CSC 240: Computer Graphics

Midterm: Fall 2016

Completed by: Sunday, October 30 at 4pm

- This exam is to be taken in the Young Science Library during any of their open hours.
- The time limit is **2 hours** unless you received an email saying otherwise. I will be checking all in/out time stamps.
- No communication about the exam with anyone in the class (or outside the class).
- No electronic devices are to be used during the exam, but you may use a 2-sided cheat sheet. Your cheat sheet should be handwritten and created by you.
- Discussing the exam, going over the time limit, and using electronic devices are all honor code violations.
- The coding question is **not** to be done during these 2 hours. You have unlimited time (until the deadline) outside of the exam for that part (turn in on Moodle).
- Make sure all your work is contained on these pages (writing on the backs is okay).
- If you are unable to make progress on any part of the exam, tell me what you tried; describe your thought process.

Name	

Part 1	/20
Part 2	/20
Part 3	/20
Part 4	/20
Part 5	/20
Total	/100

Part 1: Short Answer

(a) What is the main difference between raster graphics and vector graphics?

(b) Consider a single point at the origin, $p = (0, 0)$. Out of the transformations we have studied (scale, translate, rotate, shear, and reflect), which have the ability to move p off of the origin?

(c) What is wrong with the rotation matrix below?

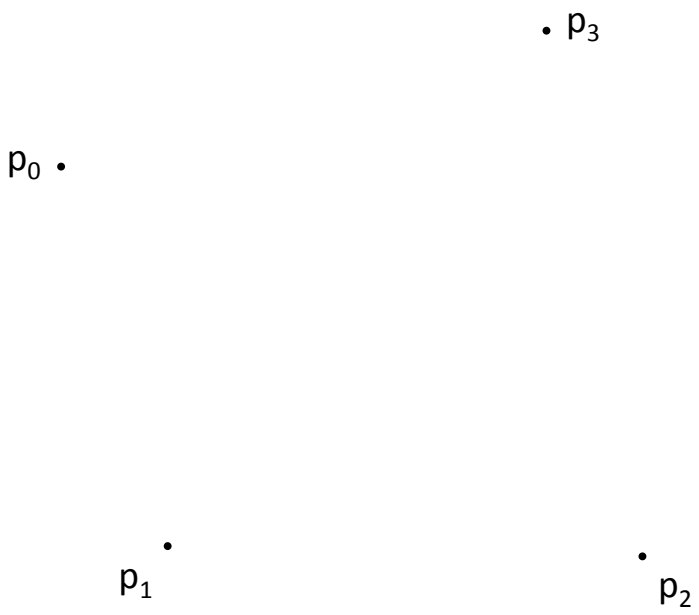
$$R_\theta = \begin{bmatrix} \cos\theta & -\sin\alpha \\ \sin\alpha & \cos\theta \end{bmatrix}$$

(d) Write some short pseudocode for the function below, which should return the point corresponding to t on a quintic (4^{th} order) Bézier curve:

```
function bezierQuinticHelper(t,p0,p1,p2,p3,p4) {


}
```

Your solution should make optimal use of lower order Bézier curve functions (assume you have access to all the functions you implemented in Homework 5).

(e) Draw the Bézier curve for the ordered set of control points below. Label/utilize the points on the curve when $t = 0$, $1/4$, $1/2$, $3/4$, and 1.



Part 2: Polygons and Fill

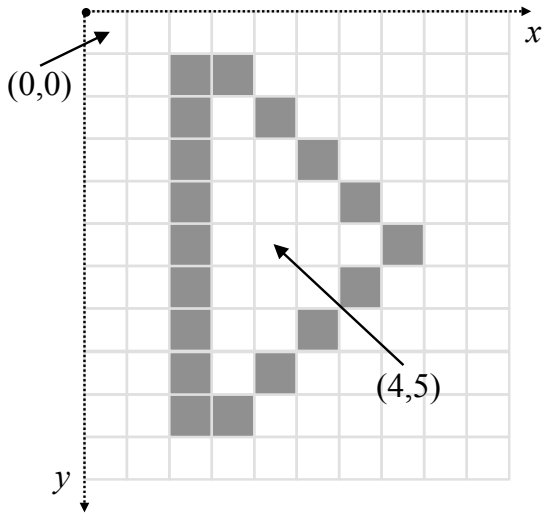
A fellow student is attempting to create a recursive framework that can fill larger shapes, and gives you the following code:

```
function floodFillNorthEast(x, y, oldColor) {
    var currColor = graphics.getImageData(x, y, 1, 1).data;

    if (colorEqual(currColor, oldColor)) {
        graphics.fillRect(x, y, 1, 1);
        floodFillNorthEast(x, y-1, oldColor);
        floodFillNorthEast(x+1, y, oldColor);
    }
}

function floodFillSouthWest(x, y, oldColor) {
    var currColor = graphics.getImageData(x, y, 1, 1).data;

    if (colorEqual(currColor, oldColor)) {
        graphics.fillRect(x, y, 1, 1);
        floodFillSouthWest(x, y+1, oldColor);
        floodFillSouthWest(x-1, y, oldColor);
    }
}
```

(a) Inside the `draw()` function, a triangle is created, along with the following calls to the flood fill methods above:

```
graphics.fillStyle = "purple";
floodFillNorthEast(4, 5, "white");
floodFillSouthWest(4, 6, "white");
```

Assuming `colorEqual()` can handle any type of inputs correctly and that y is increasing going down the screen, shade the pixels of the triangle below that are colored by this approach.



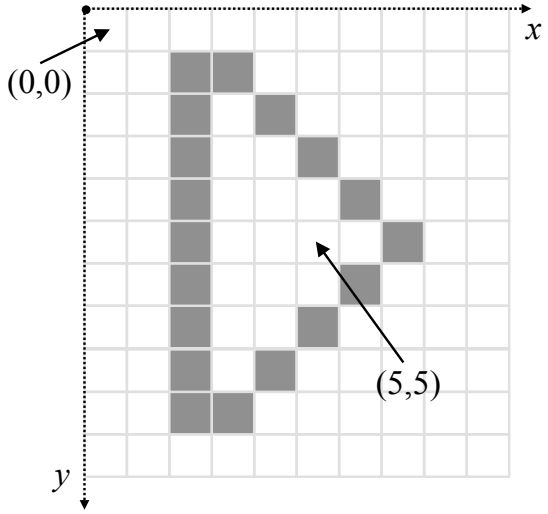
(b) Seeing the limitations of this approach, you suggest the following modifications:

```
function floodFillRight(x, y, oldColor) {
    var currColor = graphics.getImageData(x, y, 1, 1).data;

    if (colorEqual(currColor, oldColor)) {
        graphics.fillRect(x, y, 1, 1);
        floodFillRight(x, y-1, oldColor);
        floodFillRight(x+1, y, oldColor);
        floodFillRight(x, y+1, oldColor);
    }
}

function floodFillLeft(x, y, oldColor) {
    var currColor = graphics.getImageData(x, y, 1, 1).data;

    if (colorEqual(currColor, oldColor)) {
        graphics.fillRect(x, y, 1, 1);
        floodFillLeft(x, y+1, oldColor);
        floodFillLeft(x-1, y, oldColor);
        floodFillLeft(x, y-1, oldColor);
    }
}
```

Using this approach and the following modifications inside draw:

```
graphics.fillStyle = "purple";
floodFillRight(5, 5, "white");
floodFillLeft(4, 5, "white");
```

show the order that the pixels are filled in the same shape below. Use "A" for the first pixel, "B" for the second pixel, etc.



(c) Will this modification work for any regular polygon? What about any non-convex polygon? Briefly justify your answers.

(d) If instead you had used the calls:

```
graphics.fillStyle = "purple";
floodFillRight(5, 5, "white");
floodFillLeft(5, 5, "white");
```

what would have happened? Describe the problem with these function calls.

Part 3: Transformations

(a) Draw a triangle with vertices $p = (0, 0)$, $q = (2, 0)$, and $r = (1, 2)$. Write out the matrix multiplication problem that will scale this triangle by 2 in the x direction and 0.5 in the y direction. Perform this matrix multiplication and draw the scaled triangle, confirming that your picture and result matrix match. You may use either a 2x2 or 3x3 scale matrix.

(b) Then translate the resulting (scaled) triangle above by -3 in the x direction and 3 in the y direction. Again write out a matrix multiplication problem to demonstrate this result. Here we must use a 3x3 translation matrix. Draw a picture of this result (or clearly label it on your picture above), and confirm that it agrees with your result matrix.

(c) To achieve the result of the steps in (a) and (b) in JavaScript/HTML Canvas, what steps could you use? Write a few lines of pseudocode below to demonstrate this. Assume we have a function that will draw a triangle.

(d) Do scalings commute with reflections across the y-axis? That is, for a scale matrix S and a reflection matrix F (across the y-axis), is it always true that $SF = FS$? If yes, provide a proof using general variables and matrix multiplication. If no, provide a concrete counter example with a picture/graph.

Part 4: Graphics in JavaScript

Do not do this now! These same instructions are duplicated on the course website, just leave this page blank as a placeholder.

For this part you are asked to write one animation in JavaScript/HTML Canvas, starting from the code `animation.html` on the course website.

When you open this animation, you should see a diamond shape. The goal of this animation is to make a square move along this diamond in a particular way.

(a) First uncomment the `graphics.fillRect(?,?,?,?);` line and fill in the question marks to make the square appear at the top of the diamond. Use the `sideLength` variable defined at the top of the code. In essence, you are writing code for a square *centered at the origin*, but because of the initial translate in `draw()`, the square appears at the top of the diamond. For the rest of the code, this is the only way you can call `fillRect` to create a square.

(b) Use a call to `graphics.translate(tx,ty)` to make the square gradually move along the upper left side of the diamond. This is labeled *large* to indicate the square is its initial size.

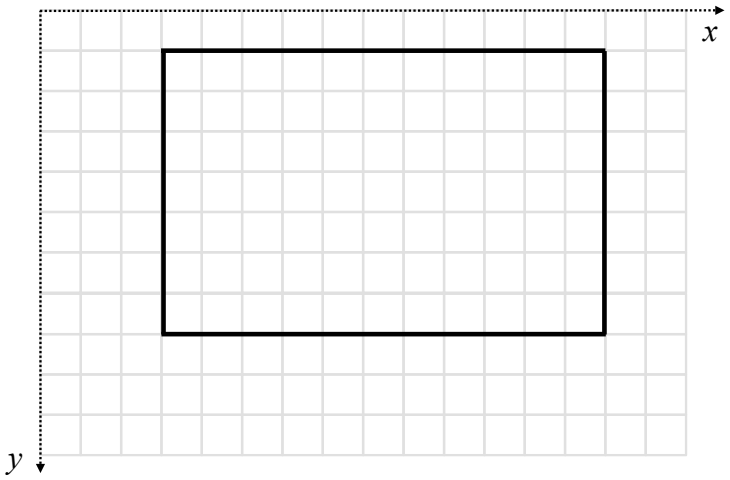
(c) When the square reaches the leftmost point of the diamond, again use translate to move it along the next side of the diamond. Complete the entire diamond in this way, then reset so that the square continues to move around the diamond indefinitely.

(d) Finally, modify your code so that the square is smaller along the sides marked *small*, and its original size along the sides marked *large*. Use `graphics.scale(ax,ay)`.

Part 5: Lines

In this part we will explore a line clipping method based on Bézier lines. Show all your work to receive full credit. (Even though this question involves line clipping, the concepts are largely about Bézier lines.)

(a) First, plot the points $p_0 = (2, 4)$ and $p_1 = (12, 9)$ on the axes below, assuming y is increasing going down the page. Note that the viewport (black lines) is defined by the lines: $x_{\min} = 3$, $x_{\max} = 14$, $y_{\min} = 1$, and $y_{\max} = 8$. Draw the line segment connecting p_0 and p_1.



(b) Now we will clip this line segment to p_0' and p_1' so that that line between p_0' and p_1' is fully within the viewport. To do this with a Bézier line, write out the Bézier line equations for $x(t)$ and $y(t)$ between p_0 (beginning of the line) and p_1 (end of line). Label $t = 0$ and $t = 1$.

(c) Next, consider clipping p_1. We need to find the intersection of our desired line and which viewport line? Plug in this viewport value into the appropriate equation above and solve for the value of t at this intersection point. Label this value of t on your graph above.

(d) Finally, use this value of t in your equations from part (b) to obtain the point p'_1. Show all your work. Does this agree with your graph?

(e) Now to clip p_0, find the t that corresponds to the intersection of our Bézier line and the appropriate viewport line. Label this t value as well.

(f) If you have time (no credit): clip p_0 and find p'_0.