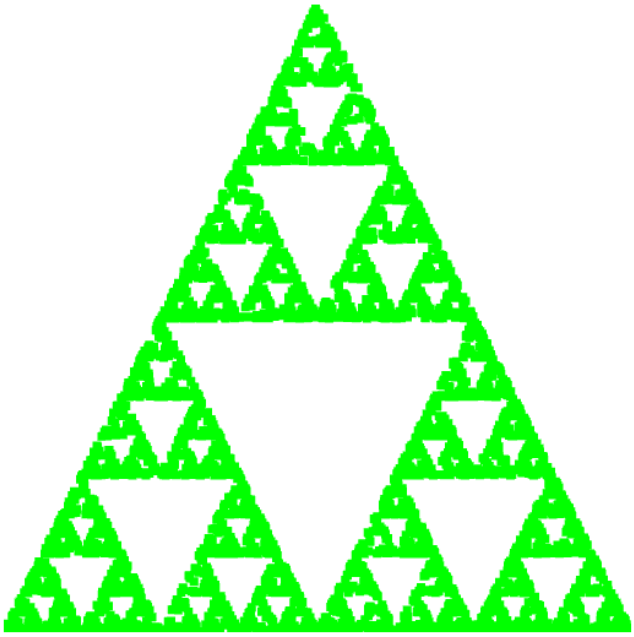
Q1:

I have changed the fragment shader color to draw the points as green.

```
gl_FragColor = vec4( 0.0, 1.0, 0.0, 1.0 );
gl_PointSize = 7.0;
```



Q2:

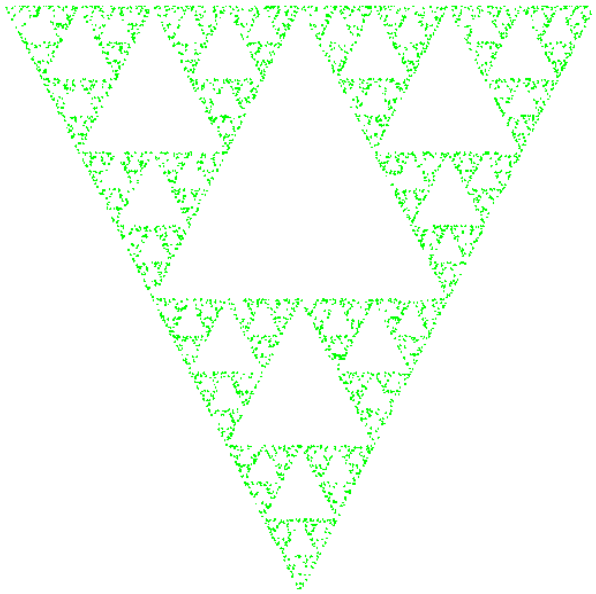
```
var vertices = [
  // vec2( -1, -1 ),
  // vec2(  0,  1 ),
  // vec2(  1, -1 )

  vec2(-1,  1),
  vec2( 1,  1),
  vec2( 0, -1)
];
```

Another way we can flip the drawing is to flip each point after calculating it. Even without changing the existing calculation code, we can add this after the original point calculationst:

```
let flipped_points = [];
for (let point of points) {
  let flipped_point = vec2(point[0], -point[1]);
```

```
  flipped_points.push(flipped_point);
}
```

to get a new array of the points in the original array flipped over the x-axis. Then we can just pass the flipped array to the gl.ARRAY_BUFFER instead.

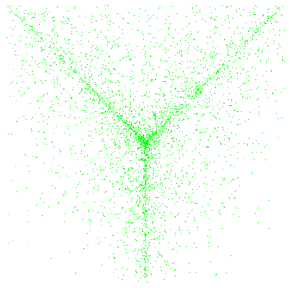
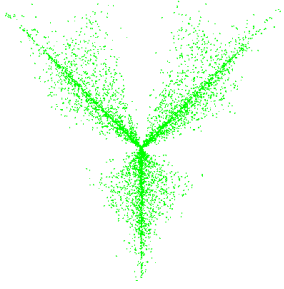


Q3:

If our code is not in the onload function, nothing will actually call it. We need an entry point for the program to start running, and it makes the most sense to do this when loading the page.

Q4:

(I messed with the brightness and contrast of the images a bit to make them easier to see)

 Math.random()	 getRandomArbitrary (0, 0.5)	 getRandomArbitrary (0.4, 0.5)	 getRandomArbitrary (0.1, 0.2)

Q5:

The 2 scale calls ensure the resulting points are scaled to properly fit on the screen. So the first scale is scale(0.25, p), because we are first adding 4 points to get p, so scaling by 0.25 is taking the average of the 4 values, which will always be another valid point -1 to 1, given each previous point was -1 to 1.
The second scale is scale(0.5, p), and does the same thing. This time we are only adding 2 points to get p, and thus the average is half of their sum. Essentially, these scales ensure that each set of points is scaled proportionally to the screen, and will fit on the screen where it is meant to be.
Any other value for the scale on the second scale() call alters the size of the 'recursive' triangles, since we are no longer properly scaling them based on the number of points used.

Q6:

This line retrieves the location of the vertices in the shader program.

```
var vPosition = gl.getAttribLocation( program, "vPosition" );
```

This line calls a function with quite a few parameters, but all it is doing is setting how the vertices will be read and interpreted when drawing later. So we are setting vPosition to read 2d points that are floats. (Also that it should not be normalized, with no stride and no offset.

```
gl.vertexAttribPointer( vPosition, 2, gl.FLOAT, false, 0, 0 );
```

I believe this is just setting the array to be rendered by WebGl.

```
gl.enableVertexAttribArray( vPosition );
```

Here is my website. Please don’t make fun of it, I have no web experience lol.
It’s at https://gammawyvern.github.io/CIS367/