CSC 240: Computer Graphics
Midterm: Fall 2019
Due: Wednesday, October 30 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	/20
Part 2	/20
Part 3	/20
Part 4	/20
Part 5	/20
Total	/100

Part 1: Line Drawing (20 points)

Suppose that we would like to adapt our basic line-drawing algorithm to draw circles. The equation of circle is $(x - x_c)^2 + (y - y_c)^2 = r^2$ where (x_c, y_c) is the center and r is the radius. This can be solved for either x or y as necessary:

$$x = x_c \pm \sqrt{r^2 - (y - y_c)^2}$$
$$y = y_c \pm \sqrt{r^2 - (x - x_c)^2}$$

a.) Explain how our basic line-drawing algorithm could be adapted to draw a circle by splitting it into four segments and using the equations above.

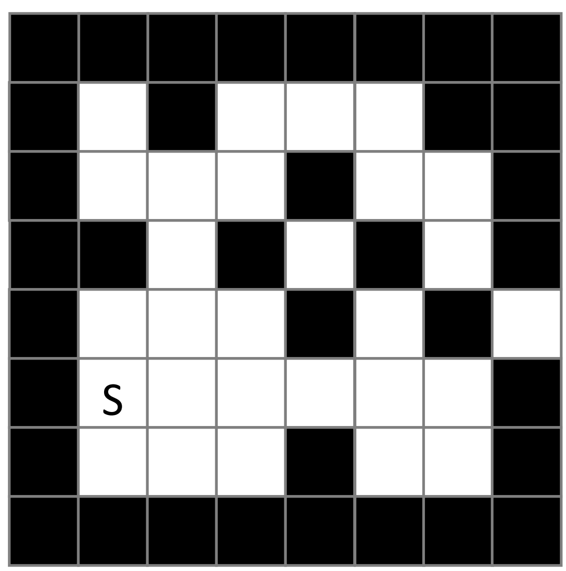
b.) How exactly should the circle be split into segments? Identify the endpoints, and explain why they make sense.

Part 2: Fill Algorithms (20 points)

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

```
function floodFill(x, y, oldColor) {
    // note that this returns [R,G,B,A]
    var pixColor = getPixel(x,y);
    if (colorEqual(oldColor,pixColor)) {
        fillPixel(x,y);
        floodFill(x+1, y, oldColor);
        floodFill(x-1, y, oldColor);
        floodFill(x, y+1, oldColor);
        floodFill(x, y-1, oldColor);
    }
}
```

a.) Number the pixels in the figure in the order they would be colored by this function, starting at the square indicated by the letter S. Assume that the positive y axis points downwards.



b.) Write a simple modification to the function so that it performs an 8-connected fill.

Part 3: Transforms (20 points)

In Homework #3 we learned that some transformation types don't commute. For example, in general translation and rotation do not commute with each other; $TR \neq RT$. However, that does not mean that we cannot find some other translation to be performed after the rotation that will give us the same end result.

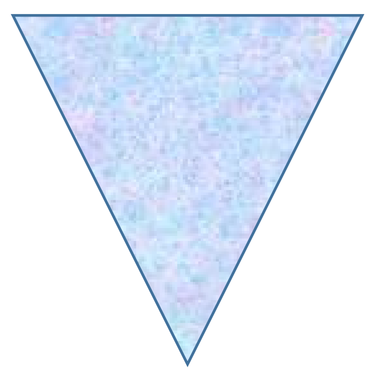
a.) Let $R = \begin{bmatrix} 0.8 & -0.6 & 0 \\ 0.6 & 0.8 & 0 \\ 0 & 0 & 1 \end{bmatrix}$ and $T = \begin{bmatrix} 1 & 0 & -10 \\ 0 & 1 & 15 \\ 0 & 0 & 1 \end{bmatrix}$. Find a new translation matrix $U = \begin{bmatrix} 1 & 0 & u_x \\ 0 & 1 & u_y \\ 0 & 0 & 1 \end{bmatrix}$ such that $TR = RU$.

b.) Suppose that $T = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$ and $S = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$. Find a new translation matrix $U = \begin{bmatrix} 1 & 0 & u_x \\ 0 & 1 & u_y \\ 0 & 0 & 1 \end{bmatrix}$ such that $TS = SU$. (Answer in terms of t_x, t_y, s_x, and s_y.)

,

Part 4: Line Clipping (20 points)

Suppose that the Delta Display Corporation has just come out with a new product: triangular screens! You have been asked to adapt the Cohen-Sutherland line clipping algorithm for their new Delta3 display. It has the shape of an inverted isosceles triangle, 1000 pixels wide at the top and 1000 pixels tall. The screen area is bounded by the lines $y = 0$, $y = 2x$, and $y = 2000 - 2x$. The positive y axis points downwards.



a.) Propose a region-labeling technque similar to the 4-bit labels used by Cohen-Sutherland. Give a specific list of steps used to determine whether a line segment needs to be clipped. How does your method differ from Cohen-Sutherland?

b.) Demonstrate how your method would clip the line segment with endpoints (100,-200) and (800,1200). Show both your work and your final answer.

Part 5: Bézier Curves and Splines (20 points)

Identify any errors in computing the Bézier curve points as stated below. If there are no errors, indicate that the sample is error-free. (For each item, you may assume that all calls to subfunctions such as `bezier1`, `bezier2`, etc. return correct values.)

a.)
```
// computes a linear Bezier point
function bezier1a(t, p0, p1) {
    p = [(1-t)*p0[0]+t*p0[1], (1-t)*p1[0]+t*p1[1]];
    return p;
}
```

b.)
```
// computes a quadratic Bezier point
function bezier2b(t, p0, p1, p2) {
    q1 = bezier1(t, p0, p1);
    q2 = bezier1(t, q1, p2);
    p = bezier1(t, p1, q2);
    return p;
}
```

c.)
```
// computes a quadratic Bezier point
function bezier2c(t, p0, p1, p2) {
    q1 = bezier1(1-t, p1, p0);
    q2 = bezier1(1-t, p2, p1);
    p = bezier1(1-t, q2, q1);
    return p;
}
```

d.)
```
// computes a cubic Bezier point
function bezier3d(t, p0, p1, p2, p3) {
    q1 = bezier2(t, p0, p1, p2);
    q2 = bezier1(t, p1, p2);
    q3 = bezier1(t, p2, p3);
    q4 = bezier1(t, q2, q3);
    p = bezier1(t, q1, q4);
    return p;
}
```