1. The two lines of code changed (after changes)

// Larger Points in Vertex Shader

gl\_PointSize = 15.0;

// Green Fragment Shader

gl\_FragColor = vec4( 0.0, 1.0, 0.0, 1.0 );

The values changed were the floating point number assigned to gl\_Pointsize. This was raised from a size of 1.0 to 15.0. The other value changed was the vec4 green component. The FragColor vec4 was changed from (0.0, 0.0, 0.0, 1.0) to (0.0, 1.0, 0.0, 1.0) making the fragment shader fill green.

1. After changes to the vertices.

a)

var vertices = [

vec2( -1, 1 ),

vec2( 1, 1 ),

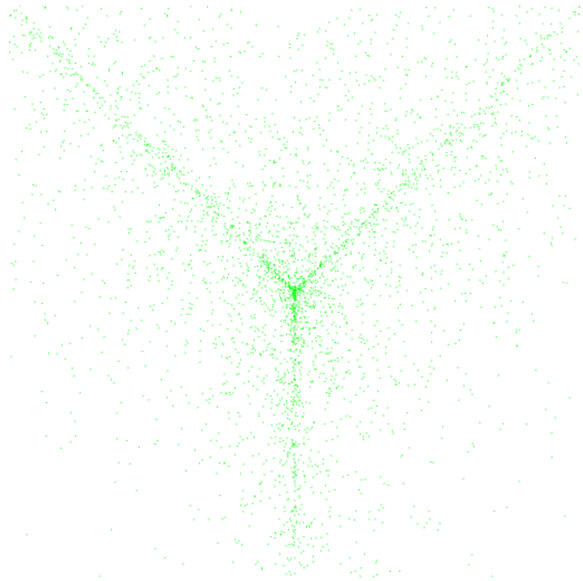
vec2( 0, -1 )

];

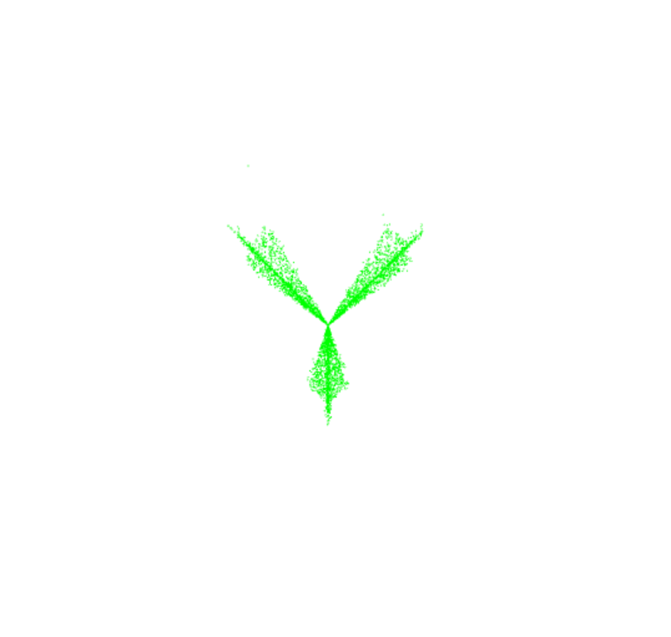
b) Another way to rotate the triangle could be to multiply the values inside of the vec2() used to make each vertex a number that would put them between -1 and 1 (after they are put into the vertices array). For example, you could do something like Q4 where you randomize both x and y values to be between -1 to 1. That is, if you want the triangle vertices to be in a different spot than originally assigned. There are a couple other possibilities with 2d Matrices too that allow you to set the triangle’s rotation angle and modify the vertices accordingly by multiplying by some factor (set by sin and cosine).

1. Our code needs to be placed inside of the window.onload() function because this ensures that the js script is executed after the html content has loaded to the screen. This includes making sure the canvas has loaded before attempting to output. The order of execution is important. HTML before the script. The onload() is executed after all code is loaded into memory.
2. Below are screenshots changing the scale values.

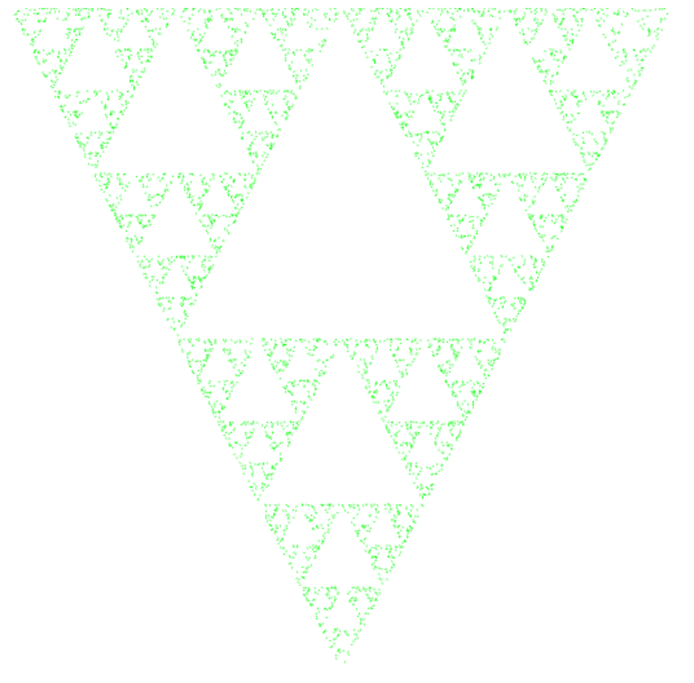
Result from Math.Random()



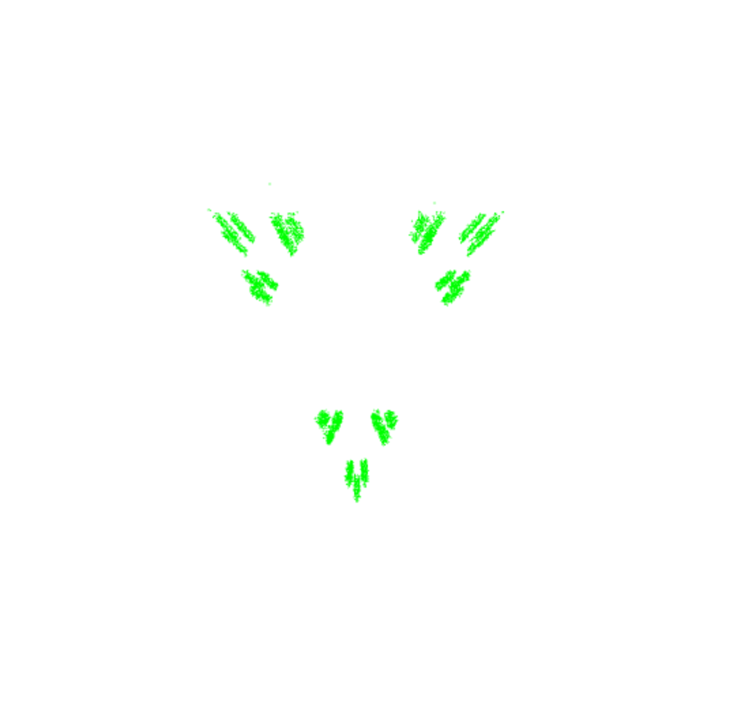
Result from getRandomArbitrary(0.0,0.25)



Result from getRandomArbitrary(0.5,0.75)



Result from getRandomArbitrary(0.25, 0.3)



1. In the second scale() function, we are scaling the vector p by a scale value of our choosing. If we were to choose a value of 0.5 as the first parameter, the distance from the previous point to the new point is halved. Basically, the smaller the value is, the closer the points will be together, and vice versa. Hence, the purpose of scaling is to fix a vector to have points be a certain distance apart creating a clear image (or desired image).
2. The following answers were found through WebGL’s online documentation.

var vPosition = gl.getAttribLocation ( program, “vPosition”);

This line of code is getting the location of an attribute variable named “vPosition” from a shader program “program”. This is needed later in to configure the attribute.

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

This line of code binds the buffer (from gl.Array\_Buffer) to a generic vertex attribute of the current vertex buffer object and specifies the layout.

gl.enableVertexAttribArray ( vPosition );

This turns on the generic vertex attribute array at the specified index into the list of attribute arrays. The makes the vertex attribute array used during rendering.