Algorithms for Generating Fractal Landscapes

Keith Stanger

December 7, 2006

Abstract

This paper aims to provide an introduction to creating fractal landscapes. I will present some methods for generating landscapes. In order to help the reader better understand the methods presented I will discuss the properties of real landscapes that make them fractals.

I will describe three algorithms for generating fractal landscapes. I will compare them based on the characteristics of the landscapes they produce. I will also discuss some useful modifications and extensions to the algorithms discussed.

An Introduction to Fractal Landscapes

This paper discusses algorithms that are used to create artificial landscapes. The goal in generating fractal landscapes is to make them look as realistic as possible. A generated landscape can be made quite realistic by adding colour, proper lighting, water, plants, atmospheric effects and other such things. This paper focuses on the process of creating the topological form of the landscape.

The output of the algorithms discussed is a set of altitudes assigned to a two-dimensional grid. Inputs into the algorithms are parameters that define certain desired characteristics of the generated landscape, mainly its roughness.

The algorithms must use randomness otherwise we would not have interesting or unique results. Simply assigning a random altitude to each point in our grid won't do because

we expect that two points that are close to each other are likely to have similar altitudes. So we need an algorithm that will take into account the proximity of two points in assigning their altitude. To understand the fractal landscape algorithms, we must first understand why a real landscape is a fractal.

Firstly, landscapes are self-similar. They are not self-similar in the sense that looking at a small portion of the landscape, you will see the same shape that you see looking at the entire landscape. Looking at a landscape at different scales, though, you will notice that it exhibits the same basic characteristics at every scale. Consider a mountainous terrain, each mountain likely has several peaks. Think of these peaks as being the result of a smaller mountainous terrain imposed on top of one mountain in the larger terrain. Further, each of these peaks contains more peaks but on a smaller scale. It is in this sense that landscapes are self-similar.

The characteristic that allows us to define landscapes as fractals is the fact that they have fractional dimensions. In 1967, Benoit Mandelbrot coined the popular question: "How long is the coast of Britain?" The answer is not as simple as it may appear. A measurement of the length of the coastline would depend entirely on the scale of the measurement device used. If you were to measure the coastline one kilometer at a time, you may think it would yield a reasonably accurate measurement. But you would have been ignoring all the rough detail within each kilometer. You would find that taking a meter stick and measuring the coastline one meter at a time would yield a far greater result. In fact, every measurement using a smaller scale would yield a greater result, without limit. The answer to the question is that the coastline of Britain does not have a length because it is not one-dimensional. It is a fractal.

The algorithms that will be examined make use of the fractal properties of landscapes in converting random numbers into landscapes. Three algorithms are examined. The Triangle Division and Diamond-Square algorithms use iterative processes. They take advantage of the self-similarity property by applying the same rule to the landscape at a smaller scale with each iteration. The random numbers are used to perturb the altitude,

creating random landscape-like features. The Fourier Transform algorithm uses discrete Fourier transforms to represent a landscape in the frequency domain. By moulding the frequencies, this algorithm can convert a set of random numbers into a landscape. Also discussed are a couple ways to achieve certain characteristics in the landscapes using the previous algorithms.

Triangle Division Algorithm

The Triangle Division algorithm takes a triangle-shaped terrain, and at each iteration divides each triangle into four smaller triangles, applying a random perturbation to each new vertex created.

1. Start with a triangle. Pick random altitudes for the endpoints.
2. For each edge on the triangle, take the midpoint and add a random perturbation. The perturbation is equally distributed between $-k^r$ and k^r, where k is the length of the divided edge and r is a roughness parameter.
3. Divide the triangle into 4 smaller triangles by connecting the midpoints, then repeat the process on each of the smaller triangles.

The process is iterated until the desired amount of detail is reached. Notice that at each iteration the random perturbations are dependent on the size of the triangles at that iteration, so the amount of perturbation becomes smaller with every iteration. This property makes the generated object landscape-like. The fact that the same process is applied to the landscape but at a smaller scale each time makes it self-similar or fractal.

This algorithm has a roughness parameter (r) which gives some control over the resulting landscape. Notice that with higher r, the perturbation decreases more quickly after each iteration than with lower r. So with relatively high r, the perturbations become relatively small at high levels of detail. The result is a smooth landscape. Conversely, a low value of r will result in a rough landscape. Therefore, r is called the roughness parameter.

Below are two landscapes using the triangle division algorithm. The algorithm can be applied to a square grid by dividing the square along a diagonal, creating two triangles.

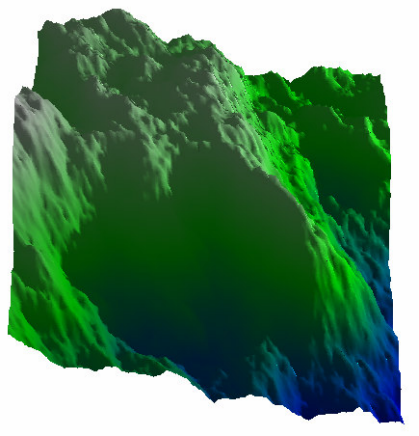


Figure 1 - Triangle Division r = 1.0

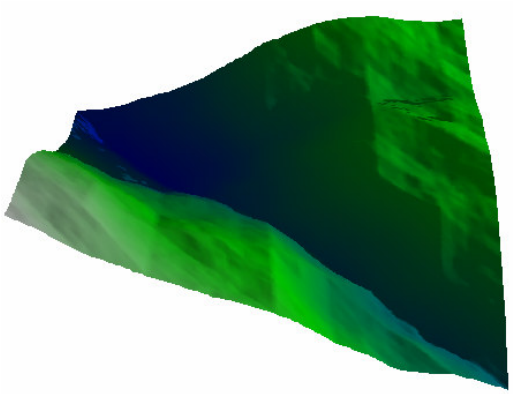


Figure 2 - Triangle Division r = 1.5

Notice that using a higher value for r produces a much smother landscape.

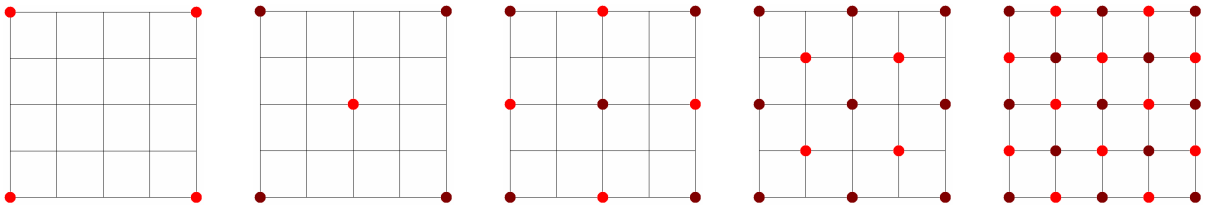
An apparent flaw in this algorithm is the forming of ridges along the edges of the triangles. This is more obvious in the smoother landscape, but notice in Figure 1 there is a ridge that forms along the main diagonal. This is a result of the geometry of the algorithm. Ideally, a fractal landscape would have no artificial forms such as this. The reason these ridges appear is because the altitude of each point is determined from only two other points, so it is dependent on points in two opposite directions, but not in any other direction.

Diamond-Square Algorithm

The Diamond-Square algorithm improves upon the flaw pointed out in the Triangle Division algorithm, by having each point depend on points in four directions rather than two. The algorithm is as follows:

1. Assign random altitudes to the four corners of a grid.
2. Diamond step: Average the four corners and add a random perturbation evenly distributed between $-r^i$ and r^i. Assign this to the midpoint of the four corners.
3. Square step: For each diamond produced, average the four corners and add a random perturbation with the same distribution.
4. Repeat steps 2 and 3 for the desired number of iterations

The value r is a roughness parameter, and i is the current iteration. The following diagrams illustrate the points computed in the diamond and square steps of the first two iterations:



Step 1 **Diamond - 1** **Square - 1** **Diamond - 2** **Square - 2**

Notice that this algorithm uses a different computation to determine perturbations as was used in the Triangle Division algorithm. Here the random numbers are scaled by a factor of r^i, compared to k^r in the other algorithm, where k is the side-length of a triangle or square. The two methods are, in fact, equivalent. Notice that the side-length of a triangle or square in either algorithm is halved at each iteration, so $k = (½)^i$ and $k^r = (½)^{ri}$, which is equal to using r'^i where $r' = (½)^r$. The main difference is that in this algorithm, increasing r increases roughness.

Here are two fractal landscapes generated by using the Diamond-Square algorithm:

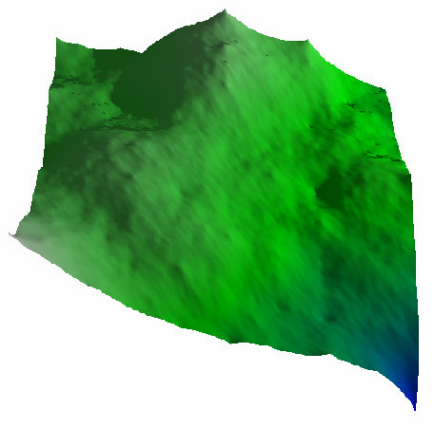


Figure 3 - Diamond-Square r = 0.4

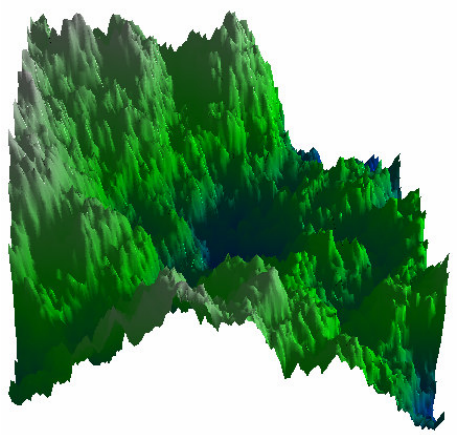


Figure 4 - Diamond-Square r = 0.6

There aren't any ridges such as those that appeared using the Triangle Division algorithm, but you can see on the smother landscape there are small peaks or troths along the major gridlines. This is better than the previous algorithm, but we still do not have a truly random fractal landscape.

A possible improvement upon this algorithm would be to use a cubic spline interpolation for each point, or some other non-linear interpolation method rather than the linear interpolation that is used here. This could help smooth out the peaks and troths that are formed by taking into account very many other points rather than just the four closest points in interpolation. An algorithm using cubic splines is not explored in this paper.

Fourier Transform Algorithm

A Fourier transform is based on the idea that a function can be expressed as a sum of sine or cosine waves at different frequencies. The Fourier transform expresses a function in this form; it converts the function to its frequency domain.

A Discrete Fourier Transform (DFT) is a Fourier Transform applied to a set of discrete values. The result is a set of discrete complex numbers of the same size as the original set. The resulting values can be thought of as amplitudes of various frequencies in the

original set. The process can be reversed without any change to the original set. A DFT can be performed in $O(n \cdot \log(n))$ using a Fast Fourier Transform (FFT) algorithm making it a reasonably efficient process. A DFT can also be applied to multiple dimensions.

The idea of a Fourier Transform ties into fractal landscapes by thinking of a landscape as a sum of waves at different frequencies. Observe that the amplitude of these waves has a decreasing trend as the frequency increases. This is essentially the same as the observation used in the previous algorithms that the size of perturbations should decrease as the scale at which they are being applied decreases.

The Fourier Transform Algorithm is quite different from the previous algorithms discussed, mainly because it is not an iterative process. At a high level it is quite simple:

1. Create a 2-dimentional grid of random values.
2. Apply a 2-dimentional FFT.
3. Scale each of the values in the transform by $1/f^r$, where f is the frequency represented by that value, and r is the roughness parameter.
4. Apply the inverse FFT. The result is a fractal landscape.

The resulting fractal landscape does not contain any artificial ridges or peaks as the previous algorithms did. In addition, the landscape has the interesting (and sometimes desirable) property that it can be tiled. One side of the landscape flows perfectly into the opposite side.

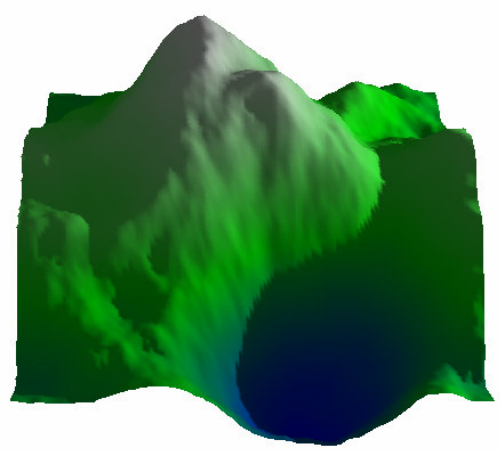


Figure 5 – Fourier r = 2.5

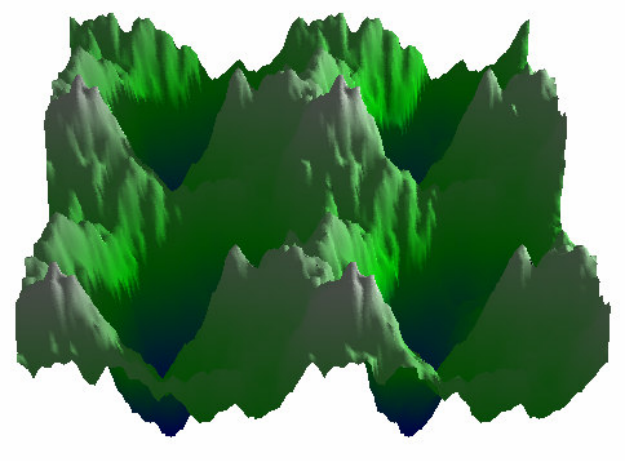


Figure 6 – Fourier r = 2.0 - tiled

Multiplication Technique

At this point it is important to think about how realistic the landscapes generated by the previous algorithms really are. In the landscapes they generate, both the valleys and peaks contain the same characteristics at the small scale. But is this truly what we want from our fractal landscapes?

In the real world the valleys are, in fact, much smother than the peaks. This is the result of erosion. The erosion process takes material from higher altitudes and settles it in the valleys, making them smother. There would also be rivers, which is a form that does not occur using any of the algorithms discussed.

A common solution to this problem is to simulate the erosion process. This is a topic beyond the scope of this paper. You could consider modifying the Triangle Division or Diamond-Square algorithms to take the interpolated altitude into account when determining the random perturbation. This would help to solve the problem of smother valleys vs. rougher peaks.

A quick and easy solution is to multiply two landscapes together. Generate two landscapes with altitudes between 0 and 1, then multiply the corresponding altitudes

together to create a new landscape. All altitudes will be scaled down, but the lower ones more than the higher ones, making the terrain features at high altitudes more exaggerated relative to the features at low altitudes. The result is smother valleys and rougher peaks, as desired.

The following two landscapes were created by generating two fractal landscapes, scaling them so that the altitudes lie between 0 and 1, then multiplying them together. The first uses the Diamond-Square algorithm, and the second uses the Fourier Transform algorithm.

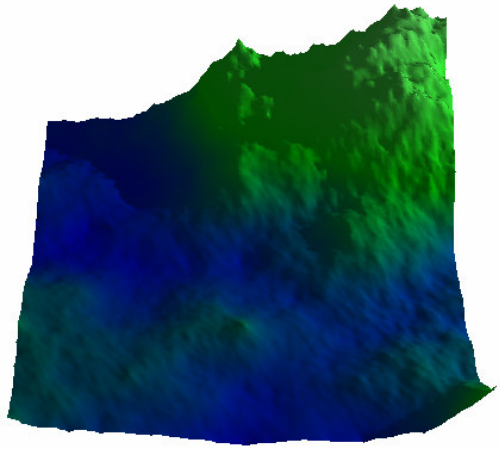


Figure 7 – DiamondSquare^2 r = 0.4, 0.5

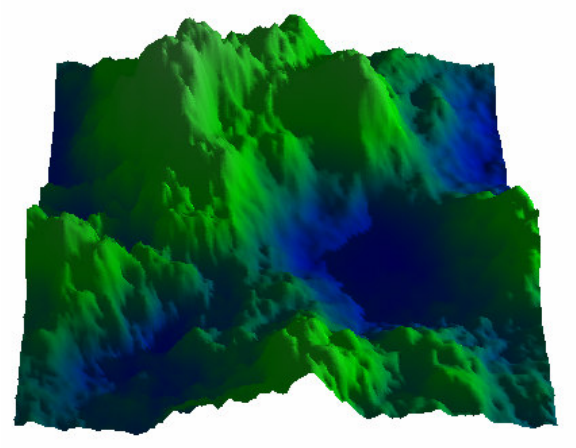


Figure 8 - Fourier^2 r = 1.7, 2.5

Modifying the Fourier Transform Algorithm

The underlying idea behind the Fourier Transform algorithm is that by taking a set of random numbers, we can scale down the higher frequencies to create a landscape. The curious mathematician would wonder what other sorts of rules can be applied while in the frequency domain to alter the characteristics of the landscape.

An interesting and useful result is achieved by scaling down the frequencies by a larger amount in one direction than the other. The resulting features are skewed in one direction, similar to the way mountain ranges typically form in a line. Figure 9 was

generated by scaling the frequencies down in one direction by twice the scaling factor. Figure 10 was generated by combining this method with the multiplication technique.

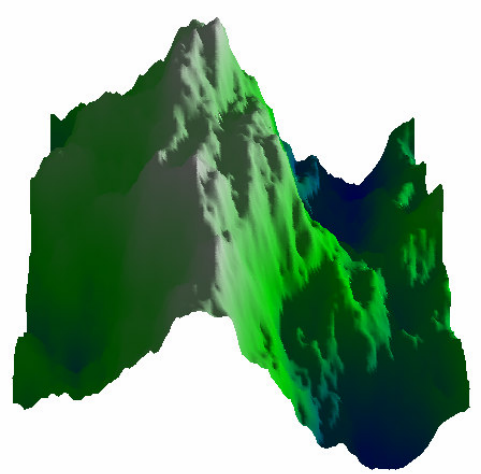


Figure 9 - Skewed Fourier

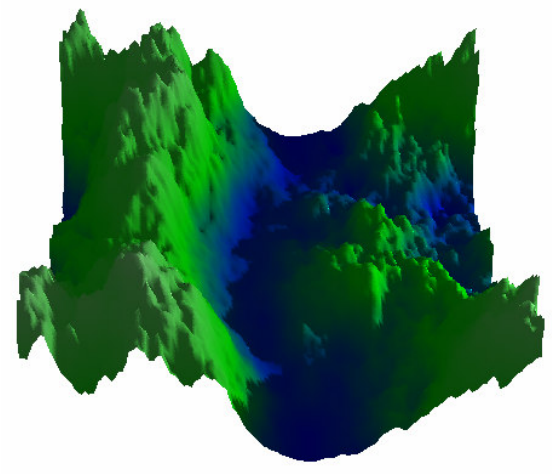


Figure 10 - Skewed Fourier^2

Conclusions

We have examined three algorithms for generating fractal landscapes. The Triangle Division and Diamond-Square algorithms are very similar. Both algorithms produce unnatural forms in the generated landscape, the Diamond-Square being an improvement upon the Triangle Division algorithm because the computation of each point is dependent upon four, rather than two, nearby points. The Fourier Transform algorithm takes advantage of the observation that frequencies can be used to describe a natural landscape. It has the property that a generated landscape can be tiled. It can also be modified to create mountain range-like landscapes.

The landscapes generated by the algorithms described fail to mimic certain characteristics of real landscapes. An easy way to create a more realistic fractal landscape is to multiply two generated landscapes together, creating smoother valleys and rougher peaks. More advanced methods exist to convert a generated landscape into a realistic landscape such

as simulating the erosion process. The algorithms discussed here provide an introduction to generating realistic landscapes.

Acknowledgements

Landscape Images used in this paper were rendered using source code downloaded from JavaWorld.com (see Works Cited). Code for generating the landscapes, with the exception of the basic Diamond-Square, is originally written.

Works Cited

Bourke, Paul. "Frequency Synthesis of Landscapes (and Clouds)." March 1997. <http://local.wasp.uwa.edu.au/~pbourke/modelling/planets/>

Brown, Adam. Fractal Landscapes. <http://www.fractal-landscapes.co.uk/maths.html>

"Cooley-Tukey FFT Algorithm." Wikipedia. 2 November 2006. <http://en.wikipedia.org/wiki/Cooley-Tukey_FFT_algorithm>

"Fast Fourier Transform." Wikipedia. 22 November 2006. <http://en.wikipedia.org/wiki/Fast_Fourier_transform>

"How Long Is the Coast of Britain? Statistical Self-Similarity and Fractional Dimension." Wikipedia. 13 November 2006. <http://en.wikipedia.org/wiki/How_Long_Is_the_Coast_of_Britain%3F_Statistical_Self-Similarity_and_Fractional_Dimension>

Hughes, Merlin. "3D Graphic Java: Render fractal landscapes." JavaWorld.com. 8 January 1998. <http://www.javaworld.com/javaworld/jw-08-1998/jw-08-step.html>

Java Source Code. JavaWorld.com. <http://www.javaworld.com/jw-08-1998/step/jw-08-step.tar.gz>

Turner, Martin J. "Modeling nature with fractals," Plus Magazine. September 1998. <http://pass.maths.org/issue6/turner2/index.html>

"VDS Fractal Landscape Generator." <http://wwwcs.uni-paderborn.de/SFB376/projects/a2/zBufferMerging/>