A PDF that contains the expected results, along with any descriptions you feel may help in the understanding/interpretation of your results.

**Understanding:**

According to my understanding, the koch snowman starts with an equilateral triangle. For every iteration, each line is replaced with a new one that is split into 4 so that a new equilateral triangle pokes out of the middle. This can go on infinitely, creating the snowflake shape.

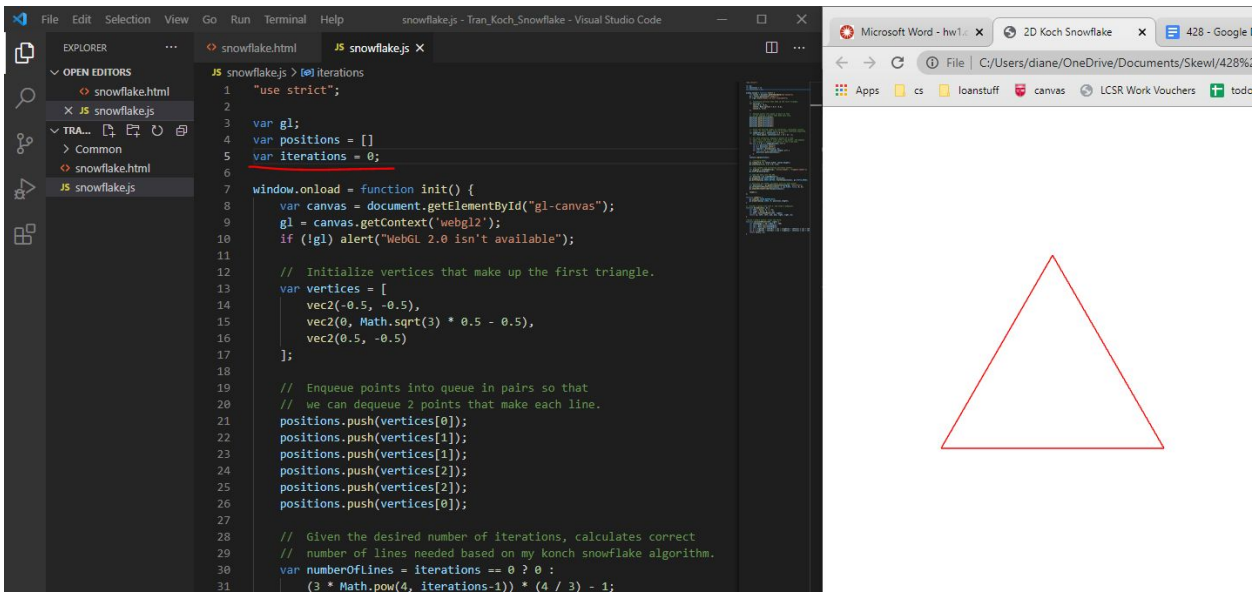
**Algorithm:**

I used a queue to keep track of the sets of vertices between each line. So if we had 3 vertices, [A, B, C], it would be stored in a queue as [A, B, B, C, C, A]. I dequeue 2 vertices at a time to grab the line I want to replace. The first would be the line A, B, then B,C, then C,A. As I create new lines, I follow the same pattern adding in new pairs of vertices to the queue.

First, a line is split into 3 lines and we get 2 new vertices at the 1/3 and 2/3 mark. Then the two new vertices are rotated to get the final middle vertex, and all 5 vertices are ordered into the new line and returned to be enqueued.

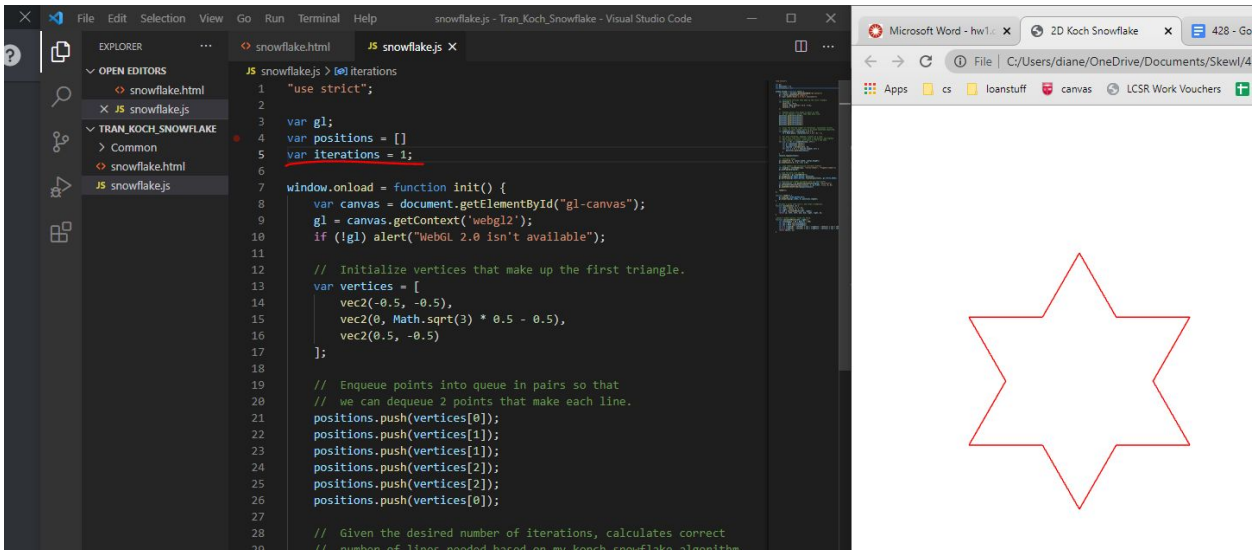
This algorithm's loop will replace one line for each iteration. I adjusted the number of lines/loops based on iterations right before the loop to create our koch snowflake shown below.

**Iterations 0:**

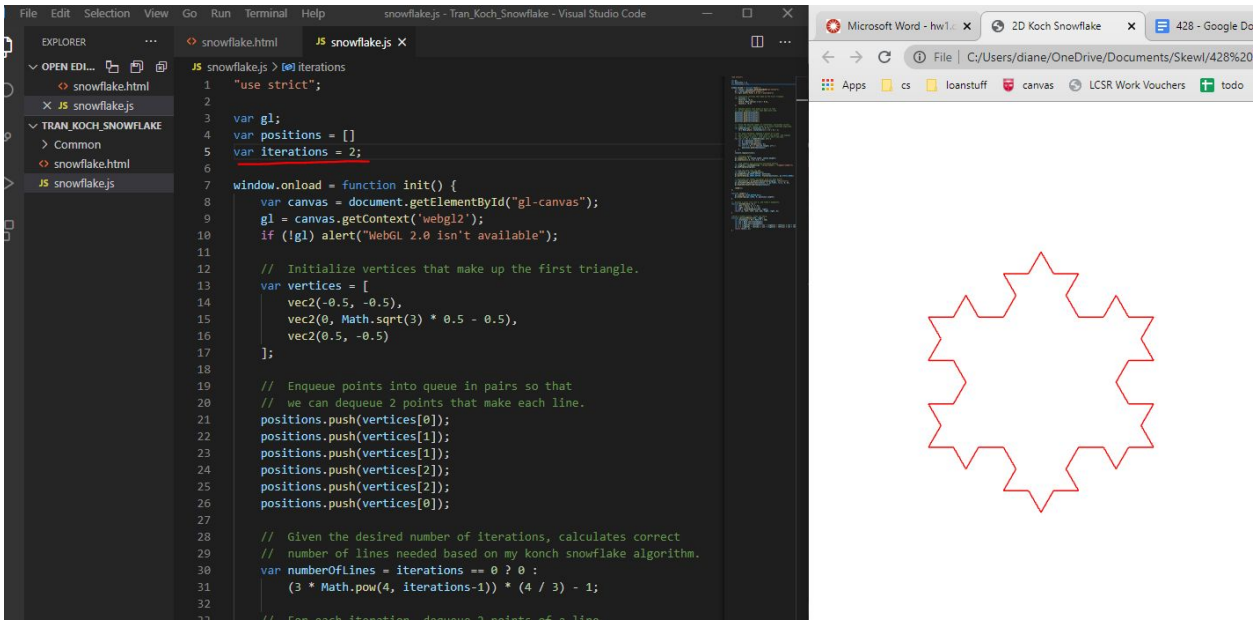


Diane Tran 198:428

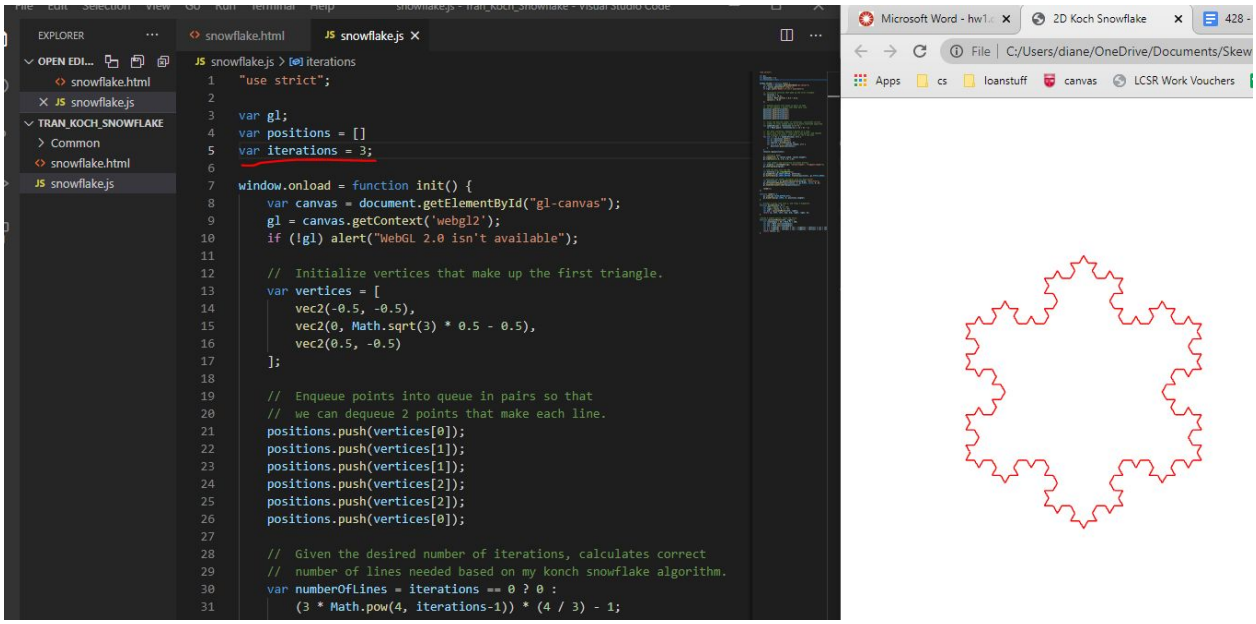
## Iterations 1:



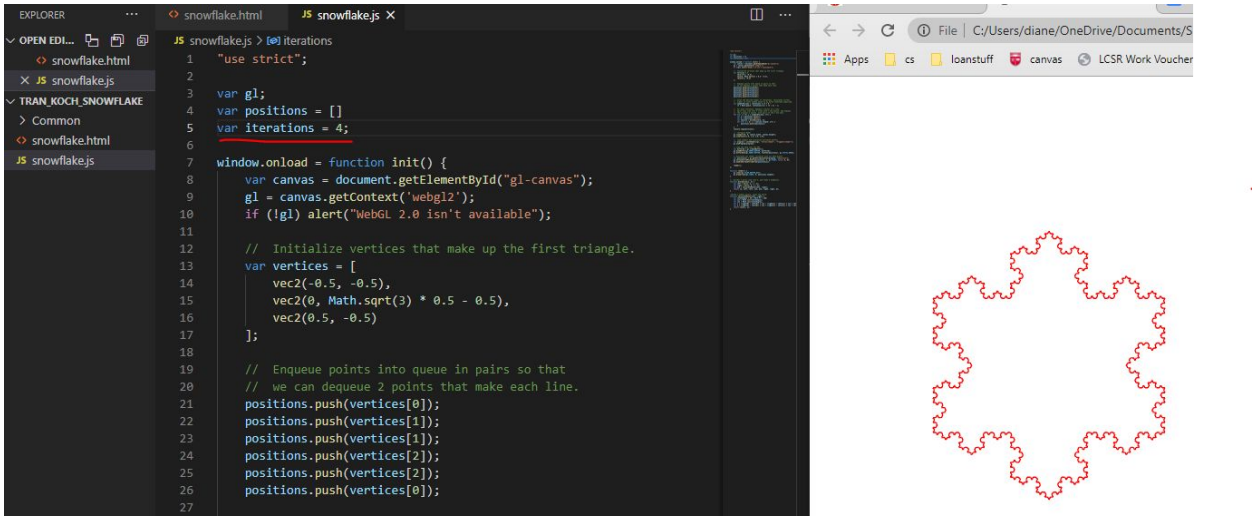

## Iterations 2:



## Iterations 3:

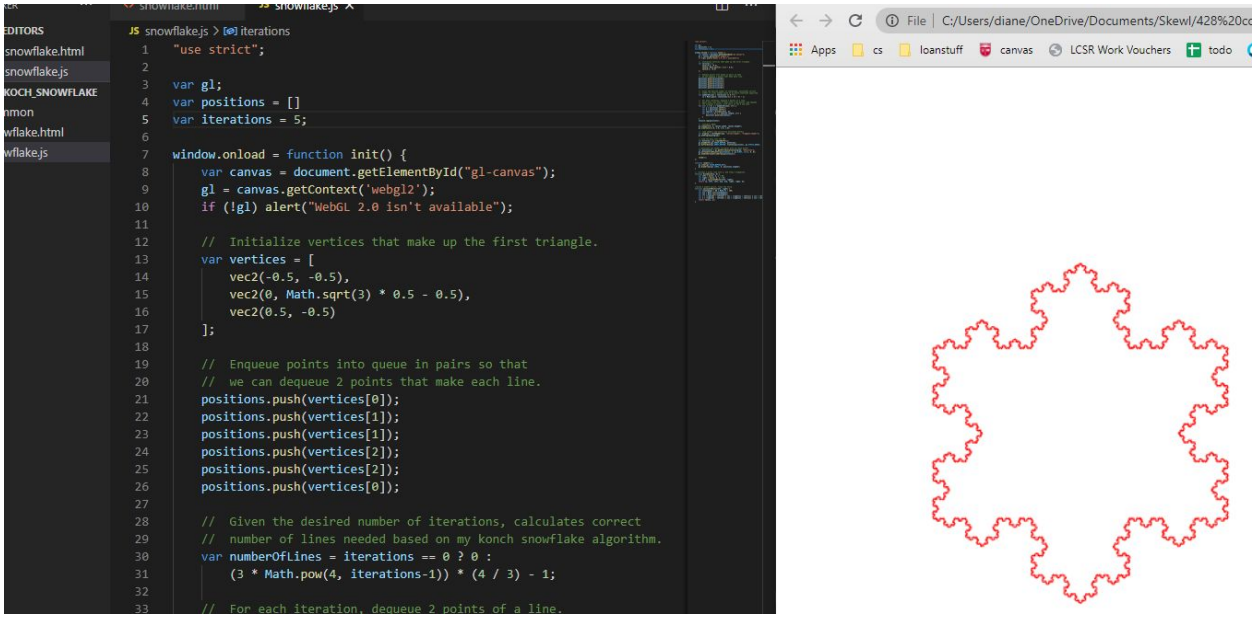



## Iterations 4:

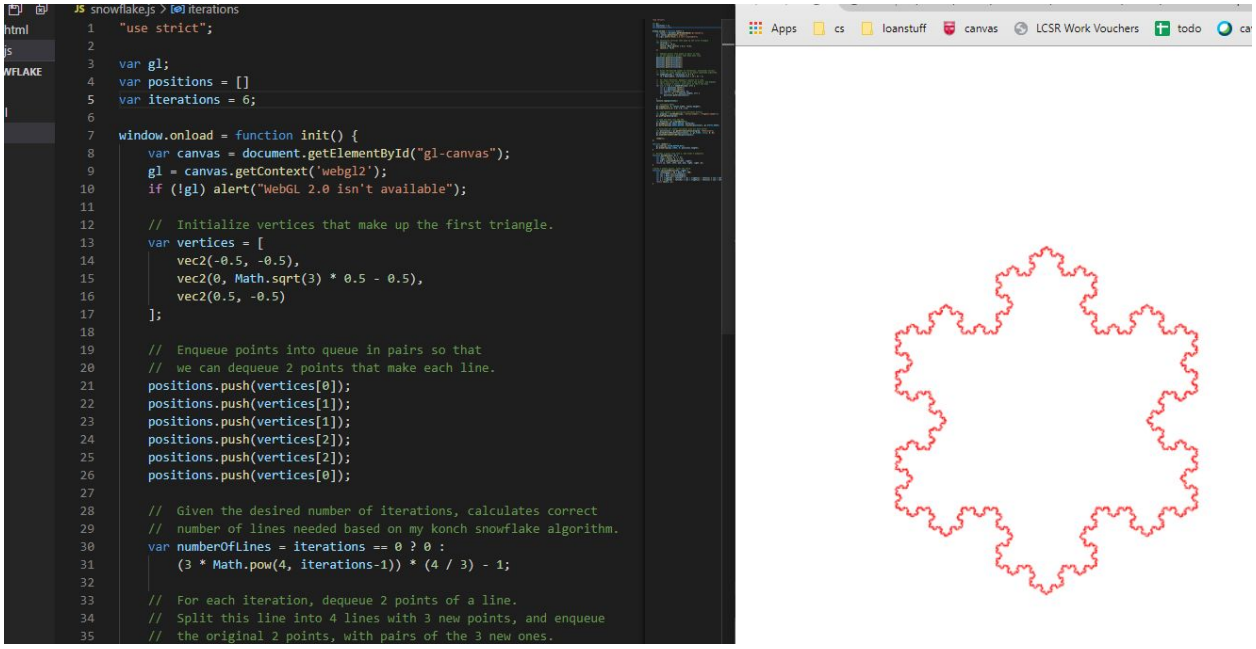


Diane Tran 198:428

## Iterations 5:

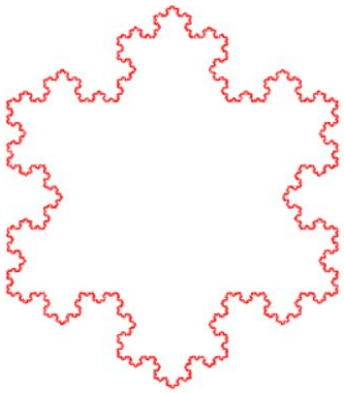



## Iterations 6:



**Iterations 7:**

```
var gl;
var positions = []
var iterations = 7;

window.onload = function init() {
    var canvas = document.getElementById("gl-canvas");
    gl = canvas.getContext('webgl2');
    if (!gl) alert("WebGL 2.0 isn't available");

    //  Initialize vertices that make up the first triangle.
    var vertices = [
        vec2(-0.5, -0.5),
        vec2(0, Math.sqrt(3) * 0.5 - 0.5),
        vec2(0.5, -0.5)
    ];

    //  Enqueue points into queue in pairs so that
    //  we can dequeue 2 points that make each line.
    positions.push(vertices[0]);
    positions.push(vertices[1]);
    positions.push(vertices[1]);
    positions.push(vertices[2]);
    positions.push(vertices[2]);
    positions.push(vertices[0]);

    //  Given the desired number of iterations, calculates correct
    //  number of lines needed based on my konch snowflake algorithm.
    var numberOfLines = iterations == 0 ? 0 :
        (3 * Math.pow(4, iterations-1)) * (4 / 3) - 1;

    //  For each iteration, dequeue 2 points of a line.
    //  Split this line into 4 lines with 3 new points, and enqueue
    //  the original 2 points, with pairs of the 3 new ones.
```



**After 7 iterations, there is no visible difference.**