Programming Assignment #3
Rasterization

Due Thursday, November 1 by 11:59pm

(The grade for this assignment is out of 100 points, but you can obtain up to a maximum of 125 points by doing more optional components. Additional points beyond 100 will be applied to your final score for all programming assignments.)

Required Components: (60 pts)

You must do all of these parts using only GL_POINTS as your rendering primitives if you use OpenGL. Alternatively, you could render to a buffer, and display it as a texture in OpenGL, or use a library such as OpenCV that can display the contents of a buffer.

(1) Implement Bresenham's line drawing algorithm as a function that takes 4 floats as input (so it can be tested with arbitrary lines) and returns a list of vertices. Display these vertices. You must handle ALL possible line slopes. (20 pts)

(2) Implement triangle rasterization. That is, implement the scan-line polygon rasterization algorithm, but it only has to work on triangles for this part of the assignment. This function should take 6 floats as input, and return a list of vertices. Display these vertices. (NOTE 1: You may assume that the triangle is not degenerate; that is, it is not just a point or a line) (NOTE 2: This function will likely build upon your line-drawing algorithm from part 1.) (20 pts)

(3) Implement 2D line-clipping using the Cohen-Sutherland algorithm. This function should take as input 4 floats (representing the segment endpoints) and produce as output 4 floats (representing the new endpoints). (20 pts)

Optional Components: (40-65 pts)
NOTE: You MUST do at least 40 points worth of components from this section to achieve full credit on this assignment.

(1) Implement another line rasterization algorithm. Run this algorithm on a set of 1000 lines and time it; also run your Bresenham algorithm on the same number of lines and time it. Report these timings. (10 pts)
(2) Implement robust scan-line polygon rasterization. That is, extend the triangle rasterizer from part 2 above to support polygons with an arbitrary number of vertices, concave polygons, and degenerate polygons (*i.e.* polygons that are actually a single point or a single line). (15 pts)
(3) Implement another line-clipping algorithm. (10 pts)
(4) Implement Sutherland-Hodgman polygon clipping. (15 pts)
(5) Implement Weiler-Atherton polygon clipping. (20 pts)
(6) Implement flood-fill (4 or 8 fill). The algorithm should take as input a list of vertices and fill in the polygon defined by those vertices. (10 pts)
(7) Implement supersampling line anti-aliasing for your line-drawing algorithm. (10 pts)
(8) Implement ratio line anti-aliasing for your line-drawing algorithm. (10 pts)