Russell_Lowenstein_Ciociolo-Hinkell Assignment 8

CS 351 Assignment 8: Z-Buffer Rendering

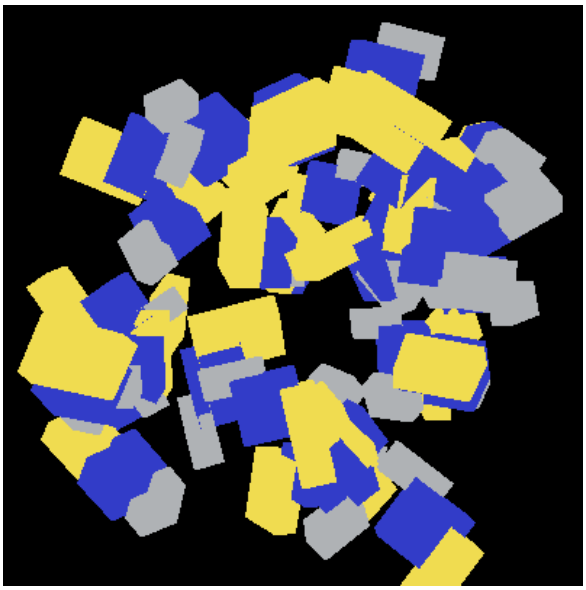
Justin Russell, Nicki Ciociolo-Hinkell, & Adam Lowenstein

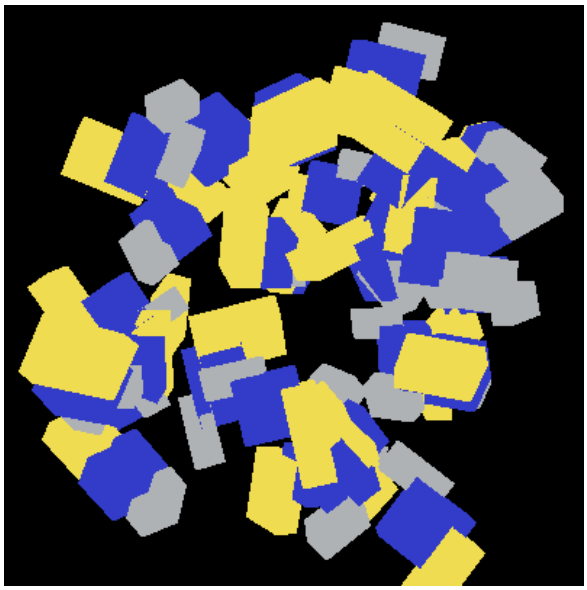
Abstract

In this assignment, we added z-buffer capabilities to our library and implemented all z-buffer-related functions. We began by adding z-coordinates or z-buffers to the relevant type definitions: iPoint required a single floating point z-value; Image required a floating point array of zbuffers (one for each pixel in the image); Polygon and Polyline both required zBuffer integers that could either be 1 or 0 depending on whether the zBuffer was used (1, or true, was the default); and Edge required two additional floats for zIntersect and dzPerColumn (the amount that the z-value changed on each iteration).

We also created additional functions to give us the ability to alter z-buffer values. These functions, which were both included in the Image.c file, proved invaluable in debugging. Image_resetZbuffer(src), which (in predictable fashion) reset all zBuffer fields for a given image src, was critical for proper display of our images, since it forced the z-buffer values for a given image to be independent of those from a previous image. [An alternative to including the Image_resetZbuffer function would have been to write the Image_zset1d function correctly in the first place. Had we done that, we would not have needed to include Image_resetZbuffer(). The end result, however, is the same.]

To test our functions, we used code provided by Professor Maxwell. The first graphic below shows our image before implementing the Image_resetZbuffer() function; the second shows the image after implementing this function:





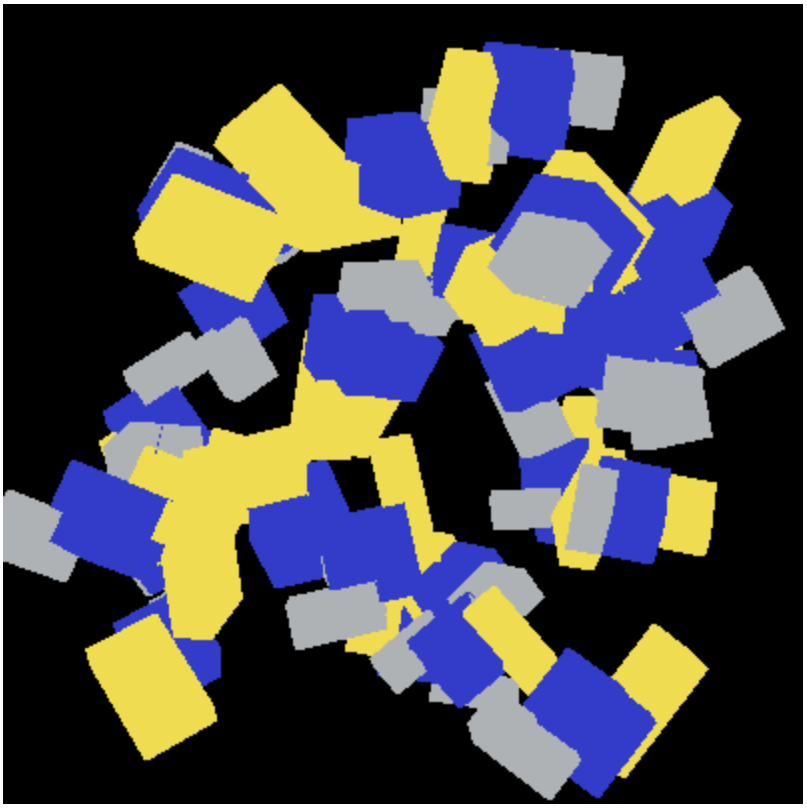
Note that the images are rotating in the left direction. To fix this problem, we made corrections in our Module.c function.

Description

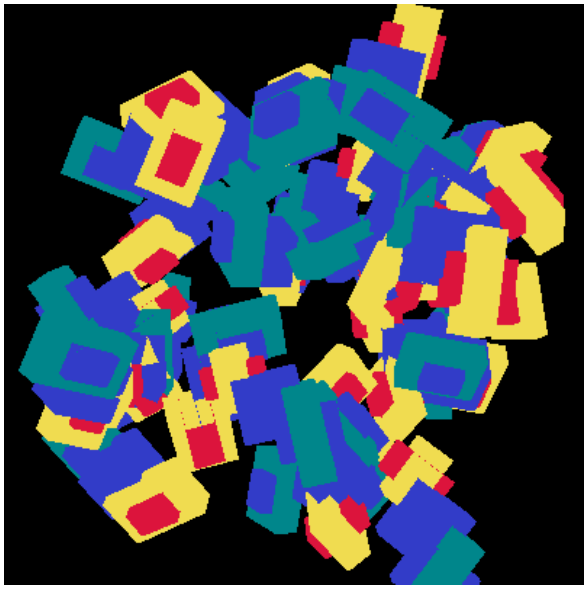
This assignment required us to make significant updates to our scanfill algorithm in order to implement z-buffers and draw the resulting image correctly. In the updateActiveList function (for polygon and polyline edges), we added a line to update the z-intersection. In makeEdgeRec, we added a number of accommodations for z-values and z-buffers, including a temporary variable dzPerScan (to allow for correct interpolation of z-values). We also created two additional temporary floating point variables, z0 and z1, to hold the values of 1/z at those points. We used 1/z values instead of z values because 1/z allows for correct interpolation in the x-, y-, and z-directions. As Professor Maxwell reminded us, regular z-values do not interpolate linearly in x and y. In scanfill, we also altered our fillscan function to draw images only after taking z-buffer values into consideration. For example, a pixel will only be drawn if it is between the front and back clip planes (the back is defined at 1.0 in the z-direction, while the front is implicitly set at infinity in 1/z coordinates).

Algorithms & Pictures

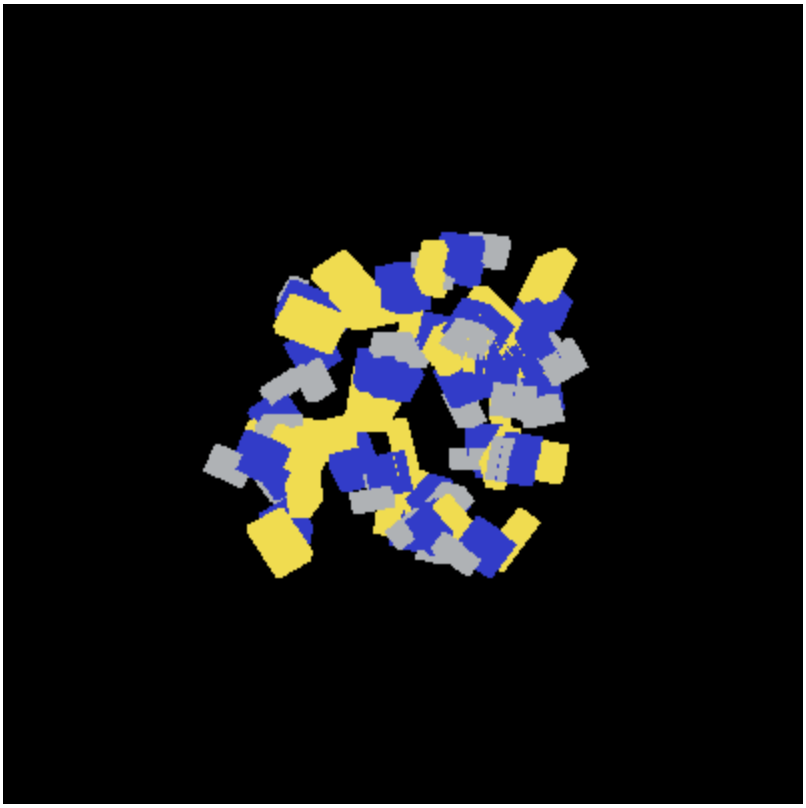
After hours and hours and hours of debugging, we generated the correct representation of Professor Maxwell's test code (note the correct direction of rotation):



Then, we changed some colors and added some rotations:



Finally, we altered the GTM:



Conclusion

Implementing z-buffers is a huge advancement for our library. As a result of this project, we now have nearly all of the tools necessary for complete three-dimensional image creation. We also used this project to fix additional outstanding bugs in our code (especially in Image.c and Module.c). Lastly, the project gave us a greater understanding of the methods for and the importance of freeing and allocating memory appropriately.
