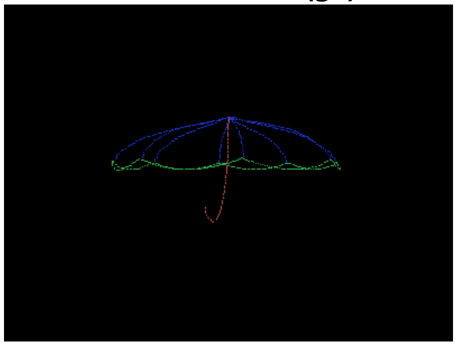
Benjamin Northrop
7/1/24
CS5310
Summer 24
****The PDF version does not support animated gifs, so you'll only see a single image. The word document (same file name) has the animated gif for reference. These images are annotated with (gif) in the title****

Introduction

Project 7 introduces us to Bezier curves and surfaces. This was a fast way to build shapes with polygons as well as curved lines defined by mathematical means.

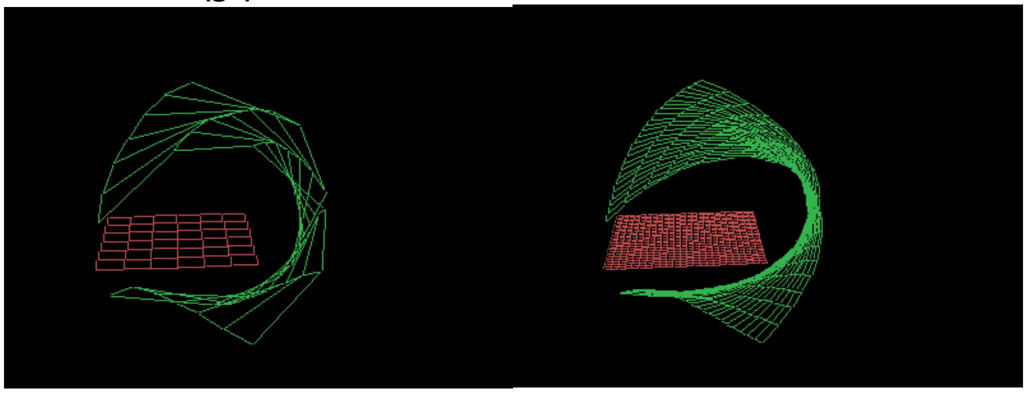
Projects

Bezier Curve umbrella (gif)



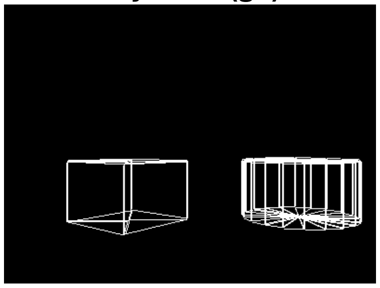
Using the test7a skeleton, I built an umbrella. Each portion is highlighted with a different color, and the green and blue portions were built with a loop. The red portion was a single line and was also built as a Bezier curve. I could have probably drawn a single straight line down, then drawn the Bezier curve for the handle, but I actually liked the curve to it (gave it a Alice in Wonderland feel to me).

Bezier Surface (gif)



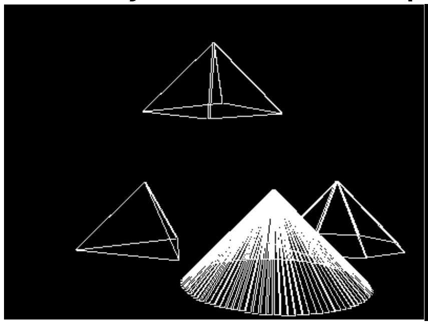
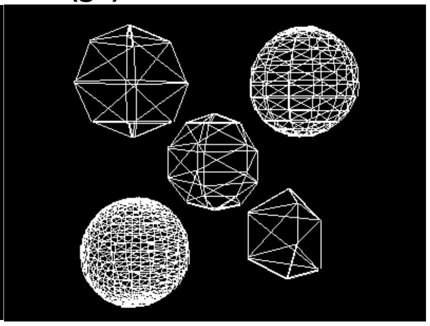
Again, we used the provided skeleton to create the bottom grid as a reference, and then built the surface around it. I wanted to play with a for loop and some trigonometry functions, as well as experiment with the random function. As a result, the x value is a function of the cosine of the index, the y value is the sin value of the index, and the z value is a pseudo-random value generated by the computer. Most other experiments ended up looking like a bundled bedsheet, but this one had a nice wave look to it.

Module Cylinder (gif)



The cylinder program was provided by the instructor in the previous homework. I made minimal changes to the actual code. Instead, I made an animation using 2 different levels of division to show the cylinder could also create a rectangular shape as well. This actually inspired me for the pyramid and sphere portions of this exercise.

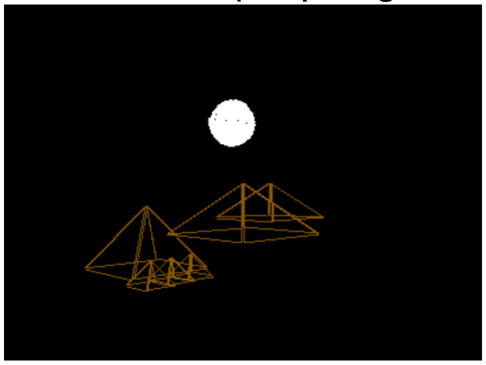
Module Pyramid and Module Sphere (gif)

Both the pyramid and sphere were designed from similar concepts. The pyramid uses the principle of a polygon fan, and sets the top point as the first point. It bound checks the sides input, but allows the user to specify how many sides are on the *base* of the pyramid, with a minimum of 3. The more sides added makes the pyramid more teepee-like.

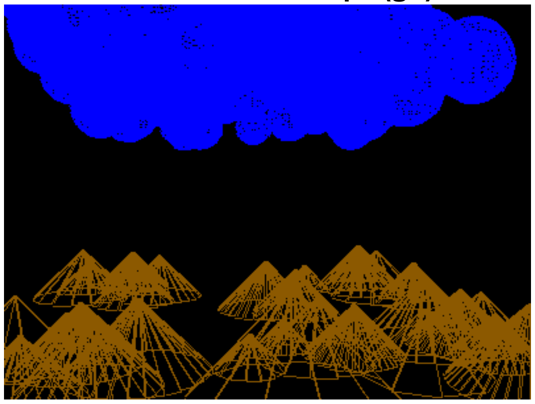
Sphere was made similarly, but this time using a polygon mesh for each “row” of polygons as we traveled up the sphere. I ran into some troubles with my initial plan of building a fan at the top and bottom, then using mesh to fill in between. As a result, the very first set of points are all at the origin, but it works with the rest of the program. The top, on the other hand, required a fan to be built to avoid a hole at the top, and that is at the end of the loop. As a result, the total “final” size of the sphere array is slightly larger than the square of the resolution. The benefit of this method though, is like the pyramid, the user can specify the number of desired divisions in the sphere. At lower divisions, we actually have more gem-like shapes than higher divisions.

Creative 1: Great (computer generated) Pyramids



This image was a practice of the pyramids and sphere. I wanted to simply practice a little more of the view pipeline, so I tried to do some simple translations and scaling to position each pyramid where I wanted it. I also practiced scaling by referencing the actual proportions of each pyramid and matching them together. As a result, I think we have a fairly accurate representation of the Great Pyramids. And to seal the deal, we have a nice night scene with a moonlit sky.

Creative 2: Perlin landscape (gif)



This image was the start of one of my goals by the end of the course – I realized that the Perlin noise generator I had implemented in one of my early projects could be used to create a landscape. In essence, the value of the noise would be an “altitude” of that point on the 2D space. I learned later that there is more to it (building a graph rather than just an array of pixels), but I still wanted to try it out as a prototype. We have 2 sections – mountains made up of pyramids, and clouds made up of spheres. I generate 3 Perlin noise values, somewhat arbitrarily, and then do a somewhat random transformation on them to determine the scale of the mountain. I translate it by a random amount (from the random function), and then give it a random number of sides, with a minimum of 5 sides.

The clouds were similar, except I did a small inner loop to create 3 spheres of clouds per mountain, and I would translate subsequent spheres in this loop by those perlin noise values. The desired result was to have a little “puffiness” to the clouds.

Extensions

Adding a functionality to the pyramid and sphere to allow user-input resolution is a little above and beyond. I also think the scene complexity of the Perlin noise landscape would be worth a point (if we’re allowing “double jeopardy”).

Reflection

The module viewing pipeline is making much more sense to me now, as I was able to confidently translate and scale objects as I tested each iteration. I'm still working on getting the coordinates correct for the initial setup, but it's coming together. I wish I could have come up with a better way to manage the actual altitude of each mountain top, but I realized I didn't know how to properly generate a point in world coordinates that would correspond to the screen coordinates. I'll look into it more down the road.

Acknowledgements

The cylinder function and the code to build the background / view pipeline was provided in the test functions. I did not modify them much except to change the perspective for a couple of the animations.