**Sierpinski Gasket Reflection**

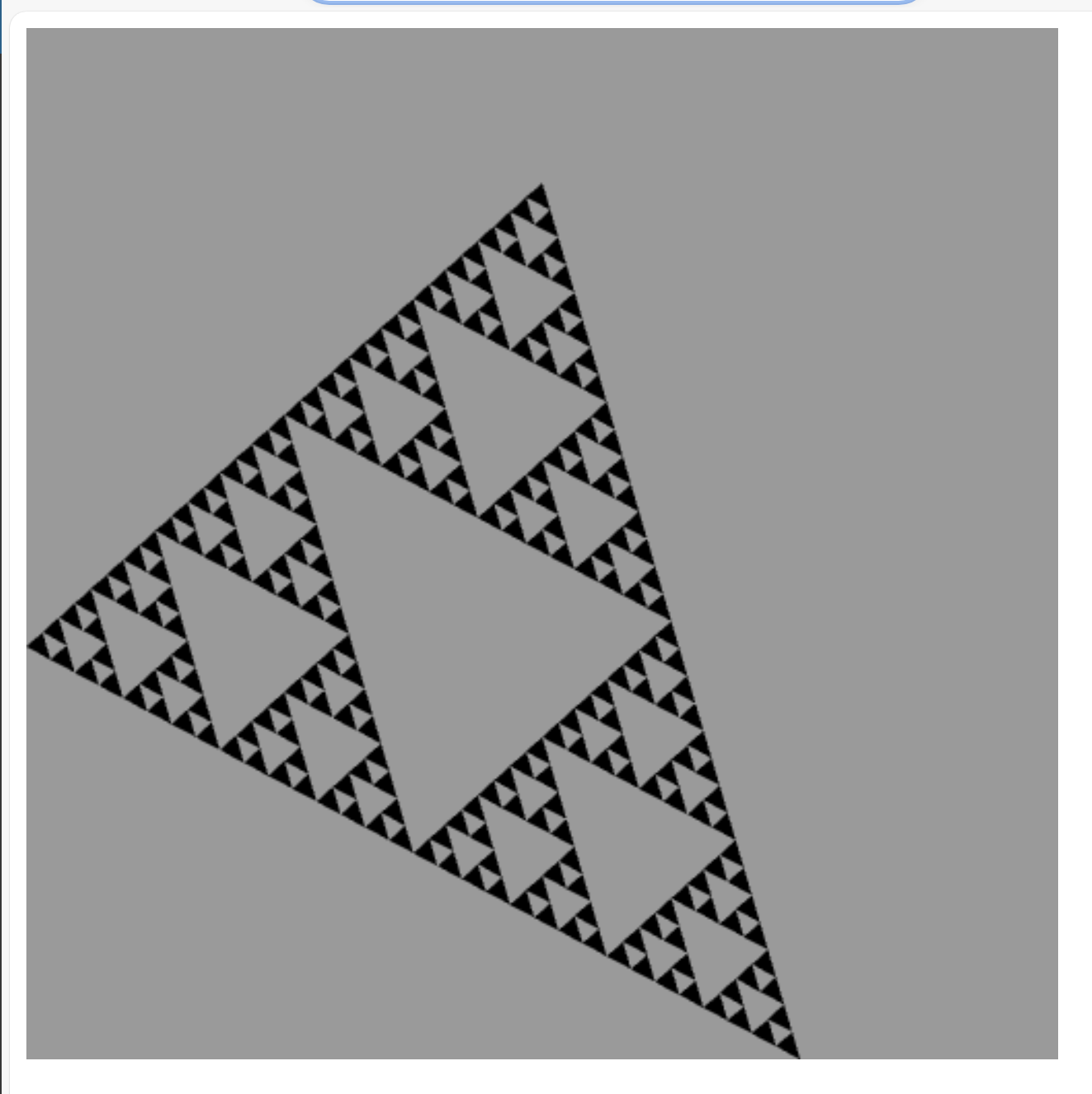
As I witnessed the rendering of the Sierpinski Gasket in WebGL onto the HTML canvas, I observed the creation of a fractal, where no matter which part of the image I looked at, the pattern I saw was repeated throughout. My favorite part about the Sierpinski Gasket is that depth can be any number from a mathematical standpoint, as the limit is infinite. On the other hand, from a physics standpoint, the triangle pattern can only be subdivided by a finite amount. It is the pleasant nature of the Sierpinski Gasket.

The primitive of the Sierpinski Gasket is a triangle, which is essential because of how the lines must be rendered. For example, three points will always be generated, the midpoints of each line segment that creates a triangle. Once those points exist, WebGL uses the triangle primitive to connect each point to the other two, generating a triangle. If the primitive were an alternate geometry, WebGL would want to render line segments to create such geometry with those three vertices, and it wouldn’t render any shape to the screen because there wouldn’t be enough information to construct the geometry. Think of building a square with three points. Where would the fourth go? There are too many options, so nothing is rendered. More simply put, the vertices and primitive must make sense together for one possible outcome.

The attributes that influence the final output are depth of recursion, initial geometry, and rendering techniques. When the recursion parameter is higher, the algorithm goes deeper by that amount, meaning the pattern repeats more and more and smaller and smaller. The initial geometry, the perimeter, enables a canvas for the algorithm to begin. It’s nice that the initial geometry is an equilateral triangle because all midpoints will be the same distance per layer of recursion with the algorithm implementation. Lastly, the rendering technique utilizes shader programs and drawing commands to transform the mathematical specimen into an image.

The pattern that emerges is an omitted triangle within the triangle in the layer above for each iteration of the depth parameter. Because the omitted triangle is formed by the midpoints of the triangle before it, it is rotated 180 degrees, making way for the pattern of three shaded triangles to occupy all other space nearest the vertices.

Aside from altering the rendering techniques which could lead to more abstract results, changing depth and the initial geometry would affect the rendered image. If depth were altered, the algorithm's recursive layers would change. If the initial geometry changes and there is no longer an equilateral triangle, then each line segment would have a different midpoint. Changing the midpoints by using a different initial geometry can provide an alternate perspective of the view. See the rendered image below to see what I’m describing.



Strangely, it feels now that it is a 2D face in a 3D canvas, and the viewer’s perspective is what changes. The viewer would be standing to the left and rotated. In reality, I just changed the starting position, but realizing the connection between the object and the viewer based on the object's initial geometry might be helpful for some low-level renderings relative to the user in the future.