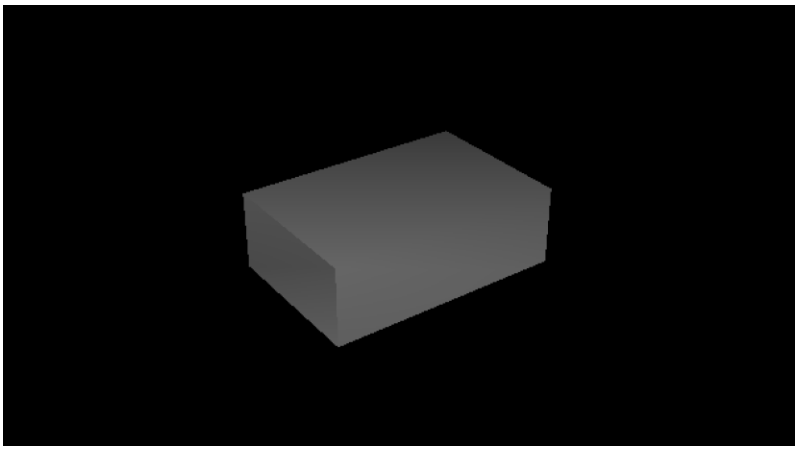
Benjamin Northrop
7/11/24
CS5310
Summer 24
****The PDF version does not support animated gifs, so you'll only see a single image. The word document (same file name) has the animated gif for reference. These images are annotated with (gif) in the title****

**Introduction**
Project 8 introduces us to Z-buffer rendering, which gives us the ability to remove hidden surfaces, as well as show depth by darkening an image as it gets further away from the viewer.

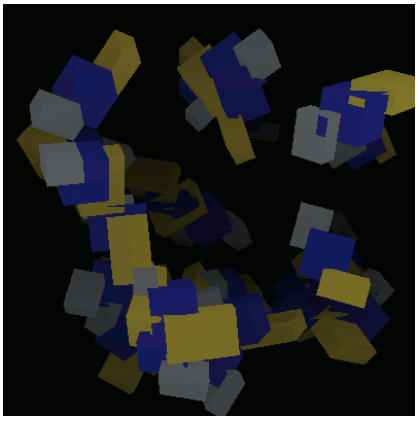
**Projects**

**zBuffer Block**



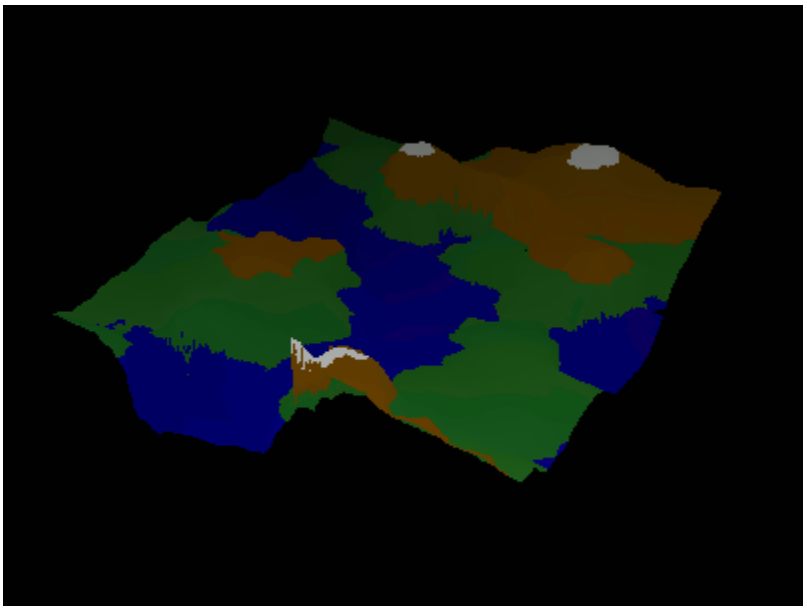
The first test function was the block from test8a. My first trouble was that I was drawing all except for the top and one side. After the debugging, I discovered that I had added clipping to my image_setz function that brought anything about 1.0 back down to 1.0. . . I think that was back when I had originally clipped all the colors back down to 1.0.

**Cubism (gif)**



The second test code provided was the cubism code. There's still a couple places where a face of one cube suddenly disappears a little early, and the cube underneath it pokes out – this may be the z-buffer failing at a certain point and then not drawing the pixel. I'll continue refactoring it and maybe adding a threshold to the z-buffer to fix it, like we discussed in class.

**Creative 1: Computer Generated Terrain (gif)**



This was a continuation of an idea I had last project – I wanted to generate terrain randomly, and the class about adding perturbations to a triangle to create tortilla chips gave me the motivation to try again. The first attempt followed the pseudocode from the class, but resulted in fracturing at shared edges of triangles due to the recursive call calculating a new midpoint y-value for the same (x, z) point twice. Instead, I researched more about height maps, and a 2D array to hold the y values of the divisions. Each recursive call increases the size of the array, fills the new columns with 0, and then calculates new values with a perturbation using a similar algorithm to the "Diamond-Square Algorithm." The result is a unit sized grid of y values, and the x-z values are scaled to the size of the image. To choose colors, y values above a certain threshold indicate snow caps, then the tree line (bare mountain), lower lands, and then water (negative y). Z-buffer shading is applied to a small degree, so the aft portion appears darker than the foreground. This was modeled after the cubism shading algorithm.

**Extensions**

Request an extension for design of this fractal terrain. It was a complex project to put together, but with a given set of arguments, could be used to design landscapes as a background for animated scenes such as flying aircraft.

**Reflection**

I am thoroughly pleased with myself for figuring out this fractal landscape problem, as most tutorials used a program to generate it, and finding material to generate the terrain from scratch was scarce. However, the Keith Stanger article provided the most thorough information about how

to manage the height map and make it work. I'm hoping to expand upon this project with the lighting homework for my final project.

**Acknowledgements**
The z-buffer used in the creative image came from the cubism.c homework.

The basis of the fractal terrain procedure came from this document:
Keith_Stanger_Fractal_Landscapes.pdf (uwaterloo.ca)

I started with a triangular algorithm, but I don't follow the diamond-square procedure exactly, as this procedure is not designed for tiling with other grids. The idea is that the program should be able to continually iterate and create finer details, and then you could zoom in on the square grid to create a large scene. Currently, I run into segmentation faults somewhere and am not exactly sure where to start debugging that. Therefore, iterations remain clipped to approximately 6-10.