Table of Contents

Fractals

Fractals are mathematical objects defined in fractional dimensions. They were first developed by a French mathematician named Dr. Benoit Mandelbrot. The use of fractals in computer graphics are many, including irregular terrain or surface generation.

We can easily model irregular surfaces using fractal geometry. Fractals have two fundamental characteristics: infinite detail and at every scale and self-similarity at a given scale. Fractal curves or surfaces have a fractal dimension, and the length of a fractal curve in the Euclidean sense is infinite, since there is an infinite amount of detail at every scale of a fractal. An example of this peculiar phenomenon can be experienced while zooming on a coastline from above. At the coarsest level one sees a rough outline of the coast, but at finer levels more and more details appear such as boulders, rocks and stones, and eventually grains of sand, and molecules. Yet at all scales the general aspect of the coastline remains identical.

Koch Curves

A fractal curve can be generated by repeatedly applying a specified transformation function to points within a region of space. The amount of detail present in the final view depends on the number of iterations and the finest resolution available. An example of a self-similar fractal is given by Koch curves. A the start, two line segments of equal length on an axis are angled as the one face of a regular triangle. Then, each one of these two segments is divided in 3 sub-segments of equal length, and the middle segment is split in two and made as a regular triangle once again, as depicted below:

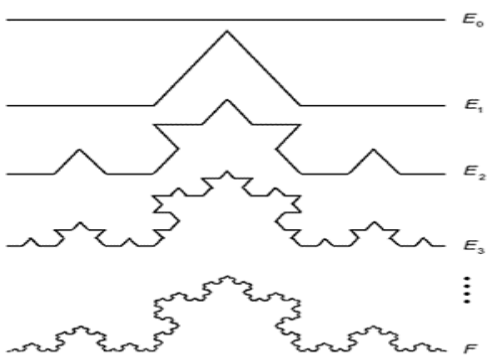

Illustration 1: A typical Koch curve

Under this transformation, it can be easily shown that the total length of the curve increases by four thirds at each iteration, and that the fractal dimension of this curve is given by:

$$d = \frac{\ln n}{\ln\left(\frac{1}{s}\right)}$$

where n is the number of subdivisions at each step, and s is the scaling factor. For the curve depicted above, we have:

$$n=4 \quad s=\frac{1}{3} \quad d=1.2619$$

Theoretically, a fractal curve such as this one has an infinite number of iterations and detail. In practice, the number of iterations must remain finite and thus the curve ends up with a finite length.

Self-Squaring Fractals

Many fractal curves are generated using functions in the complex plane. Consider a complex number $z=x+iy$ where $i=\sqrt{-1}$. We define the norm of this complex number in the complex plane as $\sqrt{x^2+y^2}$ and refer to it as the magnitude of the complex number. If $f(z)=z^2=(x+i\,y)^2=x^2+i2xy+(iy)^2=x^2-y^2+i2xy$, and an iteratively defined complex number is given by applying this formula to the result (such as in $f(f(z))$), then the sequence of complex numbers converges to 1 if the magnitude of z is smaller than one and diverges otherwise. For some functions, the boundary between those points that tend towards infinity and those that converge, is a fractal curve known as the Julia set.

The squaring transformation $z'=f(z)=\lambda z(1-z)=\lambda(z-z^2)$ with λ some constant, is rich in fractal curves. Rearranging terms yields $z^2=z-z'/\lambda=0$, and we compute the inverse of z' as:

$$z=f^{-1}(z')=\frac{1}{2}\left(1\pm\sqrt{1-\frac{4z'}{\lambda}}\right)$$

Given some point on the inside (norm of z is smaller than one) or outside (norm of z is greater than one), convergence towards points on the fractal curve ($z=1$) is obtained.

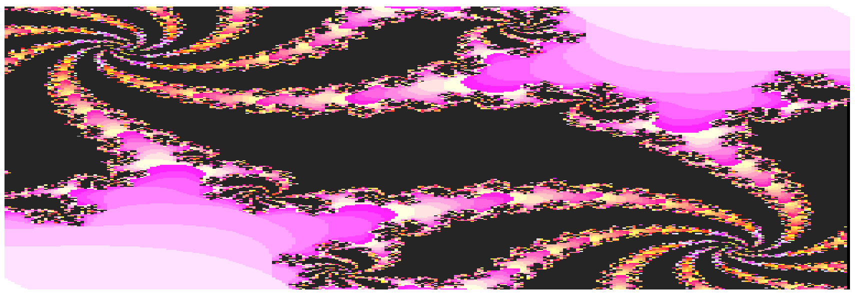

Illustration 2: An example of the Julia set

Mandelbrot Fractal Set

The Mandelbrot set is a set of complex numbers that do not diverge under squaring transformations:

$$\begin{aligned} z_0 &= z \\ z_k &= z_{k-1}^2 + z_0 \text{ for } k = 1, 2, 3 \end{aligned}$$

We select an initial value z_0 , and compute $z_1 = z_0^2 + z_0$, and iteratively continue as in $z_2 = z_1^2 + z_0$ until we can determine if the values are diverging ($\|z_k\| > k$) or a maximum number of iterations has been reached. Then, each initial value for z_0 is chosen as a point on the complex plane, and its color dictated by the number of iterations needed to achieve convergence. These values are then plotted into an image to create a fractal display.

Statistically Self-Similar Fractals

How can we transform a triangle into a mountain, including the sum of the small details? If we

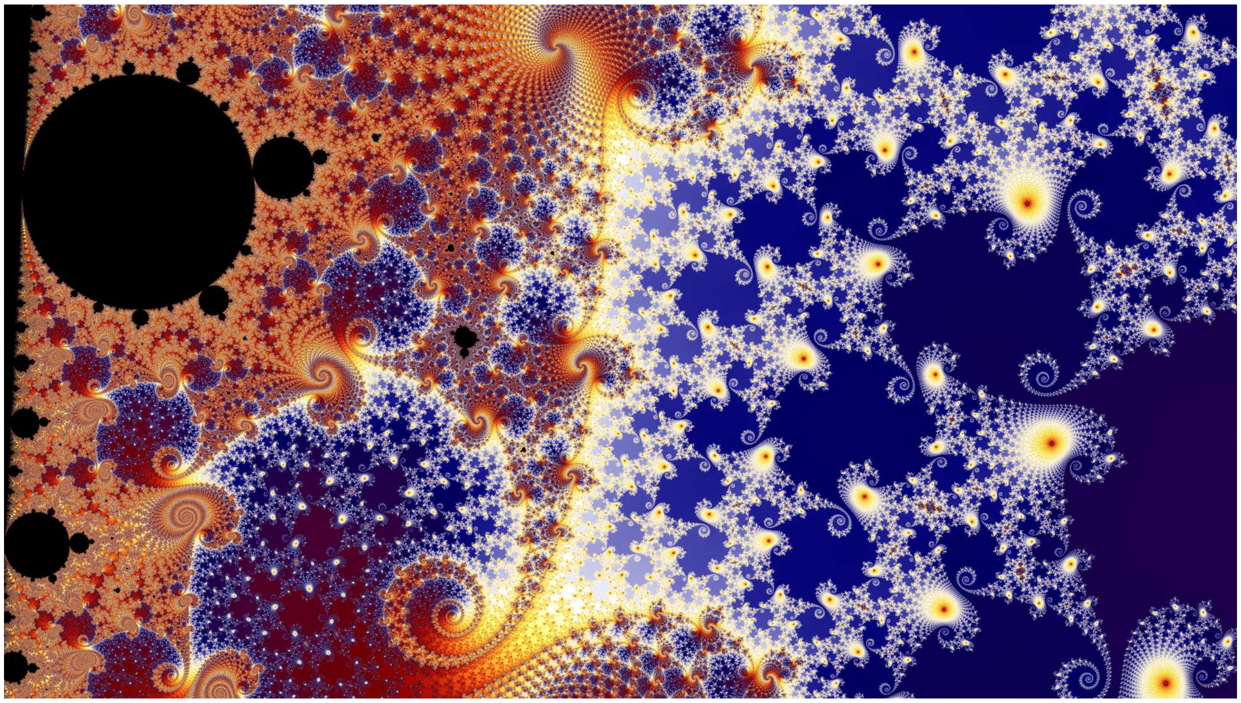

Illustration 3: A zoom into the Mandelbrot set

start with a triangle the shape of the mountain we want, then we can select a random point on each edge of the triangle, and apply a displacement along the edges corresponding to the random numbers. Then joining these displaced points with line segments and repeating the process to the desired level of detail results in a form or random tessellation that can be textured to give the illusion of a mountain range as in the picture below:

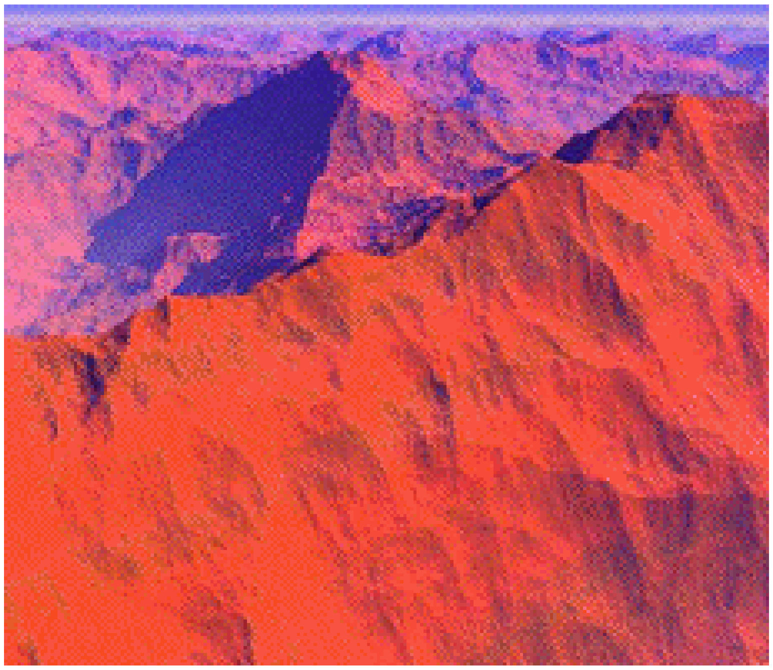

Illustration 4: Mountain range generated with statistically self-similar fractals

This technique can be applied to assembled triangles of all dimensions. It is useful to generate realistic renderings of terrain for which only GPS coordinates are available, after adequate tessellation. Techniques similar to this are used to provide the public with renderings of planet surfaces that are scanned by probes and satellites from NASA. There is quite a number of natural phenomena that are modeled very accurately with fractals. These are new mathematics (historically) and give us a hope in our ability to understand more about the world and how we see it.

Generating Fractal Terrain

We can create realistic terrain (including mountain ranges) using randomness and self-similarity. We define a roughness constant h that determines how the range of generated random numbers is reduced at each iteration. A high value for h will result in smooth terrain.

In order to simulate random terrain we need an array (2D) of height values mapping indices (x, y) to height values z .

Algorithm

- Define a large 2D square array of side size 2^n+1 , ans set the four corner points of this array to identical height values
- The square array midpoint value is obtained by averaging the four corners and adding a random value to it
- Set the square's four sides midpoints by averaging the two closest corner points and

adding a random value, for each midpoint

- Repeat this process until the 2D array is filled. At each iteration, the range of acceptable random numbers should be multiplied by = 2^{-h} , where h is the roughness constant

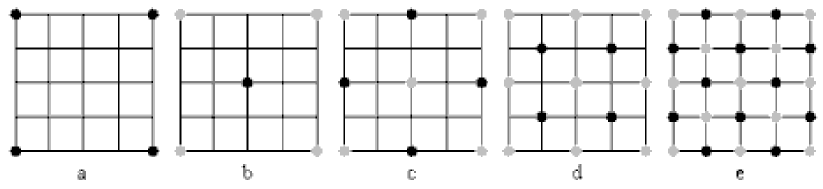

Illustration 5: The steps involved in generating terrain

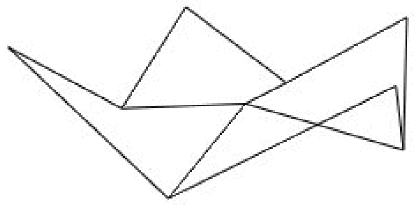

Illustration 6: Rough terrain

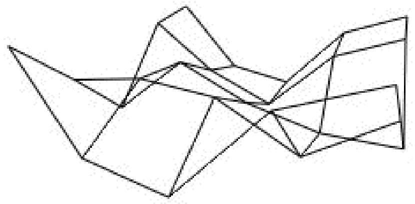

Illustration 7: Rough terrain after a few iterations