Fall Quarter 2015

# ECS175 Project 1 Information Sheet

This is a general overview of project 1, please refer to project guideline published on SmartSite and the project prompt for additional requirements. The command syntax here is for demonstration purposes only, you don't necessarily have to follow them.

## Requirements

For this project, you MUST set the pixel buffer with your own code and use glDrawPixels to draw your scene, you CAN NOT use glVertex…, gluOrtho2D or other GL commands that draw lines or fill polygons to implement the requirements. If you are not sure which specific gl command you can use, please contact the TAs.

Along with the requirements outlined in project guideline, here's a rough list of things we will check during grading. Make sure the following are working and you should do well. (Point value associated with specific part is in [])

1. General functions [10]: User should be able to specify the dimension of the view port (your OpenGL window), you should not hardcode the size. The size can either be entered interactively or through command line arguments. Your program should allow input data from file and export (save) data to file. Also, remember your program should at least be able to handle 3 polygons/lines.
2. Line drawing [20]: Specifically DDA algorithm and Bresenham algorithm. Your program should allow user to choose which algorithm to use and draw any lines the user commands. For example:
    a. >>Bresenham (0,0) (140,200) //this should draw a line using Bresenham algorithm from (0,0) to (140,200)
3. Polygon rasterization [15]: Using either DDA or Bresenham, your program should draw an n-sided polygon, filled in using the scan line algorithm, given n points from the user. To receive full credit, you only needed to implement rasterization of "convex polygons" (i.e. triangle, square, hexagon etc. Refer to http://mathworld.wolfram.com/ConvexPolygon.html). For example:
    a. >>Polygon 3 (0,0) (-10,0) (-10,-10) //this should draw a triangle with the specified 3 vertices

    Refer to extra credit section for concave polygons
4. Transformation [20]: User should be able to interactively specify a polygon, then apply either a scale, translate or rotate transformation. For example:
    a. >>translate 1 (50, 50) //this should translate polygon with internal label 1 with vector(50, 50)

5. Clipping [15]: Specifically, your program should implement Cohen-Sutherland and Sutherland-Hodgeman algorithm to clip the viewport. User should be able to list xmin, xmax, ymin, ymax values that define clipping planes for the scene.

   Refer to extra credit section for clipping with generic function

6. UI [15]: If your UI is functional without any issue (i.e. crashes when running some command, inconsistent with README instructions), you should receive full credit on this part. Refer to Project Guideline for some helpful resources for UI creation.
7. Manual [5]

## Extra Credit

The following are published extra credit for this project.

1. Rasterizing concave/complex polygon [5]
2. Support generic clipping planes (defined by any function) [10]
3. If your UI is stunning! [up to 5]

If you've done anything else extra, please document it in README file so we can assess it and try to give you more points!

## Helpful Notes

In this project, you don't need to implement normalized device coordinate, hence the requirement for user adjustable view port. However we strongly recommend you familiarize yourself with it since you will need to implement NDC in the following projects.

You should only need glDrawPixels to draw the scene, but glSwapBuffers might also be useful. Refer to glut-3.spec.pdf if you are unclear with any GL command.

Rasterizing polygons might be trickier than it seems, even just for convex polygons. The handouts given during the first lecture contains a lot of useful information. When testing your rasterizing function, we suggest testing with multiple different shapes to ensure it functions properly