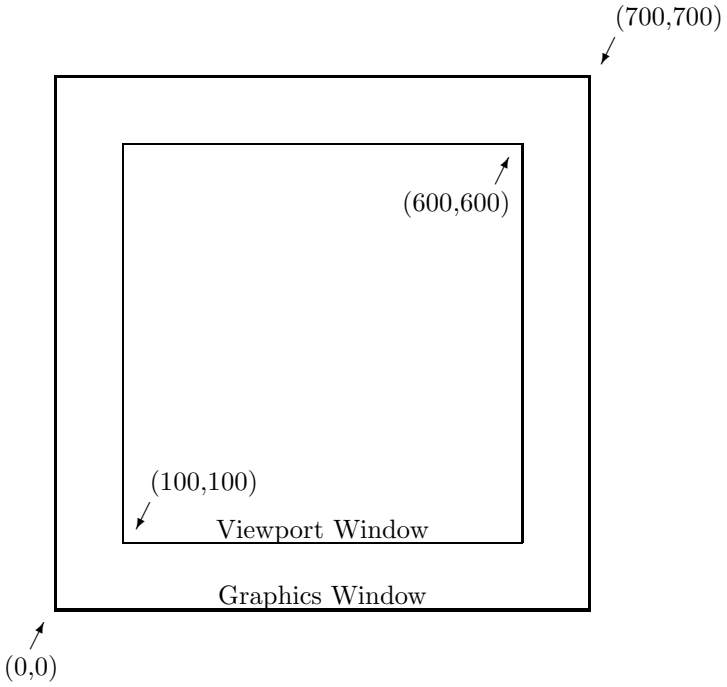
CSC 315 – Introduction to Computer Graphics

Fall 2007

Programming Assignment 1
Due by Midnight, September 19, 2007

This assignment will be comprised of four C or C++ programs utilizing the OpenGL API. The credit awarded for each program is noted at the end of each description.

Program 1 - Viewports and Scanline Conversion: You are to write a program which will define a viewport within a graphics screen and then draw lines within the viewport. The graphics screen and viewport will be defined as follows:



While you will use a command like `glOrtho2D` to define your graphics screen, one of the key components of this assignment is that you use *integers* to specify pixel coordinates. If you remember, `glOrtho2D` takes *floats* as its arguments. You will therefore have to implement some typecasting in function calls. It is also important that you use the `glVertex2i` to draw your pixels. Formerly we used the `glVertex2fv` command to draw our pixels. Some of you noticed that when you did this you got blank lines or regions on the screen. By defining pixel locations explitly with integers, we should avoid this problem.

Here are the particular things I want this program to do.

- Draw the initial graphics and viewport windows and set the background to black. Using **simple rectangular scan conversion**, set the background of the viewport to white.
- The functions of the mouse and keyboard should be as follows:
  - If the mouse is within the viewport, clicking the left button should define the initial point of a line.
  - If the mouse is within the viewport, clicking the right mouse button should draw a line between the two points using the **midpoint line scan conversion algorithm**.[1]
  - If the mouse is within the viewport, clicking the right mouse button after a line has been completed should result in the viewport being cleared.
  - If either mouse button is clicked when the mouse is outside of the viewport, the program should send an error message to `stdout` and include the graphics screen coordinates (not mouse coordinates).
  - Hitting "q" or "Q" on the keyboard should exit the program.

(10 pts.)

[1] It's worth five bonus points if you implement the **enhanced midpoint line algorithm** that looks ahead two pixels!

Program 2 - Line Clipping Algorithms: This program will be identical to the first program with one exception. In this program the starting and ending points for the line may be anywhere in the graphics screen, but only the line segment *within* the viewport will be drawn. To accomplish this, you will implement the **Cohen-Sutherland line-clipping algorithm**.[2] *(12 pts)*

Program 3 - Drawing and Clipping Circles: Using the graphics and viewport windows as before, write a program in which the mouse functions are defined as follows:

- If the mouse is within the viewport, clicking the left button should define center of a circle.
- If the mouse is within the viewport, clicking the right mouse button should define the radius of the circle. The circle should then be drawn using **Bresenham's integer midpoint circle scan conversion algorithm**.[3] Any part of the circle that lies outside of the viewport should be "clipped" using the **Sutherland-Hodgman polygon-clipping algorithm**.
- If the mouse is within the viewport, clicking the right mouse button after a circle has been completed, or before the left mouse button has been pressed, should result in the viewport being cleared.
- If either mouse button is clicked when the mouse is outside of the viewport, the program should send an error message to `stdout` and include the graphics screen coordinates (not mouse coordinates).
- Hitting "q" or "Q" on the keyboard should exit the program.

(14 pts.)

Program 4 - Coordinate Mapping and Antialiasing: In this program we will use the same definitions for the graphics and viewport windows, but we will map a new set of coordinates to the viewport. We want our program to draw the function

$$f(x) = 500 \exp\left(-\frac{x^2}{150}\right) \sin(2x)$$

on the domain [-30,30]. Your set of viewport coordinates must allow for all parts of the function to be displayed! While you may use any algorithm you choose to draw the points for the mathematical function, be warned that you will have to write the algorithm to map the coordinates needed for the mathematical function to the coordinate system of your viewport. The mathematical function will be using *floats* or *doubles*. The routine to write pixels has to utilize *ints*. The mouse and keyboard should function as follows:

- If the mouse is within the viewport, clicking the left button should return the *coordinates relative to the mathematical function* to `stdout` with a suitable notification that this is a *function coordinate*.
- If the mouse is within the viewport, clicking the right button should do nothing.
- If the mouse is outside the viewport, clicking the left mouse button should return the *coordinates of the graphics window* to `stdout` with a suitable notification that this is a *graphics window coordinate*.
- If the mouse is outside the viewport, clicking the right button should do nothing.
- Hitting "q" or "Q" on the keyboard should exit the program.

Include enough points to make your function look *smooth*. To increase the apparent resolution, utilize **antialiasing** via a **weighted area sampling function**. I am leaving the construction of this up to you – but I expect some explanation of the implementation in the code as comments. Just so you know, with this program you will have to include the math header file and link with the math library. *(14 pts)*

Assignment Submission Procedures

You should have a directory named `assign1`. Under this directory I would like you to have four other directories `prog1`–`prog4`. Each directory should contain the corresponding program code and makefile. Send me, via electronic mail, a compressed tar file of the `assign1` directory and all the subdirectories. Use your last name as the name of tar file. If your last name were Doofus, you would `cd` to the directory containing the `assign1` directory and type:

```
tar -cvf - assign1 | gzip --best >Doofus.tgz
```

Mail this file to `pounds_aj@mercer.edu` by the time noted.

[2] For five bonus points you may implement the **parametric line clipping algorithm** rather than the **Cohen-Sutherland algorithm**.

[3] For five bonus points you may implement the **midpoint circle scan conversion algorithm that uses second order differences** instead of Bresenham's algorithm.