CSC 240: Computer Graphics
SOLUTION -- Midterm: Fall 2021
Friday, October 15 – Sunday, October 17

- This is a self-scheduled, closed book exam.
- While completing the exam you are allowed to use one page of notes, 8.5x11, both sides.
- Testing hours are Friday 3:00-9:00 pm, Saturday and Sunday 12:00-6:00 pm.
- You have two hours to complete the exam from the time you sign it out.
- Students with accommodations allowing extra time can compute their time accordingly.
- Exams must be turned in by the end of the testing window, so please plan ahead.
- Students with accommodations for individual space can reserve a room in advance.
- If you are unable to make progress on any part of the exam, tell me what you tried: describe your thought process. I may be able to grant partial credit.
- When your exam is complete, before submitting it, please copy, sign, and date the statement below:

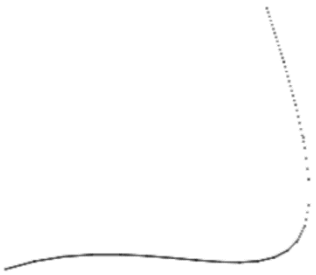
  "I certify that my work on this exam adheres to the Smith Honor Code and the instructions given above."

Signed: Date:

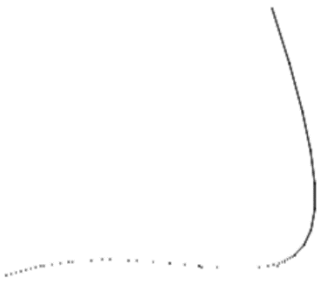
Part 1: Lines and Curves (20 points)

Suppose that you have a little brother who is learning computer graphics. He has written his own function to implement line drawing, and used it to write a function that draws Bézier curves. But there's a problem, and he comes to you for advice. The curve that was drawn has sections with gaps in them, as shown at right.

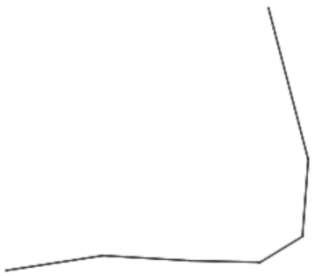
a.) Given what you know about line drawing and Bézier curve algorithms, what do you think is likely wrong with his program? What advice would you give for fixing it?
The line drawing is looping over x regardless of slope. It needs to loop over y when the slope is greater than 1.



b.) Your brother works for a long time making changes to the program and comes back with the image at right. The part that was incomplete before is now solid at last, but another part of the curve is now dotted! He's still not happy, and hopes you can tell him again what is wrong. Now what is likely wrong with the program?
Now the line drawing is always looping over y. When the slope is less than q it should loop over x.



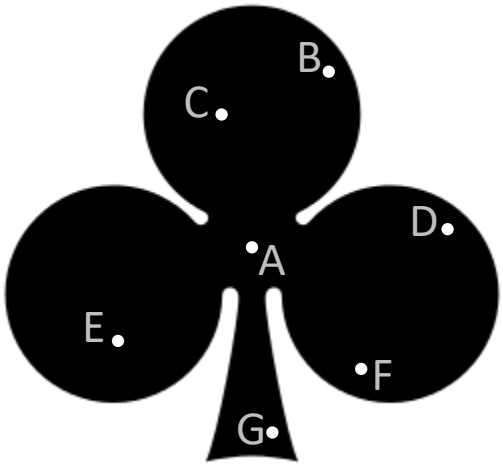
c.) After more work, your brother thinks he almost has it. The program is drawing a connected sequence of points. However, it has sharp angles instead of the smooth curve he wanted. He has made so many changes to the program he's not sure which ones are responsible. What is likely to be causing the curve to look jointed?
The number of steps in t is likely too small, meaning that the computed points are far apart. When they are joined by a straight lines, you get the sharp angles in the image.



d.) What is the minimum order of the Bézier curve in this image (linear, quadratic, etc.)?
The curve has two inflection points, so it must be at least cubic.

Part 2: Fill Algorithms (20 points)

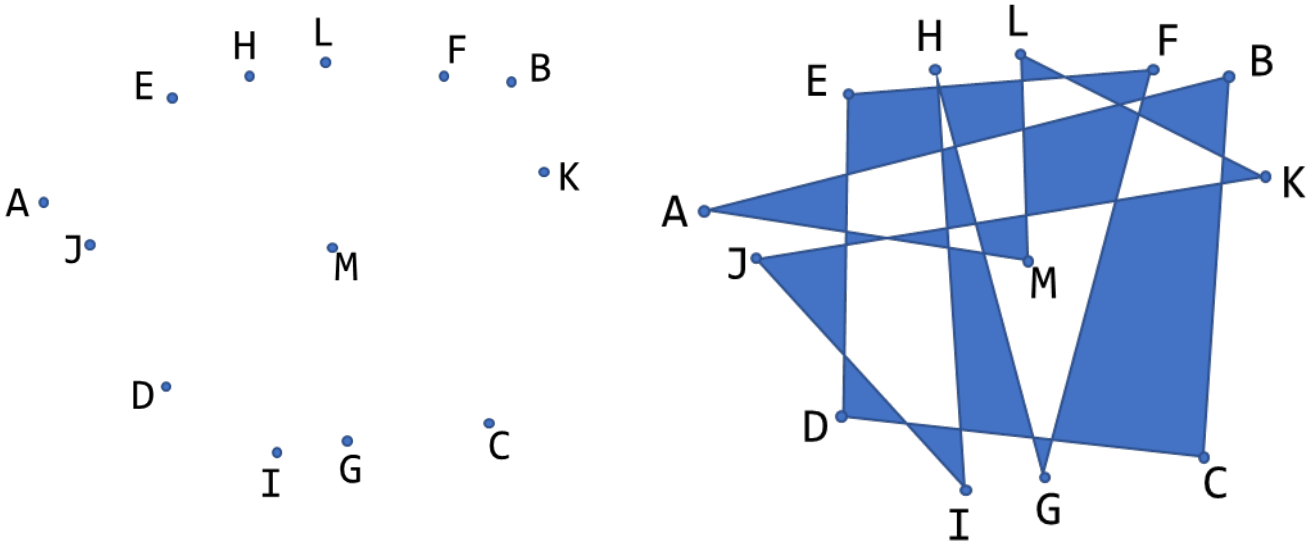
a.) The shape at right is to be filled using a flood fill algorithm. The implementation attempts recursive calls in the following order: north, south, east, west. If the flood fill starts at point A, list the order in which the labeled points would be filled.



It will start by going north until it hits the top, then it will move east one step and then head south. The fill will scan up and down until it has filled all the way to the east. Depending on how it hits the boundary at dividing points, it might fill in several possible orders:

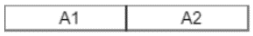
A, G, B, F, D, C, E,
A, G, F, D, B, C, E
A, B, F, D, G, C, E
A, F, D, B, G, C, E

b.) A polygon is to be drawn using the vertices A through M below, connected in alphabetical order. Draw how the resulting shape would be filled in using the scanline fill algorithm.



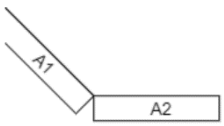
Part 3: Hierarchical Transformations (20 points)

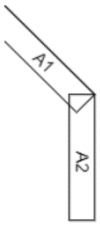
A robot arm is simulated using a hierarchical model with two segments. Segment A1 is the root part and segment A2 is a subpart. Both are drawn using the command `graphics.fillRect(0,0,100,20);` with different modeling transforms accounting for the difference in results. The function to draw the arm is shown below. As shown, the code generates the image shown above.

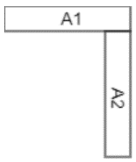



```
function applyA1Transform() {
    // identity
}

function applyA2Transform() {
    graphics.translate(100,0);
}

function draw() {
    applyA1Transform();
    drawA1();
    graphics.save();
    applyA2Transform();
    drawA2();
    graphics.restore();
}
```

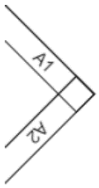
Below are several pictures showing different configurations of this system. Give the graphics transformations that would be required in `applyA1Transform` and `applyA2Transform` to generate these images, assuming that none of the other code changed. All are combinations of translation and rotation only.

a.) 

b.) 

c.) 

d.) 

e.) 


```
a.) applyA1Transform: graphics.rotate(Math.PI/4);
    applyA2Transform: graphics.translate(100,0); graphics.rotate(-Math.PI/4);
b.) applyA1Transform: graphics.rotate(Math.PI/4);
    applyA2Transform: graphics.translate(100,0); graphics.rotate(Math.PI/4);
c.) applyA1Transform: // identity
    applyA2Transform: graphics.translate(100,20); graphics.rotate(Math.PI/2);
d.) applyA1Transform: graphics.rotate(Math.PI/4);
    applyA2Transform: graphics.translate(100,0); graphics.rotate(-Math.PI);
e.) applyA1Transform: graphics.rotate(Math.PI/4);
    applyA2Transform: graphics.translate(100,0); graphics.rotate(Math.PI/2);
```

Part Four: Curves (20 points)

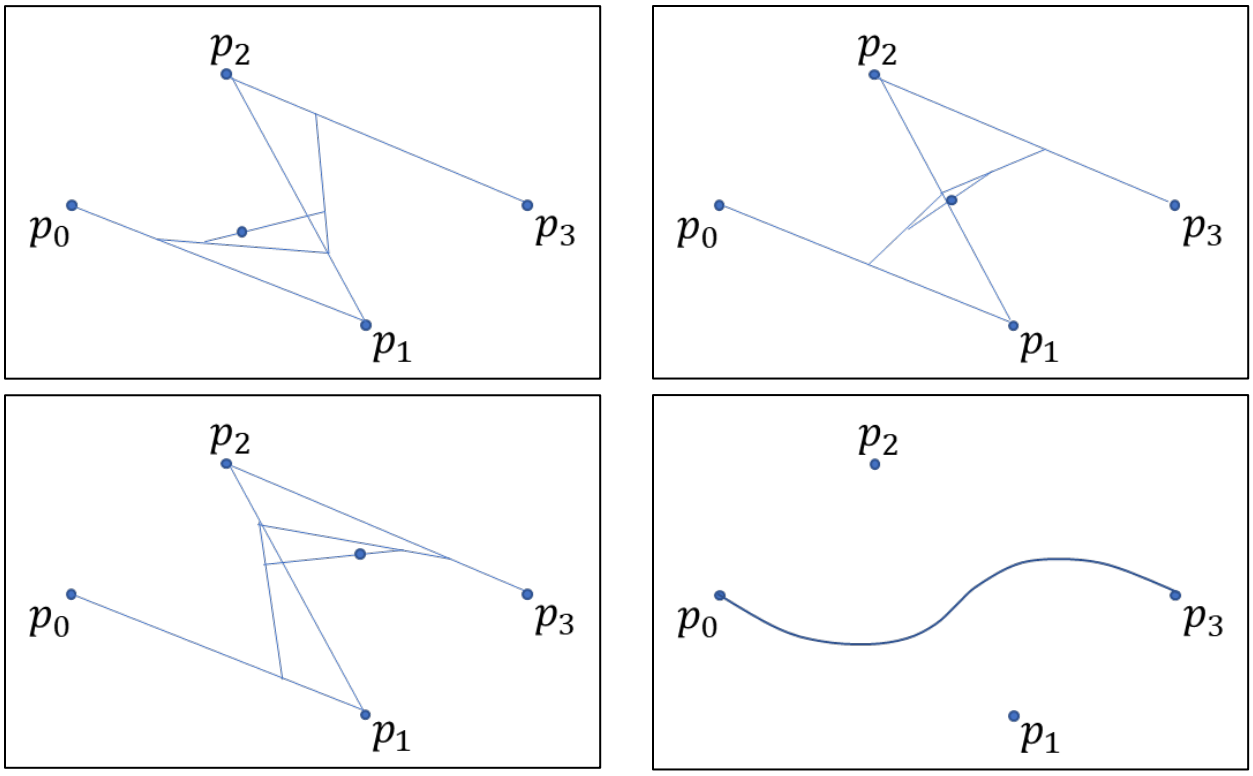
a.) Compute the point at $t = 0.7$ for a linear Bezier curve with control points $p_0 = (3,5)$ and $p_1 = (7,1)$.

$$p_{01}^x = 0.3 * 3 + 0.7 * 7 = 5.8$$
$$p_{01}^y = 0.3 * 5 + 0.7 * 1 = 2.2$$
$$p_{01} = (5.8,2.2)$$

b.) Compute the point at $t = 0.5$ for a quadratic Bezier curve with control points $p_0 = (2,2)$, $p_1 = (4,0)$ and $p_2 = (4,8)$.

$$p_{01}^x = 0.5^2 * 2 + 2 * 0.5 * 0.5 * 4 + 0.5^2 * 4 = 3.5$$
$$p_{01}^y = 0.5^2 * 2 + 2 * 0.5 * 0.5 * 0 + 0.5^2 * 8 = 2.5$$
$$p_{01} = (3.5,2.5)$$

c.) Visually find the points at $t = 0.25$, 0.5, and 0.75 for the control points below and sketch the resulting curve. Use one box for each t value, and the last box for the final curve.



d.) Why might splines be a better choice than single curves for creating some alphabetic letter shapes?

Some letters have sharp angles, which are hard to model with single curves. With a spline you can put the angle at a knot.

Part 5: Line Clipping (20 points)

In this problem, you are given a clipping window (defining the "viewport") and an example line, and asked to perform the steps of the Cohen-Sutherland line clipping algorithm. Assume the origin is at the top left and y is increasing going down (like HTML canvas).

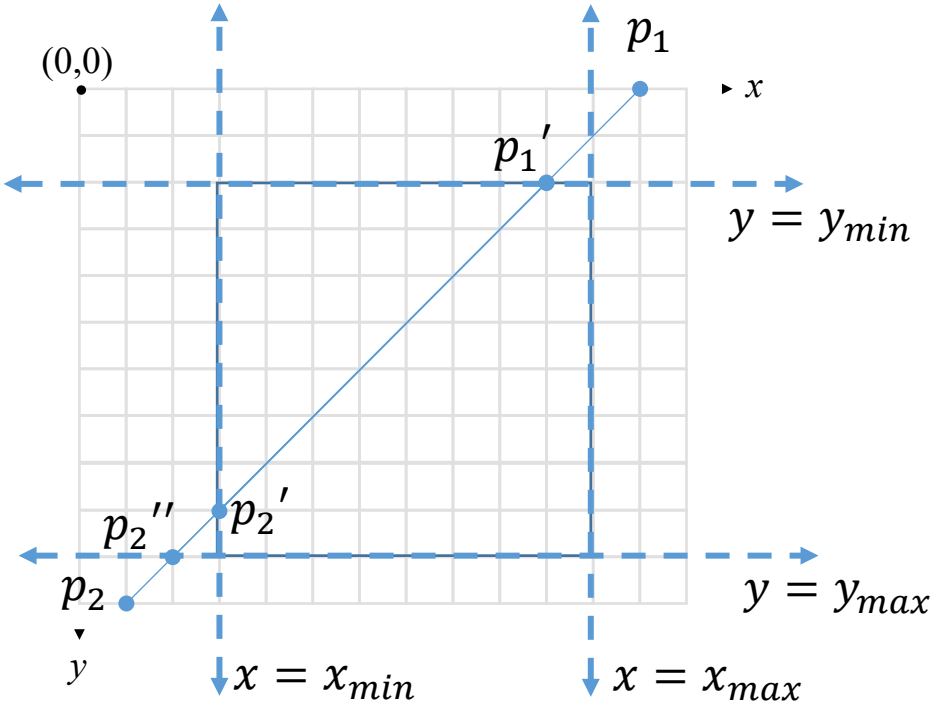
Input: viewport defined by the lines $x_{min} = 3$, $x_{max} = 11$, $y_{min} = 2$, $y_{max} = 10$. line defined by the points $p_1 = (12,0)$ and $p_2 = (1,11)$.

Output: p_1' and p_2', the points defining the line that should actually be drawn.

a.) Draw out the viewport and the line, labeling p_1, p_2, x_{min}, x_{max}, y_{min}, and y_{max}.

b.) Write out the binary 4-digit codes for p_1 and p_2.
p_1:1010; p_2: 0101

c.) Write out what case each point falls under and show how the algorithm would update the points. What are the final p_1' and p_2'?



$p_1' = (10,2)$
$p_2'' = (2,10)$
$p_2' = (3,9)$

d.) Label p_1' and p_2' on your picture and make sure they agree visually with your calculations. *See diagram*

e.) How many "rounds" of clipping are required to make this line fall within the viewport? *We clip p_1 just once, and p_2 twice. The fourth round we draw the line.*