Benjamin Northrop

8/13/24

CS5310

Summer 24

\*\*\*\*The PDF version does not support animated gifs, so you’ll only see a single image. The word document (same file name) has the animated gif for reference. These images are annotated with (gif) in the title\*\*\*\*

**Introduction**

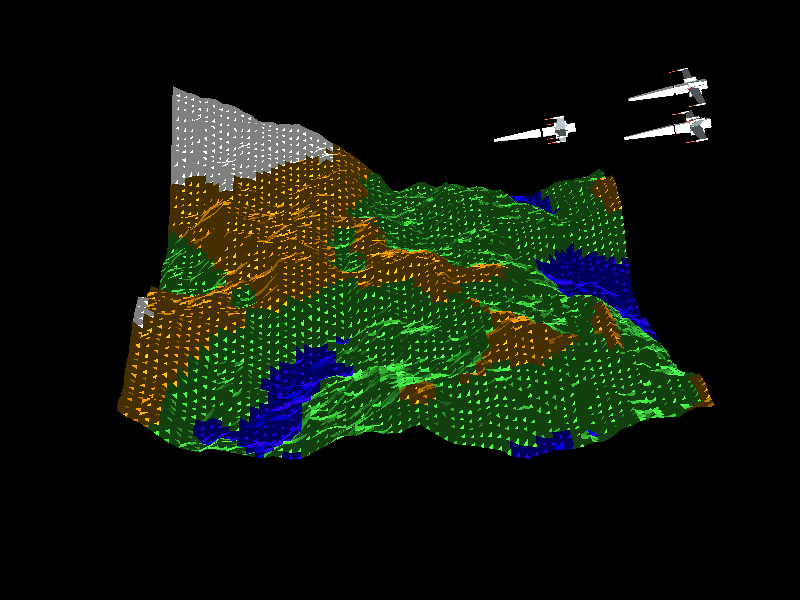
Project 10 wraps up the course and allows us to build two extensions into our graphics program. In my case, I did Phong shading and a Ray Tracer algorithm.

**Projects**

**Icon

Description automatically generated with low confidence**

The first addition was Phong Shading. I incorporated it into the module\_draw pipeline. The polygon struct added two additional attributes that saved a pointer to its surface normals and vertex locations in world coordinates. This allowed the same polygon to be accessed equally with either Gouraud shading or Phong shading. This image showed the smoother shading across polygons in the terrain so that it flows a little better, but also shows a little better highlighting in the specular reflections. The downside is, the artifacts remain so that certain triangles still show a perfect reflection, creating a grid-like effect.

****

**Addition 2: Ray Tracer**

A picture containing icon

Description automatically generated

This second extension is a ray tracer.

We’ve created a rayTracer struct that holds a polygon database. The polygon database is similar to an array list in Java or vector in C++, in that when it reaches its max capacity it will automatically create a new array of double size, and copy the original array into the new array. This allows a little flexibility but still fast access.

The pseudocode for the algorithm is as follows:

-Make one pass through the modules

-Take in all transformation matrices and polygons, storing their pointers into the ray tracer database.

-Calculate the Center of Projection and view window in world coordinates

-Shoot a ray from the COP through each pixel of the view window

-For each ray:

-Iterate through the polygon database and determine if the ray intersects any polygons

-If no hits, return black (or background color)

-If hit, iterate through lights and shoot a ray from the intersection point to the light

-For any non-ambient light, iterate again through the polygon database to determine if the ray is blocked by a polygon closer than the light

-Calculate the lighting contribution of the light only if the light is not blocked by another polygon.

-Color the pixel with the result of the lighting contributions.

**Extensions**

No extensions requested in this case – even with travel days I don’t think I made it in time.

**Reflection**

The Ray Tracer currently works to get the shading done correctly for a surface without a polygon blocking the path. I’m still working on a portion that would incorporate shadows. However, I was quite stoked to get the shading down pat.

**Acknowledgements**

Bruce helped immensely in getting the ray tracer off the ground. There were a couple key points that I had mistakes on (such as using the source rather than making a copy) that I probably never would have found. Thank you!

The x-wing module code was provided in a previous homework. The ray tracer pseudocode has been sourced from various sources for educational purposes. Here are some of the references I’ve been using to try to understand the process and get the code working:

[as3.pdf (stanford.edu)](https://graphics.stanford.edu/courses/cs148-10-summer/as3/instructions/as3.pdf)

[Raytracing Algorithm (tripod.com)](https://fuzzyphoton.tripod.com/rtalgo.htm)

[Ray Tracing Resources Page (realtimerendering.com)](https://www.realtimerendering.com/raytracing.html)