2 Procedural Fractal Terrains

F. Kenton Musgrave
FractalWorlds.com
musgrave@fractalworlds.com
www.fractalworlds.com/musgrave

Abstract

These course notes describe the fundamentals of fractal terrain models, with an emphasis on procedural models. We describe in detail fractional Brownian motion (fBm), the archetypal random fractal upon which all synthetic terrain models are based. FBm is by design statistically homogeneous; Nature is not. We discuss non-homogeneous terrain models based on fBm that begin to evoke the heterogeneity seen in Nature. Domain distortion in procedural terrain models is invoked to emulate geomorphic effects such as soft sediment deformation and metamorphism. A "slumping" filter is described to emulate diffusive erosion converging to an angle of repose. Finally, we note the need for physical simulation of fluvial and glacial erosion to create context-sensitive fractal features such as river drainage networks, and the current impracticability of such simulations as compared to fBm-based models.

2.1 Introduction

Since Richard Voss' classic images of fractal landscapes appeared in Benoit Mandelbrot's book "The Fractal Geometry of Nature" [2] and elsewhere the computer graphics community has been fascinated by fractal landscapes. The quality of the "fractal forgeries of Nature," as Mandelbrot calls them, has steadily improved over the roughly two decades of their use. I had the great privilege of working closely with Mandelbrot at Yale from 1987 to 1993. My own doctoral dissertation [3], written during that time, contributed certain advances to the field, mostly in the area of non-homogeneous fractal terrain models.

There are two textbooks that cover the topic of fractal terrain models: "The Science of Fractal Images" [5] and "Texturing and Modeling: A Procedural Approach." [1] The first covers exhaustively the mathematics behind these models. The second covers the practice of designing and using such models. As an author of portions of both books, *I must refer you to these texts for detailed treatment of this subject matter*. Not only have I and others done our best to do a clear and exhaustive presentation there, I am also constrained by copyright law not to reproduce that material here.

Given those constraints, what I will do here is try to convey the *intuition* of fractal terrain models. While the fundamentals can seem slippery at first, it turns out that they are deceptively simple. And simple is good.

If you are more interested in using fractal terrains than in implementing them, you're in luck: At the time of this writing I have just completed, about three weeks ago, a comprehensive implementation of almost every terrain model I have ever conceived of and

shipped them out in MetaCreations' Bryce 4. So if you want to play with terrain models, or see what a wide variety of them look like, get a copy of Bryce 4. Bryce is aimed at hobbyists more than professionals, so it is fun to use, too. These and other, more complex, colorful fractal textures based on the methods described here are also available as the "Noizes" in MetaCreations' KPT5 product.

2.2 What is a *fractal*?

What *is* a fractal? Let me define a fractal as: *a geometrically complex object, the complexity of which arises through the repetition of some shape over a range of scales*. Note the simplicity and breadth of this definition. This simple, heuristic definition will be sufficient for our purposes in describing terrains. There are more mathematical definitions, but they are not useful in this context, so we'll keep to this heuristic definition.

Fractals are common in Nature: mountains, clouds, trees, turbulence, circulatory systems in plants and animals are fractal, as are a wide variety of more subtle phenomena such as noise in transistors and fluctuations in river fluxes. Fractal geometry is very powerful—though not sufficient—for describing the complex forms found in Nature. The geometry we learned in school, the Euclidean geometry of lines, planes, spheres and cones, describes very well most man-made objects but it fails utterly when confronted with most natural phenomena, e.g., mountains, clouds, and lightning. Yet those same phenomena can be described quite succinctly by fractal geometry. Conversely, fractal geometry has very little use in describing man-made objects—human artisans rarely have the patience to build the kind of complexity that characterizes fractals.

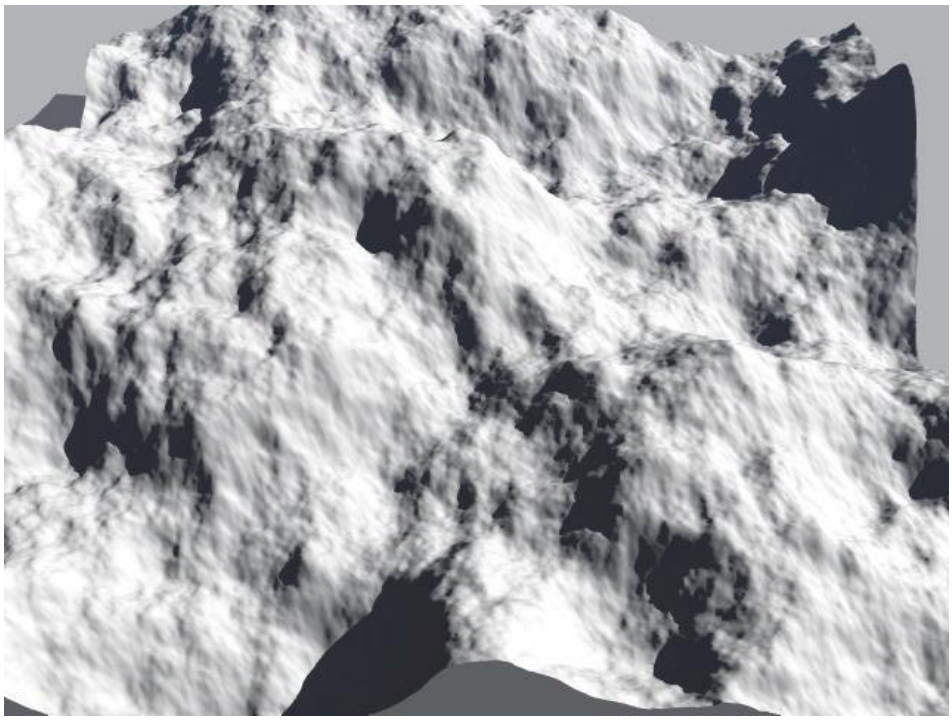
Though our heuristic definition of fractals is sufficient, let me explain some of the mathematical details that we will be using. Fractals have the peculiar property of *fractal dimension*, which can have non-integer values such as 2.3. We are all familiar with the Euclidean integer-valued dimensions: a dimension of zero corresponds to a point, one to a line, two to a plane, and three to space. The real-valued fractal dimensions, such as 2.3, provide a continuous "slider" for the visual complexity of a fractal construction. The whole component of the fractal dimension—the "2" in 2.3—indicates the underlying Euclidean dimension of the fractal, in this case a plane. The "fractional" part—the ".3" in 2.3—is called the *fractal increment*. As this part varies from .0 to .999..., the fractal literally goes from (locally) occupying only its underlying Euclidean dimension, for instance a plane, to filling some part of the next higher dimension, such as space. It does this by becoming ever more convoluted, as the value of the fractal increment increases. For the discussion here, we'll stick to the intuition of fractal dimension as a slider that makes our terrains rougher or smoother.

The source of the convoluted complexity that leads to this intermediate dimensionality is, again, simply the *repetition of some underlying shape*, over a variety of different scales. I refer to this underlying shape as the *basis function*. For the mathematically well-defined fBm, the basis is a sine wave. The basis function can be literally anything, but for the

fractals described here it is either a variation of Ken Perlin's "noise" function[1] or Steve Worley's Voronoi functions. [7] For now, think of the basis function as providing a kind of cottage cheese with lumps all of a particular size. We build a fractal from it, simply by scaling down the lateral size of the lumps, and their height as well, and adding them back in to the original lumps. We do this several times, and presto! we have a fractal.

To be a little more technical: We refer to the lateral size of the lumps as the *frequency* of the function (more specifically, the *spatial frequency*). The height of the lumps we refer to as the *amplitude*. The amount by which we change the lateral size, or frequency, in each step of our iterative addition process, is referred to as the *lacunarity* of the fractal. (Lacunarity is a Latin word meaning "gap". The gap, in this case, is between successive frequencies in the fractal construction.) In practice, lacunarity is a non-issue, as we almost always leave it set at a value very close to 2.0. In music doubling the frequency, which is what a lacunarity value of exactly 2.0 implies, raises a given pitch by exactly one octave; hence we generally speak of the number of *octaves* in our fractals—this corresponds to the number of times we scaled down and added back in, smaller lumps to bigger lumps. The number of octaves in the fractal determines its *bandwidth*.

There is a well-defined relationship between the amount by which we scale size and the amount by which we scale the amplitude. This relationship is what determines the fractal dimension of our result. Again, I will decline to get any more technical, and refer the interested reader to "The Science of Fractal Images" [5] for details.



An ordinary fractional Brownian motion (fBm) terrain patch of fractal dimension ~2.1.

[1] The version of the noise function I prefer to use has zero crossings (i.e., its value is zero) at lattice points, and a range of [-1,1]. These properties have specific importance in certain constructions. My preferred Perlin noise is what Peachey calls *gradient noise*. [1]

2.3 Procedural fBm

Let's see exactly how to build the archetypal fractal procedural texture, fBm:

```
/*
 * Procedural fBm evaluated at "point"; returns value stored in "value".
 *
 * Parameters:
 *    "H" is the fractal increment
 *    "lacunarity" is the gap between successive frequencies
 *    "octaves" is the number of frequencies in the fBm
 *    "Basis()" is usually Perlin noise
 */
double
fBm( Vector point; double H, lacunarity, octaves )
{
      double            value, frequency, remainder, Basis();
      int               i;
      static            first = TRUE;
      static double     exponent_array[MAX_OCTAVES];

      /* precompute and store spectral weights */
      if ( first ) {
            frequency = 1.0;
            for (i=0; i<MAX_OCTAVES; i++) {
                  /* compute weight for each frequency */
                  exponent_array[i] = pow( frequency, -H );
                  frequency *= lacunarity;
            }
            first = FALSE;
      }

      value = 0.0;

      /* inner loop of spectral construction */
      for (i=0; i<octaves; i++) {
            value += Basis( point ) * exponent_array[i];
            point.x *= lacunarity;
            point.y *= lacunarity;
            point.z *= lacunarity;
      } /* for */

      remainder = octaves - (int)octaves;
      if ( remainder )  /* add in "octaves" remainder */
            /* "i" and spatial freq. are preset in loop above */
            value += remainder * Basis( point ) * exponent_array[i];

      return( value );

} /* fBm() */
```

Note that most of the above code has to do with initialization, computing and storing the exponent array for efficiency, and the picayune point of dealing with the remainder of `octaves`. The fractal itself is constructed in the six-line inner loop. In practice, I code this in C++. This has the following advantages: 1) the initialization can be performed in the constructor, 2) there can be a separate exponent array for each instance of the function, allowing each instance to have a different fractal dimension, and 3) overloaded vector operators make the code even more terse and readable.

This is a generalization of Perlin's original "chaos" function. [6] I have provided new parameters to control lacunarity (which in most cases can simply be fixed at 2.0, as Perlin originally did), the fractal increment H, and the number of octaves in the construction. Also, I've substituted `Basis()` where Perlin called his noise function. For a completely general fractal function one can make `Basis()` a pointer to a function, and pass it as a parameter. I currently use a table of basis functions, and pass a specific one to each instance of a fractal function.

We accommodate non-integer values for the octaves to accommodate rendering with level of detail as used, for instance, in QAEB tracing. [1] Formally, a properly band-limited fBm seen from a distance of 1.0 would have:

$$\text{octaves} = \log_2(\text{screen resolution}) - 2$$

or a value of about 6 to 10. The -2 term in this expression has to do with the fact that the Nyquist limit is 1/2 of the screen resolution, and that the highest frequency in the Perlin noise function is ~1/2 of the lattice spacing. Then we have $1/2 * 1/2 = 1/4$, and $\log_2(1/4) = -2$. You can use a smaller number of octaves to speed the rendering time of test images.

The parameter H is equal to 1.0 - [the fractal increment]. It corresponds to the H described by Voss in *The Science of Fractal Images*. When H=1, the fBm is relatively smooth; as H goes to 0, the function becomes more like white noise. Figure 1 shows traces of the function for various values of H. The underlying Euclidean dimension of the function is given by the dimensionality of the vector-valued argument `point`. The code example has a base dimension of 3; if you write custom noise functions, you can half the execution time for each dimension you drop. This can be useful when QAEB tracing terrains, for instance. 4 dimensional basis functions can be useful for animating volumetric phenomena such as flames.

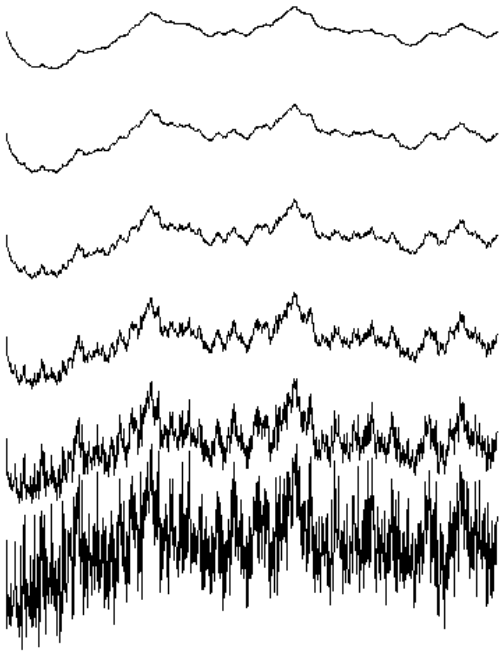


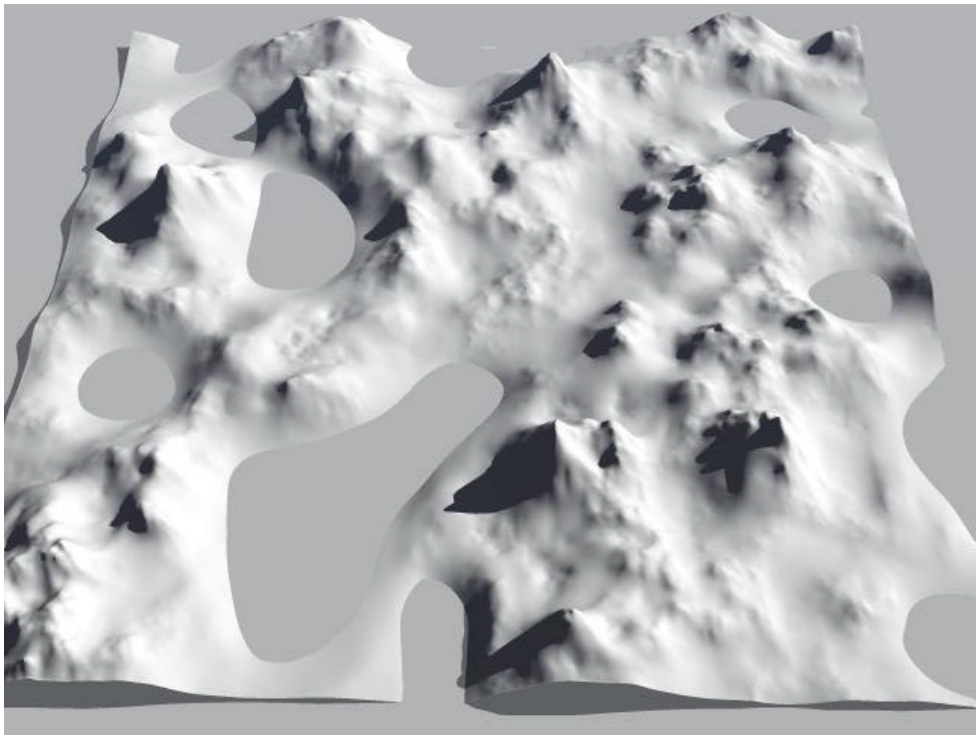
Figure 1. Traces of fBm for H varying from 1.0 to 0.0 in increments of 0.2.

2.4 Heterogeneous fBm

The fBm described above is, as well as we can make it, statistically *homogeneous* and *isotropic*. Homogeneous means "the same everywhere" and isotropic means "the same in all directions". Fractal phenomena in Nature are rarely so simple and well-behaved. For instance, synthetic fractal mountains constructed with a single, uniform fractal dimension everywhere have the same roughness everywhere. Real mountains are never like that: They typically rise out of flatter terrain and have rolling foothills at their feet, not to mention a host of other complications. For greater realism we desire some more interesting, heterogeneous fractal functions.

An early conjecture of mine was that valley floors should be smooth, as compared to the mountains. It occurred to me that this could be accomplished by scaling higher frequencies in the summation by the value of the previous frequency. Note that this assumes that the basis function has a range of [0..1] rather than [-1..1]. Since a valley corresponds to a local minimum in the terrain, I conjectured that a low value of the basis function at a given frequency would imply a local minimum (valley) in the terrain, and that therefore all higher frequencies at that place should be damped to keep things locally smooth. I quickly realized that this reasoning is flawed, as a valley is defined by the local values of the first and second derivative of the terrain. The first derivative must be small, indicating that the surface is locally horizontal, and the second derivative must indicate that we are at a local minimum, rather than a maximum (corresponding to a peak in the terrain). Fortunately, I didn't figure that out until after I had implemented such a function. While it does not do what I had intended, it is simple and yields very nice heterogeneous

terrains nevertheless. For reasons that Mandelbrot does not entirely approve of, I call this function a "hybrid multifractal."

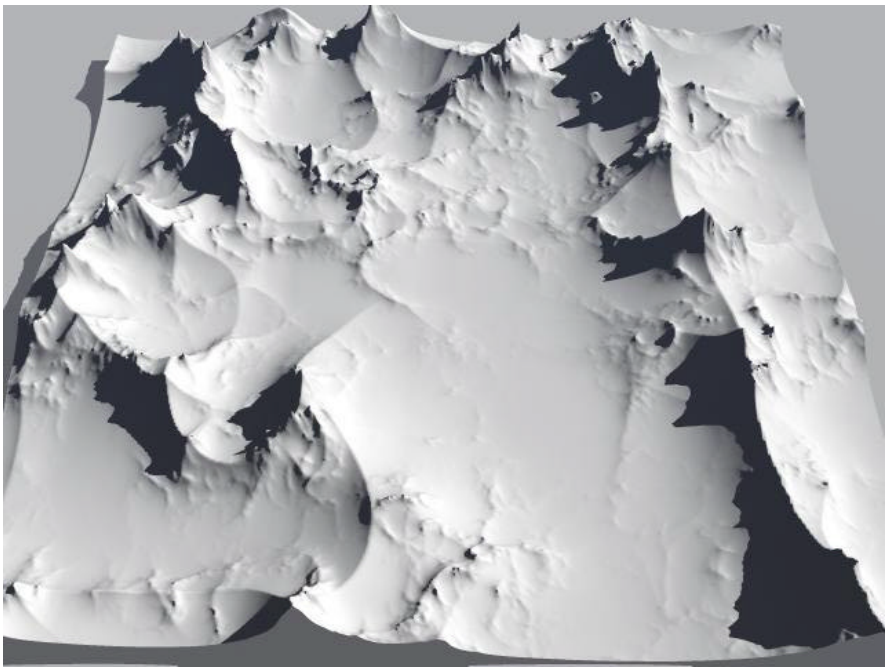


A hybrid multifractal terrain patch made with a Perlin noise basis: the “alpine hills” Bryce 4 terrain model. There is a flat ground plane added at altitude zero to mask details below.

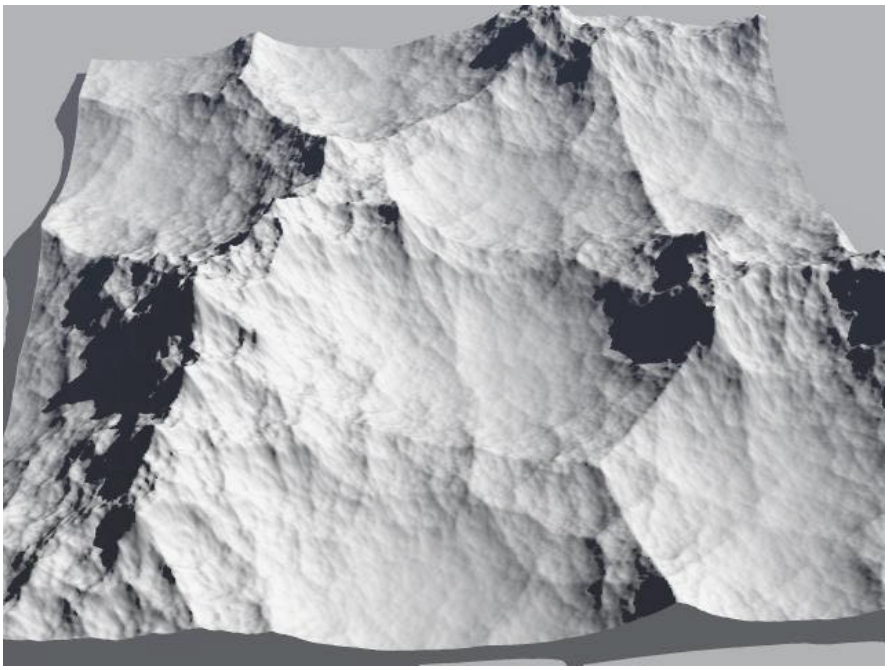
```
/* Hybrid additive/multiplicative multifractal terrain model.
 *
 * Some good parameter values to start with:
 *    H:          0.25
 *    offset:     0.7
 */
double HybridMultifractal( Vector point, double H, double lacunarity,
                           double octaves, double offset )
{
      double            frequency, result, signal, weight, remainder;
      double            Noise3();
      int               i;
      static            first = TRUE;
      static double     *exponent_array;

      /* precompute and store spectral weights */
      if ( first ) {
            /* seize required memory for exponent_array */
            exponent_array =
                        (double *)malloc( octaves * sizeof(double) );
            frequency = 1.0;
            for (i=0; i<octaves; i++) {
                  /* compute weight for each frequency */
                  exponent_array[i] = pow( frequency, -H );
                  frequency *= lacunarity;
            }
            first = FALSE;
      }

      /* get first octave of function */
```

```
        result = ( Noise3( point ) + offset ) * exponent_array[0];
        weight = result;
        /* increase frequency */
        point.x *= lacunarity;
        point.y *= lacunarity;
        point.z *= lacunarity;

        /* spectral construction inner loop, where the fractal is built */
        for (i=1; i<octaves; i++) {
              /* prevent divergence */
              if ( weight > 1.0 )  weight = 1.0;

              /* get next higher frequency */
              signal = ( Noise3( point ) + offset ) * exponent_array[i];
              /* add it in, weighted by previous freq's local value */
              result += weight * signal;

              /* update the (monotonically decreasing) weighting value */
              /* (this is why H must specify a high fractal dimension) */
              weight *= signal;

              /* increase frequency */
              point.x *= lacunarity;
              point.y *= lacunarity;
              point.z *= lacunarity;
        } /* for */

        /* take care of remainder in “octaves” */
        remainder = octaves - (int)octaves;
        if ( remainder )
              /* “i” and spatial freq. are preset in loop above */
              result += remainder * Noise3( point ) * exponent_array[i];

        return( result );
} /* HybridMultifractal() */
```

Note the offset applied to the Perlin noise function, to move its range from [-1,1] to something closer to [0,2]. (If your noise function has a different range, you’ll need to adjust this.) I've successfully used similar heterogeneous fractal constructions with other basis functions such as one minus the absolute value of Perlin noise, and Worley's Voronoi bases. The terrain models "ridges," "Mordor," and "shattered hills" in Bryce 4 are examples of such terrain models.



The “ridges” terrain model from Bryce 4: a hybrid multifractal made from one minus the absolute value of Perlin noise.



The “Mordor” terrain model from Bryce 4: a hybrid multifractal made from Worley’s Voronoi distance-squared basis.

The “shattered hills” terrain model from Bryce 4: a hybrid multifractal made from Worley’s Voronoi distance basis. Note the smooth horizontal band in the lower left around where the value of the first octave was zero. Higher frequencies were heavily damped there, leading to a lack of detail in the final result. Note also that the creases in the basis function from the first octave are clearly visible there. This is a clear example of how the character of the basis function can show through in the final fractal. Compare this and the previous two illustrations to see more on the affects of the basis function on the resulting fractal—the fractal construction algorithm is the same for all of them, but the basis functions are different. This illustrates the importance of the basis function in determining the “look” of a fractal.

2.5 Warped Terrains: Domain Distortion

Sometimes rock flows, as in deformation of soft sediments prior to lithification and under the tremendous heat and pressure of metamorphosis and orogenesis (mountain building). Including these effects can add variety and realism in terrain models.

The basic idea in domain distortion is to add a vector-valued fractal or non-fractal function to the evaluation point, before passing it to the terrain function. It is basically simple functional composition, as in g(f(x)). The only trick is that the argument "x" is vector-valued, i.e., (x,y) or (x,y,z), rather than scalar valued. Thus the distorting function should also be vector-valued. In practice, I get such a 3-vector from three evaluations of a scalar valued function, as in:

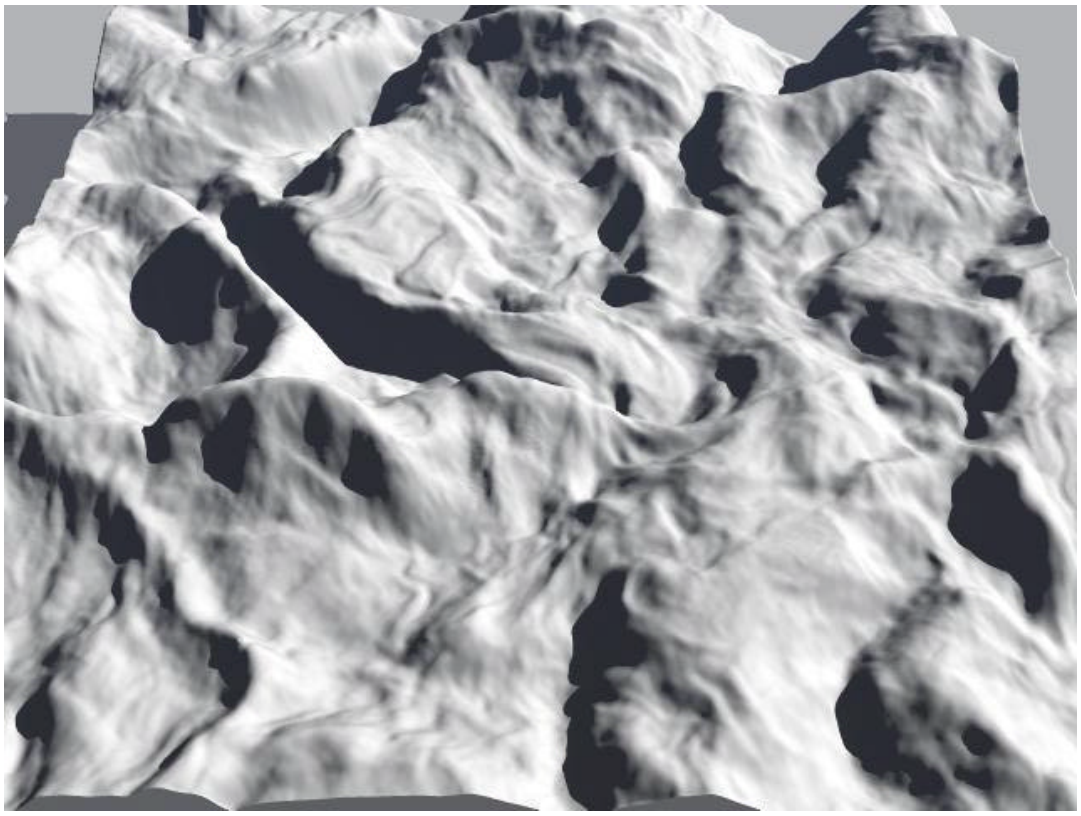
```
g( x+f(x,y,z), y+f(x+10,y,z), z+f(x,y+10,z) )
```

Here is a simple domain-distorted fBm function:

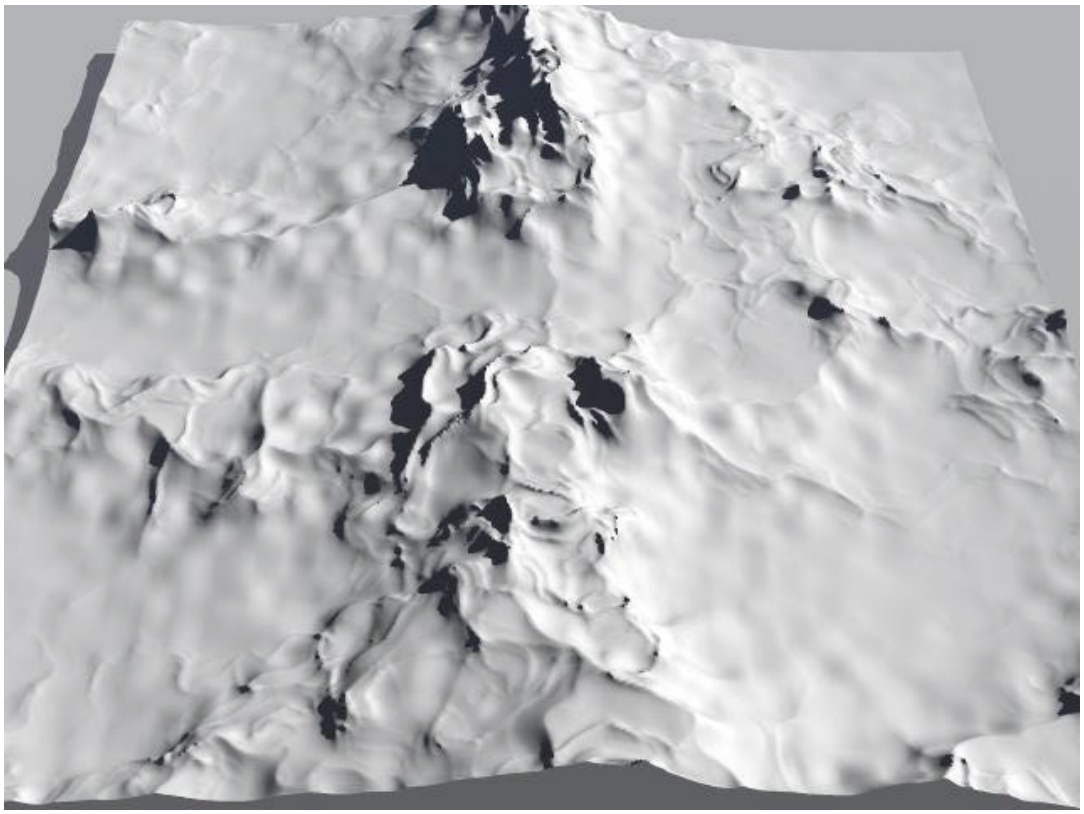
```
/* Domain-distorted fBm.
 *
 * Some good parameter values to start with:
 *
 *    H:          0.25
 *    distortion: 0.3
 */
double
WarpedFBm( Vector point, double H, double lacunarity,
           double octaves, double distortion )
{
      double      Noise3();
      Vector      tmp, distort;
      int         i;

      /* compute distortion vector */
      tmp = point;
      distort.x = fBm( tmp );
      tmp.x += 10.5;
      distort.y = fBm( tmp );
      tmp.y += 10.5;
      distort.z = fBm( tmp );

      /* add distortion to sample point */
      point.x += distortion * distort.x;
      point.y += distortion * distort.y;
      point.z += distortion * distort.z;

      return( fBm( point ) );
```

} /* WarpedFBm() */

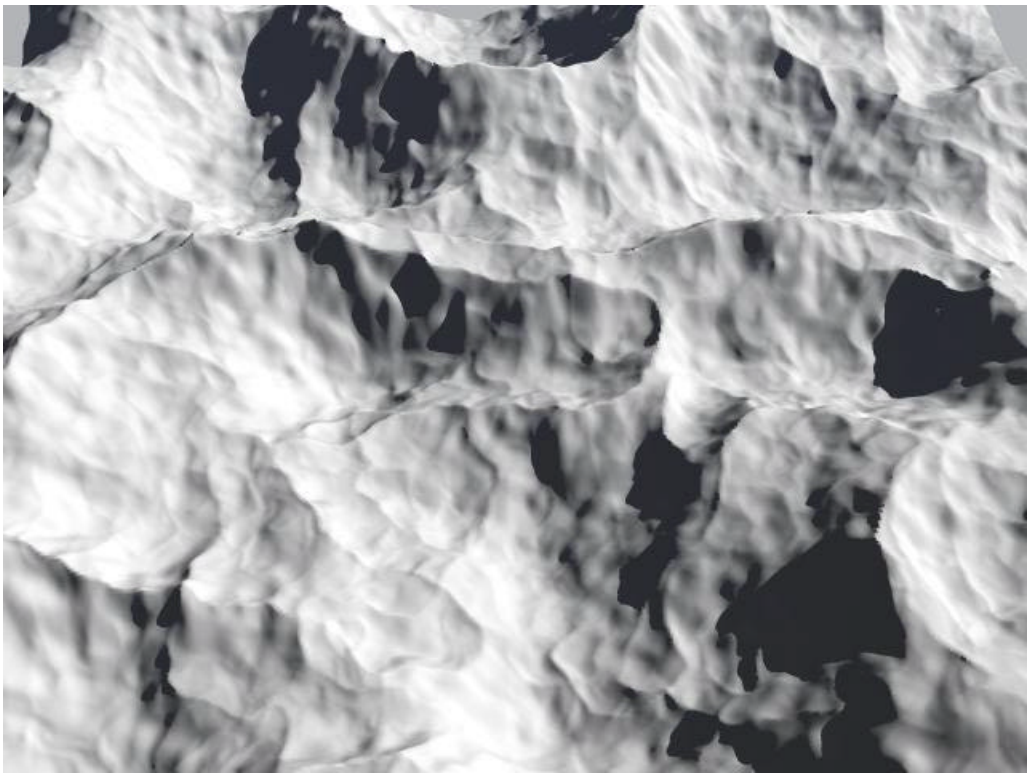
In the above case, we have fBm distorted with fBm. This particular function can be seen in the "lava" terrain model in Bryce 4. The distortor and distortee functions need not be the same fractal. I have experimented with a variety of domain distortions in terrain models: "warped ridges," "warped slickrock," "weathered dikes" and "warped zorch" are examples appearing in Bryce 4. "Zorch" is a general fractal terrain that is built from a randomly selected basis function; "warped zorch" is a random terrain distorted by a fractal built from another randomly-selected basis function. It's fun to see what kind of "looks" you get from such randomization of basis functions and other fractal parameters.



A sample of the “lava” terrain model in Bryce 4: fBm distorted with fBm.



A sample of the “warped ridges” terrain model in Bryce 4: the “ridges” model distorted with fBm.



A sample of the "warped slickrock" terrain model in Bryce 4: fBm constructed from one minus the absolute value of Perlin noise, distorted with fBm.

An example of the "warped zorch" terrain model in Bryce 4: fBm constructed from some randomly-selected basis function, distorted by fBm constructed from another randomly-selected basis function. There is a tremendous variety of possible results in this scheme.

2.6 Slumping: Forming Talus Slopes

Many terrains are composed of ridges between drainage channels, where the slopes of the relief are close to a fixed *angle of repose* or *talus slope*. Santa Catalina Island off the coast from Los Angeles, California, is a good example. The terrain is eroded away by water flowing through the channels. The strength of the substrate only allows it to support faces up to a certain angle; when this angle is exceeded a landslide occurs, bringing the face to or below the critical angle. In such terrains, fractal character is evident in the distribution of the drainage channels and on small scales on the terrain surface. The faces of the mountains, all at or near a given angle of repose, are not particularly fractal.

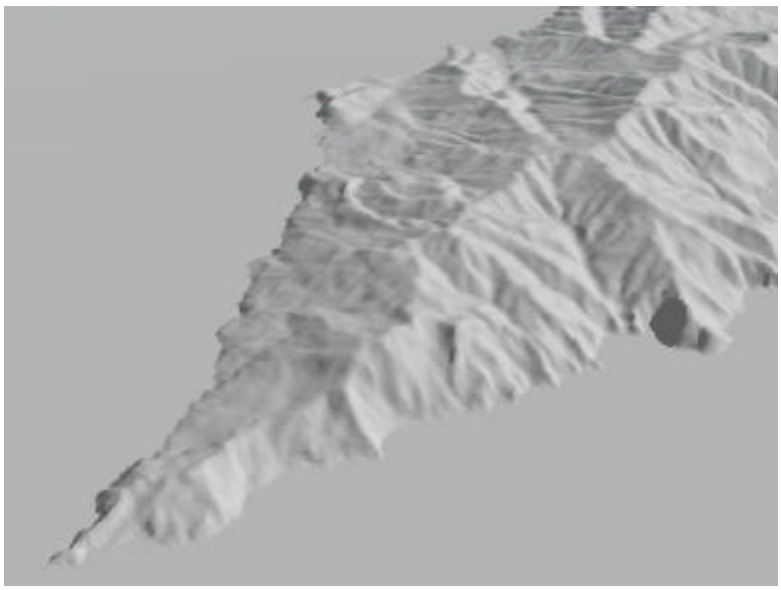
One can emulate the angular character of such terrain by applying a modified low-pass filter to steeper terrain models. This can be viewed as a physical model of *diffusive* erosion. The filter is exactly like a standard low-pass filter, except that the value to which it converges is not a DC signal but rather, for instance, a slope of 45 degrees. Slopes less than the angle of repose are unaffected.

It is implemented with code like this:

```
delta = [difference of adjacent samples] / [sample spacing]
talus_moved = delta - talus_slope;
if ( talus_moved > 0.0 )
    talus_moved *= diffusion_coeff;
else
    talus_moved = 0.0;
```

The value of `talus_slope` is the derivative of the angle of repose (sign depending on the sign of `delta`). The value of `diffusion_coeff` should be positive but much less than 1.0; it controls the rate of diffusion.

Applying this filter iteratively to a steep terrain model causes the terrain to "slump" down to the specified angle of repose.



Santa Catalina Island, rendered in Bryce from a USGS DEM. Note the fairly constant angle of the slopes of the terrain.

2.7 Fluvial Erosion

What's missing in the above model is the fractal network of drainage channels between the ridges created by the slumping filter. We addressed fluvial erosion simulation in a fairly ad hoc model in 1989. [4] Since then, we have expended much effort in developing a more physically accurate simulation informed by models from the literature of fluvial geomorphology.

Here is the bad news: These more accurate models are highly nonlinear and the partial differential equations one needs to solve for the transport and erosion are particularly nasty, so the simulations must employ very small time steps to remain stable. This makes the computational cost prohibitive as we're out to simulate the passage of geologic time, implying that our simulation needs to work much faster than Nature, which is incredibly slow and patient in comparison. While this certainly is much faster than Nature, it's still not fast enough for our purposes in image synthesis. Such simulation also has the significant drawback of requiring a fixed post spacing in the height field, making LOD rendering problematic.

As computers get faster and memory cheaper such simulations will become more feasible. But they are complex, both in the physical model of rainfall, fluid transport, substrate, erosion and deposition, and in the numerical methods required to solve the PDEs. The upshot is that, while they will definitely continue to be of interest as the only known way to create (realistically) the context sensitive fractal drainage networks, they will remain impractical for the foreseeable future.

Even if we had an efficient working model, users would find the number of parameters driving the model intimidating. Any somewhat realistic model will have on the order of 100 such parameters. Searching a 100-dimensional parameter space for a desired result is a very difficult task—particularly when it takes minutes to days to evaluate the results of a given set of parameter values.

Also, the literature of formal models of fluvial geomorphology generally does not even address deposition—issues of erosion and transport complicate the models enough, without addressing deposition. So even our "most accurate" physical model (I use the quotation marks because these published models have not been confirmed, computationally) must become ad hoc when including the very important phenomenon of fluvial deposition.

The most striking feature of the fluvial erosion models is their complexity and inefficiency, as compared to the fractal terrain generation algorithms, in creating natural fractals that are to human perception less important in achieving the impression of realism in synthetic terrain models. That is, we are generally less aware of the fractals in the drainage networks than in the roughness of the terrain. Mandelbrot always scoffed at my fluvial erosion models—"Not enough bang for the buck!" he would say. After a few years of struggling with these physical models, I began to appreciate his point. Now I firmly believe it is well taken.

2.8 Conclusions

Terrains are more geometrically complex than we can hope to reproduce with reasonably elegant and usable models, but the simple complexity of fractals gives us an initial handle on describing such complexity. As Voss illustrated in the 1970's, fractional Brownian motion can provide a convincing terrain model, if limited to local models where the terrain has the same statistics everywhere. The procedural, heterogeneous fractal models presented here and elsewhere [1] extend the expressive power of fBm without compromising its elegance. Simulating diffusive erosion is fast and easy. Simulating fluvial and glacial erosion, probably the most important morphogenic forces besides uplift acting on terrains, remains impractical.

There is plenty of work yet to be done in developing fractal models of natural phenomena. Turbulence has yet to be modeled, realistically, more elegantly than by solving the Navier-Stokes equations. Trees are distinctly fractal, yet subtleties in their randomness continue to escape us. Similarly, people often say these terrain models "look so real," yet they actually bear little resemblance to the true form of terrains in Nature which are both more complex and utterly different in their character, being dominated by context-sensitive erosion features not captured by fBm-based fractal models. Such features can be added—only on a local scale—at great cost in intellectual and programming effort as well as in computation time. But local models are insufficient, and once again Nature's complexity escapes us.

Our main artistic advantage in this endeavor is people's profound insensitivity to the true appearance of Nature—we monkeys are easily fooled here. Try simulating the appearance of human faces, and you are likely to have a very different experience: The human brain appears like a neural net that is most exquisitely sensitive to the subtleties in that particular height field. I, for one, am grateful that my personal predilection has been to work at modeling natural phenomena, where the room for error in "realistic" models is huge. And besides, the average pretty synthetic landscape is better looking than the average pretty synthetic face. For the time being.

References

[1] Ebert, D.S., ed. *Textures and Modeling: A Procedural Approach*. 2 ed. 1998, Academic Press: Cambridge, MA.

[2] Mandelbrot, B.B., *The Fractal Geometry of Nature*. 1982, New York: W. H. Freeman and Co.

[3] Musgrave, F.K., *Methods for Realistic Landscape Imaging*. 1994, Ann Arbor, Michigan: UMI Dissertation Services (Order Number 9415872).

[4] Musgrave, F.K., C.E. Kolb, and R.S. Mace, *The Synthesis and Rendering of Eroded Fractal Terrains*. Computer Graphics, July, 1989. **23**(3): p. 41-50.

[5] Peitgen, H.O. and D. Saupe, ed. *The Science of Fractal Images*. 1988, Springer-Verlag: New York.

[6] Perlin, K., *An Image Synthesizer*. Computer Graphics, July, 1985. **19**(3): p. 287-296.

[7] Worley, S. *A Cellular Texture Basis Function*. in *SIGGRAPH 96*. 1996. ACM SIGGRAPH.