generated will change how the tree is built, though.

I'm happy with the data structure I've made, including the segment objects. I believe that level1.js was constructed in less than an hour, much of which was me sketching the points on paper before typing them in! This could have taken all day if I were setting up all the attribute variables manually. Also, after overcoming the nuances of Javascript and WebGL, I genuinely had fun creating binary trees! I had to look up how to compute intersection points and other geometric functions, but visualizing how the nodes will be arranged and sorted was one of the most fun parts of the project. Perhaps this is how John Carmack felt when he designed his game engines?

The biggest obstacle for this project was my other class final project being procrastinated on heavily by team members. I had nearly no time to continue this project after the presentation until Thursday afternoon! Thankfully, the only part left which I felt was mandatory was the BSP tree builder, which was easier than I thought and fun to make. I could easily see myself adding new features to this project in the future if a future employer were interested in seeing what else I could do with this!

I really enjoyed working on this project, though I wish we had more experience with Javascript before getting to this. Most of the time where I needed to look up some functions, I would search for WebGL help, not knowing I was working on a Javascript component instead. Also, I did all my code in Wordpad. Debugging would have been much smoother for all of my projects if editing software was recommended at the start of the course. I'm okay with doing things “the hard way” if I enjoy it, though. I play DOOM on “Ultraviolence” difficulty, for crying out loud!