Rick Gray
1375849

Homework 1: Frame Buffer

The assignment given was to develop a “frame buffer.” The purpose of the frame buffer is to load and process the given rectangle data into a single frame with which will be used by the calling function. For demoing purposes in this case, the calling function will use the produced in-memory frame to export a bitmap file. The frame buffer must correctly account for RGB color as well as correctly show the “topmost” color of each pixel given layered rectangles.

Loading each rectangle’s color to memory was simply done using a standard matrix in memory; each pixel was iterated over and the color of the rectangle loaded to its appropriate place in the matrix. The algorithm used to handle flattening the z axis is referred to as “depth testing.” The algorithm works by checking every rectangle that encompasses a pixel and keeping track of whichever rectangle has the forwardmost z coordinate; it is in front of all others and thus should be visible. For each rectangle, if the z coordinate is closer to the viewing port, then the rectangle’s color is saved to that pixel and the z coordinate is updated. However, if the rectangle is farther from the viewing port, it is simply discarded as it is spatially “behind” the forwardmost rectangle and would be hidden from view. Once every z coordinate of every pixel of every rectangle has been iterated over, the final color matrix represents the view of the rectangles from the viewing port as if they were behind or in front of each other by their given z coordinate.

Implementation was simply a matter of declaring matrices in memory to track color as well as to track the current forwardmost rectangle on the z axis. With each call to *drawPoint* in the frame buffer, a vertex (pixel) and a color was passed in as arguments. Each vertex coordinate was rounded and casted from *double* to *int*. The boundaries were checked next to account for any pixels of the rectangles that fall off screen. Following this, if the depth test was not enabled, the respective indices of the color matrix were loaded with the color. If depth testing was not enabled, then the rectangles would overwrite each other simply depending on the order of them in the input. If depth testing is enabled, then the pixel’s z coordinate is compared to its respective point in the depth buffer matrix. Again, if it is smaller, the data is discarded as this rectangle is hidden behind another at this pixel. If the z coordinate is greater than or equal to the z coordinate held in the matrix, we know this rectangle is in front of all other known rectangles and should be updated as such. The z coordinate is saved to the depth buffer for future comparisons and the color buffer is updated with this rectangle’s color.

The resulting images produced follow expectations. When the depth testing flag is disabled, the resulting image shows the rectangles simply painted over each other in order of their listing in the input file. The yellow rectangle is listed before the light brown rectangle, so it is covered by the light brown rectangle despite being in front of it. The dark brown rectangle is also visible despite being farther from the view port than can be seen by the set viewing distance. With depth testing enabled, the yellow rectangle properly covers the light brown rectangle but is still covered by the light green rectangle. The dark brown rectangle is not visible at all, however.