CS 430 – Computer Graphics
Fall 2017
Assignment 1

In this assignment you will demonstrate your ability to:

1. Read in data from a file and organize it into structures for drawing
2. Set up a software frame buffer to render to
3. Implement a line drawing algorithm to render the lines into the software frame buffer.
4. Output the software frame buffer as an XPM file.

For this assignment you may assume that all endpoints that define the lines are within the bounds of the software frame buffer.

Make sure you give yourself adequate time. The programming components in particular can be quite time consuming.

As a reminder you may use the programming language of your choice (as long as the TA and Professor are comfortable with your choice), though I recommend C/C++. Regardless, make sure that your program can run on the Drexel tux cluster to ensure its system independence.

Submission Guidelines

1. Assignments must be submitted via Bd Learn
2. Submit a single compressed file (zip, tar, etc..) containing:
    a. A PDF file containing your answer to the theory question(s).
    b. A README text file (**not** Word or PDF) that explains
        i. Features of your program
        ii. Language and OS used
        iii. Compiler or interpreter used
        iv. Name of file containing main()
        v. How to compile/link your program
    c. Your source files and any necessary makefiles, scripts files, etc… to compile and run your program.

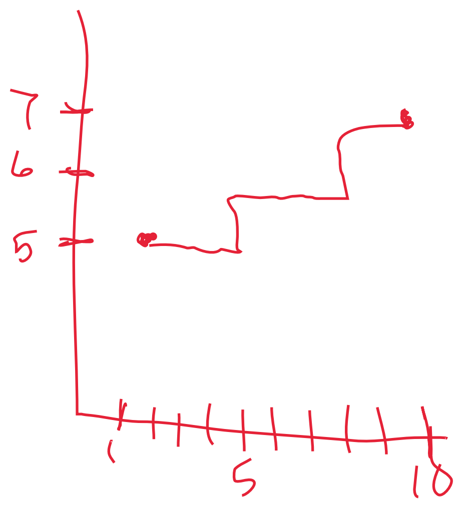
Theory Question(s)

1. What is the implicit equation of a 2D line? How can the equation be used to characterize the location of an arbitrary 2D point (x, y) relative to the line? That is, how can it be used to tell where an arbitrary point is relative to the line? (4pts)

2. For the line segment specified by endpoints (10,7), (1,5) use the DDA algorithm to fill out the table below and then use its results to draw the line. (8pts)

X (floating point)	Y (floating point)	Pixel(X)	Pixel(Y)
1	5	1	5
2	5 22	2	5
3	5 44	3	5
4	5 66	4	6
5	5 88	5	6
6	6	6	6
7	6.22	7	6
8	6 44	8	6
9	6 66	9	7
10	6 ~~8~~8	10	7

$$\frac{\Delta Y \quad 5-7}{\Delta X \quad 1-10} \qquad +\frac{2}{9} = m$$



3. For the line segment specified by endpoints (10,7), (1,5) use the Bresenham algorithm to complete the table below and use the table to draw the line. (8pts)

D (current)	Pixel(X)	Pixel(Y)	D (updated)
N/A	1	5	-5
-5	2	5	9
9	3	6	-15
-15	4	6	-7
-7	5	6	-3
-3	6	6	-3
-3	7	6	1
1	8	7	-13
-13	9	7	-9

$\frac{2}{9}$

$$y - d_y = \frac{dy}{dx}(x - dx)$$

$$y - 2 = \frac{2}{9}(x - 9)$$

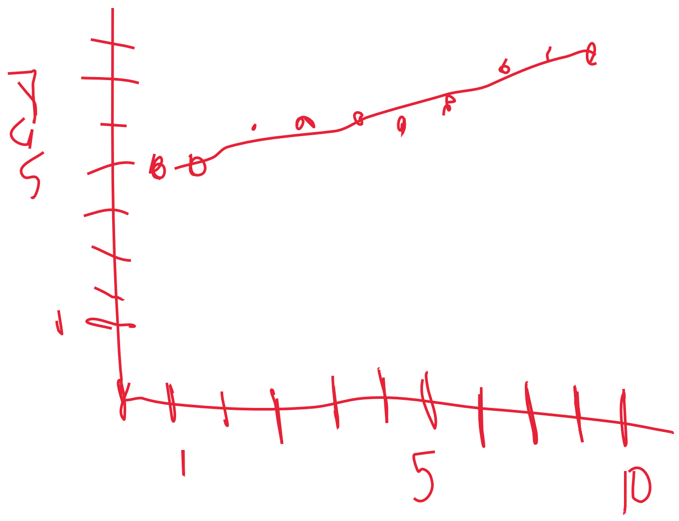
$4 - 9 + 86 \quad 45 \quad 2$

$$2(2) - 9 + 2(9(5) - 2(1))$$

$$2dyP_x - 2dxP_y + \underbrace{(2dy - dx + 2(dxP_y - dyP_x))}_{\text{constant}}$$

$4 \quad 18$

$$D = 2(2)(P_x) - 2(9)P_y + 81$$



4. Derive the decision equation(s) (init, and the two directions) used in the Bresenham algorithm when the slope m<-1 (8pts)

$m = (P_x + \frac{1}{2}, P_y - 1)$

$2ax + 2by + c = 0$

$2a(P_x + \frac{1}{2}) + 2b(P_y - 1) + c = 0$

$2aP_x + a + 2bP_y - 2b + c = 0$

$2aP_x + 2bP_y + (a - 2b + c)$

Introduction:

For the majority of our assignments we will be writing out into an image format called XPM and reading in from a simplified postscript (PS) file format. So first let's get an idea of what each of these plaintext files should contain…..

PS File Format

For most of the assignments you will need to read in files in the Simplified Postscript file format (*.ps). There are many commands supported in the .ps format, but in each assignment you will need to only support some small subset.

- The text of the supported commands will be bounded by two delimiters where %%%BEGIN marks the beginning and %%%END marks the end
- There may be blank lines anywhere between these delimiters
- All text outside these delimiters should be ignored
- Postscript assumes that the origin (0, 0) is the lower left corner of the window
  - Note this is different than XPM
- For this assignment you only need to support lines in the following format
  - x1 y1 x2 y2 Line
- Here's an example PS file

```
%%%BEGIN
375 100 300 230 Line
499 0 0 250 Line
170 450 400 350 Line
350 300 120 400 Line
%%%END
```

NOTE: Be careful that the input file is formatted for the OS you are using. One easy way to convert is to choose "Save As" in Textpad and choose our OS.

XPM Files Format

We will be writing XPM files to act as our “software pixel buffer” throughout the course.

You can find a brief description of it in the XPM handbook (you’ll just need Chapter 1 and 2 at most)

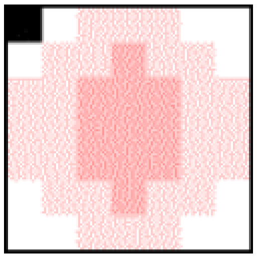
Several free programs can view, read and write XPM files. One program for Windows is irfanview. You must install all the plugins. You can use gqview for Linux. For OSX you can obtain XQuartz then ssh to tux with x-forwarding (*ssh -x*).

Here’s some of the basics of the XPM format:

- The XPM format assumes pixel (0, 0) is in the upper left-hand corner
  - This is different from most other applications that assume (0,0) is the bottom-corner. So you will have to compensate for that.
- Put data for “width height ncolors cpp” in the first string. For example:
  - /* width height num_colors chars_per_pixel */
  - "7 7 4 1"
- Then you have “ncolors” strings associating characters with colors. The character ‘c’ in the string associates the previous symbol in the string with the subsequent RGB hexadecimal color in that same string. For example:
  - /*colors */
  - "- c #ffffff",
  - "# c #ffe0e0",
  - "a c #ffb7b7",
  - "X c #010101",
- And last you have “height” number of strings each with “width” * “chars_per_pixel” characters, where every “chars_per_pixel” length string must be one of the previously defined groups in the “colors” section. For example:
  - "X-###--",
  - "-##a##-",
  - "##aaa##",
  - "##aaa##",
  - "##aaa##",
  - "-##a##-",
  - "--###--"

Here’s an example image

```
/* XPM */
static char *sco100[] = {
/* width height num_colors chars_per_pixel */
"7 7 4 1",
/* colors */
"- c #ffffff",
"# c #ffe0e0",
"a c #ffb7b7",
"X c #010101",
/* pixels */
"X-###--",
"-##a##-",
"##aaa##",
"##aaa##",
"##aaa##",
"-##a##-",
"--###--"
};
```



The actual programming assignment!

Write a program that accepts the following command arguments. Defaults are in parenthesis. You should be able to process any subset of the options in any arbitrary order.

a. [-f] The next argument is the input "Postscript" file (hw1.ps)

For now we will hard-code the size of your frame buffer to be 500x500 (that is 500 pixels wide by 500 pixels high) and that you may assume that all the vertices specified in the file are within those bounds.

Your program should print output images in the XPM file format to a file called *<filename>.xpm* where *<filename>* is the parameter [-f] without the .ps suffix. All pixels should be initialized to white. Draw your objects in black.

Your general program flow should be:

1. Read in line segment data (PS)
2. Scan covert (i.e. draw) lines into software frame buffer using either the DDA or the Bresenham Algorithm.
3. Output your image (the frame buffer and header information necessary to make an XPM file)

Bonus

For ten additional points, allow the user to pass a `[-x]` flag to indicate to use Breshnham's line drawing algorithm. If that flag is omitted, use DDA for line drawing.

Grading Scheme:

In order to get any credit at all you must be able to generate at least read in a PS file and write out a valid XPM file

1. Theory Questions (28pts)
2. Can read in PS file and output a valid XPM file (35pts)
3. Successfully handles seven basic single line test cases (3pts each = 21pts)
4. Successfully handles a multi-line test case (10pts)
5. Program easy to compile and run based on README (6pts)
6. Bonus (10pts)

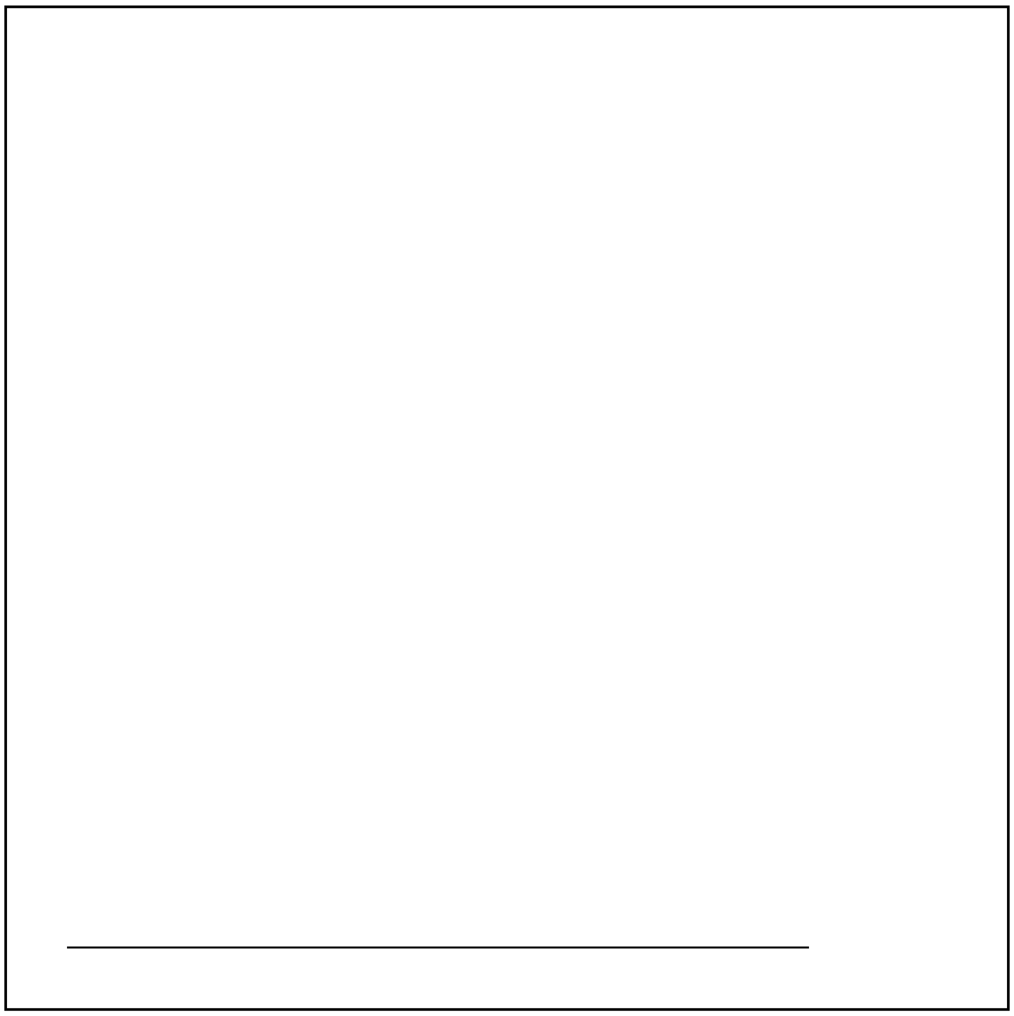
Common deductions:

1. Nothing drawn (-100pts)
2. Missing files (including readme) (-5pts each)
3. Cannot compile/run out-of-the-box on TUX (-10pts)

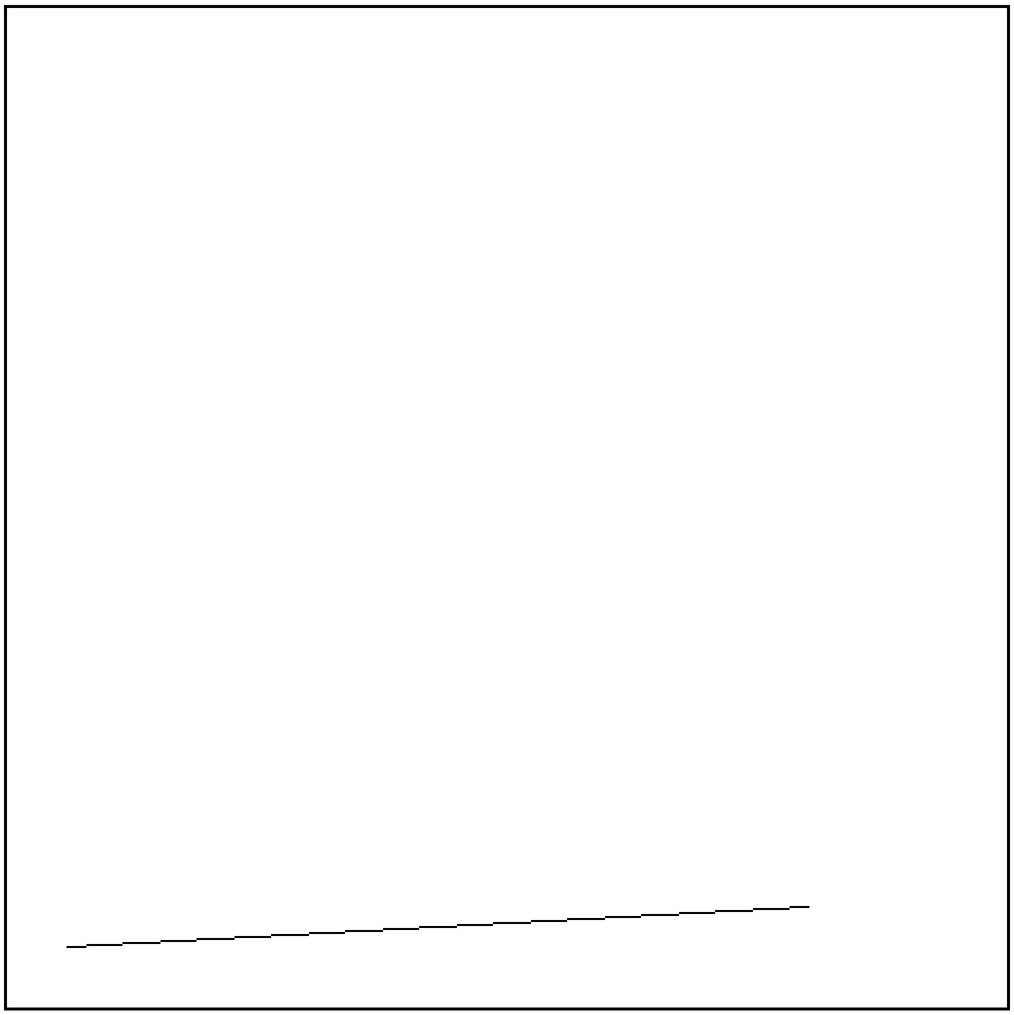
Example Output

Here is the output of a few of the test cases (note: border is not include in the actual output):

./A1 -f hw1_1.ps



./A1 -f hw1_2.ps



./A1 -f hw1_8.ps