CSC 240: Computer Graphics
Midterm: Fall 2018

Due: Wednesday, October 24 at 11:55 pm (on Moodle)

- This is a take-home exam with unlimited time from when it is out to when it is due.
- It is open-notes, so you may use any course materials. If you use any online resources that haven't been part of this class, please cite them explicitly.
- You may not communicate or consult about the exam with anyone in the class (or outside the class). However, you can email me if you need clarification.
- If there is a clarification I think should be made to the entire class, I'll post it on Piazza.
- I will still have office hours as usual, but I might not say much about the exam!
- Turn in your exam by scanning it and submitting on Moodle.
- If you are unable to make progress on any part of the exam, tell me what you tried: describe your thought process.
- 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. I have explicitly cited any resources used beyond my own notes and the materials available from the course web page."

Signed: Date:

Name:	
Part 1	/30
Part 2	/30
Part 3	/30
Part 4	/30
Part 5	/30
Total	/150

Part 1: Line Drawing (30 points)

If we want to draw a line between p0 = (0,2) and p1 = (4,4), the basic algorithm we have been following will do something like this:

```
for x = 0 to 4
    compute $y = \frac{1}{2}x + 2$
    ink(x,round(y))
endfor
```

a.) Suppose that instead of a line, we want to draw the curve $y = \frac{1}{8}x^2 + 2$. Would the same algorithm work well here, if we just substitute the new equation for the curve in place of the line? Why or why not?

b.) Suppose that we still want to draw a part of the curve $y = \frac{1}{8}x^2 + 2$ but now with a different pair of endpoints. Regardless of your answer in part (a), identify the range of *x* values for which the algorithm above would do a good job of drawing the curve. Explain how you arrived at your answer.

c.) Is there an alternative strategy for drawing the desired curve that will work outside the range of *x* values you identified for part (b)? Describe it, or explain why such a strategy does not exist.

Part 2: Fill Algorithms (30 points)

Consider the modified sweep fill algorithm presented below, and answer the questions that follow.

```
function sweepFill2(x, y, oldColor) {
    // fill in this scanline
    var rx = x;
    while (colorEqual(getPixel(rx,y),oldColor)) {
        fillPixel(rx,y);
        rx++;
    }
    var lx = x-1;
    while (colorEqual(getPixel(lx,y),oldColor)) {
        fillPixel(lx,y);
        lx--;
    }

    // now look above and below
    if (colorEqual(getPixel(x,y-1),oldColor)) {
        sweepFill2(x, y-1, oldColor);
    }
    if (colorEqual(getPixel(x,y+1),oldColor)) {
        sweepFill2(x, y+1, oldColor);
    }
}
```

a.) Will this function always completely fill an arbitrary polygon? Give an argument in support, or provide a counterexample.

b.) Will this function always completely fill an arbitrary convex polygon? Give an argument in support, or provide a counterexample.

c.) Will this function always completely fill an arbitrary rectangle? Give an argument in support, or provide a counterexample.

Part 3: Transforms (30 points)

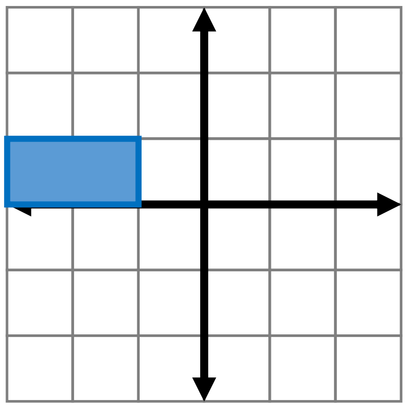
As you know, matrix transformations may be composed by multiplying two or more transformation matrices so as to form a new one. For this problem, your task is to construct a matrix that will transform the unit square with opposite corners at (0,0) and (1,1) into the shape shown. (Assume that the positive *y* axis points upwards in these figures.) The only catch is that you have to construct your transformation using only the limited set of building blocks shown below. You may use each of these as many times as you like, and multiply them in any order that you like. If you multiply by a matrix several times in a row, you may use exponents in your answer. For example, A^4B^2 = AAAABB

Transformation matrices you may use to build your result:

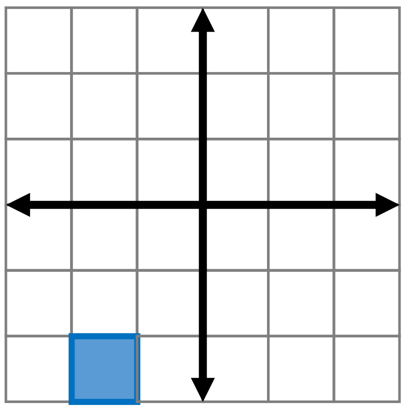
$$U = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}, R = \begin{bmatrix} \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} & 0 \\ \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & 0 \\ 0 & 0 & 1 \end{bmatrix}, S = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Target shapes:

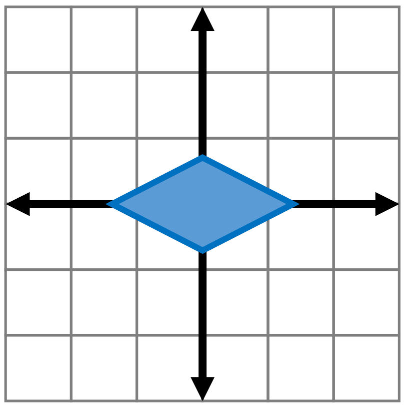
a.)



b.)



c.)



Part 4: **Bézier Curves** (30 points)

In class we looked briefly at the triangle fill algorithm, which uses a weighted sum of the three vertex colors to achieve a smooth shading effect known as Goraud shading. In a similar manner, we can draw smoothly colored Bézier curves. Each control point is assigned a color and the color of any point on the curve is computed as the weighted sum of the control point colors. The weights on the control points are parameterized by *t* in exactly the same way as the positions are when finding the location of the control point.

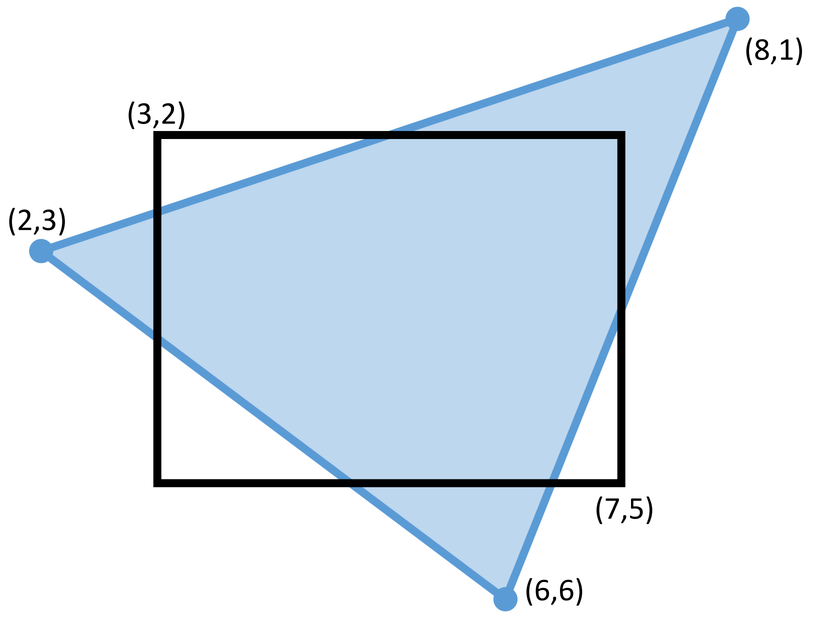
a.) Suppose that we have a quadratic curve, and the points are assigned colors as follows: p_0 = (128,0,0); p_1 = (0,128,0); p_2 = (0,0,128). Compute the blended colors that would be drawn at *t* = 0.25, *t* = 0.5, and *t* = 0.75.

b.) Assuming that colors are stored as arrays with three elements (red, green, and blue components), fill in an implementation for the function below:

```
// return the color of a point on the Bezier line between p0 and p1
// t is the position; c0 and c1 are the endpoint colors
function bezierLineColor(t, c0, c1) {
    // fill in here









}
```

Part 5: Line Clipping (30 points)

We briefly covered the Sutherland Hodgman algorithm for clipping polygons. Consider the blue polygon shown in the diagram below, with the viewport shown in black. **Compute and list** the full set of vertices that form the fully clipped polygon after Sutherland Hodgman has been applied. (*Hint: Use what you already know about line clipping.*)


(8,1)
(3,2)
(2,3)
(7,5)
(6,6)