CSC 240: Computer Graphics
Midterm SOLUTION: 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:

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

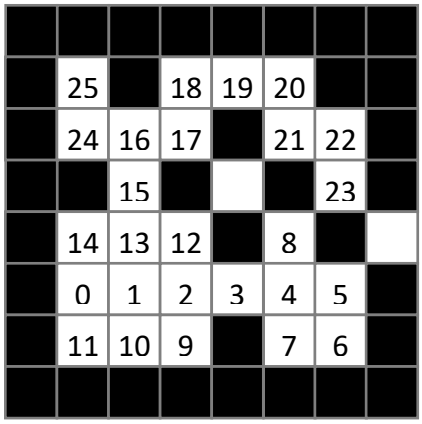
a.) Explain how our basic line-drawing algorithm could be adapted to draw a circle by splitting it into four segments and using the equations above.
The first equation gives you x in terms of y, so you could loop over the y values and compute the corresponding x. The second gives you y in terms of x, so you could loop over the x values and compute y.

b.) How exactly should the circle be split into segments? Identify the endpoints.
The circle should be split at the diagonal corners, $(x_c \pm \sqrt{r}\,, y_c \pm \sqrt{r})$ because these are the points where the slope changes from less than 1 to more than one. The horizontal sections will be filled in by looping from $x_c - \sqrt{r}$ to $x_c + \sqrt{r}$ using the equation(s) for y in terms of x, and the vertical sections will be filled in by loopiong from $y_c - \sqrt{r}$ to $y_c + \sqrt{r}$ using the equation(s) for x in terms of y. Note that the choice of $\pm$ as positive or negative corresponds to which part of the circle is drawn.

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.
See diagram.

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

Add the following lines right after the four calls to floodFill:

```
floodFill(x+1, y+1, oldColor);
floodFill(x-1, y-1, oldColor);
floodFill(x-1, y+1, oldColor);
floodFill(x+1, y-1, oldColor);
```

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 there exists no combination of rotation followed by translation 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 $TR = RU$.

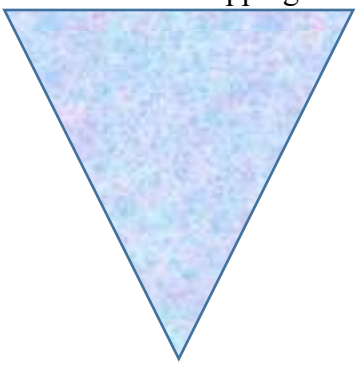
$u_x = 1$ and $u_y = 18$

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 $TS = SU$.

$u_x = t_x/s_x$ and $u_y = t_y/s_y$

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 thus bounded by the lines $y = 0$, $y = 2x$, and $y = 2000 - 2x$.



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?

We will need a three-bit code because there are three bounding lines on our screen. Let's order them top, left, right. The first bit will be 1 iff y<0. The second bit will be 1 iff y>2x. The third bit will be 1 iff y>2000-2x. As with Cohen-Sutherland, comparing the bit strings of the endpoints reveals whether the segment needs to be analyzed more closely.

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.

Code for first is 100. Code for second is 001. Must perform intersections. Equation of our line is $y + 200 = \frac{1200-(-200)}{800-100}(x - 100)$, *or* $y = 2x - 400$.

Intersection with $y = 0$*: x = 200*

Intersection with $y = 2000 - 2x$*: x = 600, y = 800*

Cropped segment goes from (200,0) to (600,800)

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;
}
```

Incorrect combination of variable names and subscripts; should be

```
p = [(1-t)*p0[0]+t*p1[0], (1-t)*p0[1]+t*p1[1]];
```

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, pq);
    return p;
}
```

Incorrect; q1 used in place of p1 and vice versa

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;
```

```
    }
```
Correct; inversion of point order compensates for use of 1-t.

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;
}
```
Correct; quadratic bezier can be computed directly or built according to its definition.