14

A BRIEF INTRODUCTION TO FRACTALS

F. KENTON MUSGRAVE

Our world is visually complex. Achieving realism in computer graphics is largely a matter of reproducing that complexity in our synthetic images. *Fractal geometry* is our first cogent language of visual complexity; it provides a potent vocabulary for complex form, particularly the kinds of forms found in nature. Fractal geometry can map seemingly chaotic complexity into the terse, deterministic idiom of mathematics—simple equations that efficiently encapsulate lots of complexity. Furthermore, the way in which fractals encapsulate this complexity is exquisitely suited to the capabilities of the digital computer (unlike much mathematics). Computer graphics, for its part, can translate the fractal mathematical abstractions into the single form best suited to human cognition: images.

Fractals and computer graphics have grown up together, and both are relatively new disciplines. This is partly because the study of fractals is simply not possible without computer graphics: fractals are too complex to comprehend except through pictures and too tedious to create except with a computer. Thus computers have always been essential to the study of fractals. For their part, fractals have always been the source of much of the visual complexity in realistic computer graphics. Fractals are particularly salient in synthetic images of nature such as landscapes, while fractal textures are often used to add visual interest to relatively simple and boring geometric models. Fractals can even comprise abstract art in themselves, as Figures 14.1, 19.4, and 19.5 illustrate. It is safe to say that fractals have been the source of much of the beauty in synthetic images throughout the brief history of computer graphics. Fractals and computer graphics go hand in hand.

This chapter provides a brief overview of fractal geometry. It is designed to be a sort of "fractals for artists" discussion of the relevant issues. I have tried hard to be accurate in the details while avoiding mathematical technicalities, knowing how stultifying they can be. The mathematics is covered in exquisite detail in the excellent text *The Science of Fractal Images* (Peitgen and Saupe 1988) if you're interested.

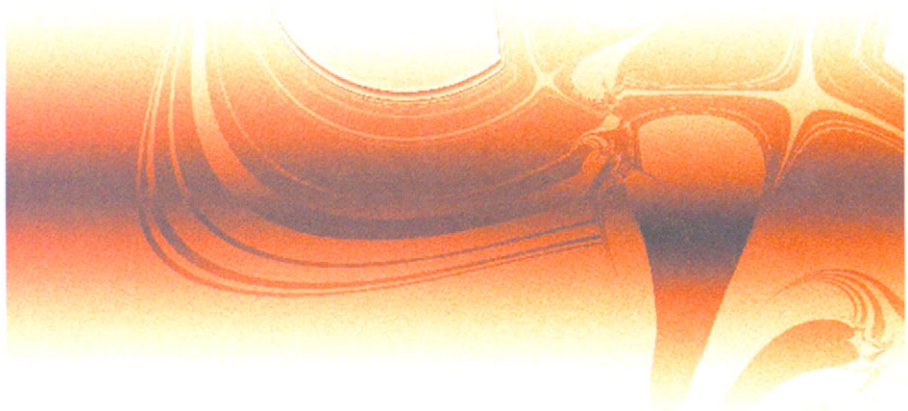

FIGURE 14.1 "Genetic Sand Painting" is a procedural texture "grown" using Karl Sims's genetic software. Copyright © 1994 F. Kenton Musgrave.

The importance of this chapter lies in the introduction of two fractal constructions and the code segments that implement them. Let me state up front that the second, pure multifractal, function may safely be ignored as simply a mathematical curiosity, but the first is *the* primary building block for all of the constructions I will present in later chapters. So, whatever your level of technical and/or mathematical competence, I urge you to read on. I hope you enjoy the discussion; I have attempted to make it provocative as well as informative.

Finally, a warning: Don't expect to get all this your first time through. Expect to have to read it maybe three times before you really get it. It took me at least that many times when I was first grappling with the weirdly counterintuitive concepts behind fractals. Don't feel slow or dense—you're not alone! I like to tell my students, "Many things are obvious—after you've thought about them for several years." So take your time. There's some major conceptual weirdness coming right up.

WHAT IS A FRACTAL?

As important as fractals are to computer graphics, they have always been widely misunderstood in our field. At times they have even become a source of heated controversy. Usually, in my experience, the problems arise mainly from incomplete

knowledge of exactly what fractals are and are not. They *are* a potent language of form for shapes and phenomena common in nature. They are decidedly *not* the end-all for describing all aspects of the world we inhabit, or for creating realistic synthetic images. They encompass simultaneously more and less than what many people think. I will attempt to illuminate the concept of a "fractal" in the following, and also to make a first cut at delineating both the power of fractal geometry and at least some of its limitations. Having worked with Benoit Mandelbrot, the father of fractal geometry, for six years, I may be qualified to present the topic and to address some of the misunderstandings of fractals that are common in our field.

Fractal geometry is mathematics, but it is a particularly user-friendly form of math. In practice, it can be approached entirely heuristically: no understanding of the underlying formulas is required, and the numerical parameters of the user interface may be used intuitively, their exact values being more or less arbitrary "positions of a slider," a slider that controls some aspect of visual behavior. For the purposes of this text, I will develop this heuristic approach to fractals in computer graphics. For my own purposes, I think entirely visually; the equations simply describe shapes and rules for their combination. Works for me! I am *not* a mathematician—rumors to the contrary are greatly exaggerated.

What, then, *is* a fractal? Let me define a fractal as "a geometrically complex object, the complexity of which arises through the repetition of a given form over a range of scales."[1] Note the simplicity and breadth of this definition. This simple, heuristic definition will be sufficient for our purposes here.

Probably the easiest way to think of fractals is as a new form of symmetry—dilation symmetry. Dilation symmetry is when an object is invariant under change of scale—zooming in or zooming out. This invariance may be only in gross appearance, not in the exact details. For example, clouds exhibit dilation symmetry in that a small part of a cloud looks like a larger part, but only qualitatively, not exactly. Tree branches, river networks, and lightning display this dilation symmetry, too. And all are fractal.

In my current worldview there are at least two kinds of complexity: fractal and nonfractal. Nonfractal complexity seems to be characterized by the accumulation of a variety of features through distinct and unrelated events over time—like the scuffs, holes, and stains on an old pair of shoes, for instance. Because of the independence of the events that create the features that comprise this complexity, it is hard to characterize it succinctly. Fractal complexity, on the other hand, can be very simple: just keep repeating the same thing over and over, at different scales.

1. No tricks here: "scales" simply means "sizes."

Our heuristic definition is sufficient, but let me explain a little further for your edification. Fractals have a peculiar property called *fractal dimension.* We are all familiar with the Euclidean integer-valued dimensions: zero dimensions corresponds to a point, one to a line, two to a plane, and three to space. Fractal dimension extends this concept to allow real-numbered values such as 2.3. Visually, values of fractal dimension between the integer values (for example, 2.3, which lies between 2 and 3 dimensions) provide a continuous "slider" for the visual complexity of the fractal. Something with a fractal dimension of 2.0 is (at least locally) planar, and as the value of the fractal dimension rises from 2.0 to 3.0, that plane becomes rougher and rougher—and more visually complex—until it densely occupies (at least locally) some volume of three-dimensional space. I warned you that this would get weird!

Again, the "whole" component of the fractal dimension, or the 2 in 2.3, indicates the underlying Euclidean dimension of the fractal, in this case a plane. The "fractional" part, for example, 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 densely filling some local 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.

Why do I keep qualifying these statements with the word "locally"? Because while a beach ball is, for example, a three-dimensional object, its surface is not: if we zoom in infinitely close, it becomes "locally" planar. Sure, it's a bit of a fine mathematical point, but understanding it and keeping it in mind is necessary to help avoid even more confusion than is inevitable in learning what fractals are. So a surface doesn't have to be a perfectly flat Euclidean plane to have a fractal dimension of 2.0, nor does it have to fill all of three-space to have a fractal dimension of 3.0. This one took me a while to get my head around, too. You're not alone.

Fractal dimension is a peculiar property, to be sure. The mathematical definition of it involves infinity; therefore, I claim, the human mind simply cannot fully comprehend it. But you quickly become numb (well, after a few years maybe) to the incomprehensibility, and furthermore, for our purposes here it is beside the point. So let us not belabor the technical details; rather, let us continue in developing an intuitive grasp of fractals.

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.* While that function can be literally anything, for most of my constructions it is a variant of Ken Perlin's *noise* function.[2] For now, think of the noise function as providing a kind of

2. I generally prefer to use "gradient noise," as described by Darwyn Peachey in Chapter 2.

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 fancy Latin word for "gap." The gap, in this case, is between successive frequencies in the fractal construction. In practice, lacunarity is pretty much a nonissue, as we almost always leave it set at 2.0 (although in Chapter 6 Steve Worley describes some cases where you might want to change that value slightly). In music, doubling the frequency—which is exactly what a lacunarity value of 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 in smaller lumps to bigger lumps.

There is a well-defined relationship between the amount by which we scale size and the amount by which we scale the height, or frequency versus amplitude. This relationship is what determines the fractal dimension of our result. (Again, I decline to get any more technical and refer the interested reader to *The Science of Fractal Images* for details.) The particular kind of fractal we're building is called *fractional Brownian motion,* or fBm for short. fBm is characterized by its *power spectrum,* which charts exactly how amplitude relates to frequency. Oops! Pardon me—I'll knock off the math.

Allow me now to point out two of the most common misconceptions about fractals in our field of computer graphics. The first is that all fractals are variants of fBm. Not so! Go back and look at our definition of fractals: it subsumes a far larger class of objects than our scaled-and-added-up cottage cheese lumps. Many things are fractal; more interestingly, perhaps, many things are "only sort of" fractal. Fractalness is a property that is, believe it or not, best left loosely defined. Think of it as a quality like color: it is hard to define "blue" precisely, and any really precise definition of "blue" is likely to be overly restrictive in certain circumstances. The same goes for fractals.

The second common misconception about fractals is this: that an object must have infinite detail in order to qualify as fractal. Also not so! Benoit and I agree that to talk about self-similarity over a range of less than three scales is rather vacuous. So we may heuristically constrain fractals to be "objects that display self-similarity at a minimum of three separate scales." This magic number three may not even provide such a great definition of "fractal"—later I will describe a nice model of water

that can use only two octaves of noise. Is it fractal, or is it not? You could make a convincing argument either way. The main point is this: as long as something displays self-similarity over some, albeit perhaps small, range of scale, it may qualify as "fractal." Note that *all* fractals in nature exhibit their fractal behavior over a limited range of scale—even the large-scale cosmological structure of the universe. The distribution of galaxies and clusters of galaxies is quite fractal, but there *is* a finite and observable largest scale of features in this universe (knowledge more recent than the first edition of this book!). Another example: Seen from a distance in space, Earth is smoother than a glass marble, yet on smaller scales it has many mountain ranges that are quite fractal.

We refer to the size above which self-similarity ceases to manifest itself as the *upper crossover scale*. Similarly, there is a smaller size below which self-similarity no longer is manifest; this is the *lower crossover scale*. All fractals in nature, then, are what we call *band-limited*—they are fractal only over some limited range of scales. Mandelbrot makes this striking observation: the Himalayas and the runway at JFK have approximately the same fractal dimension (i.e., roughness)—they differ only in their crossover scales! (Didn't I warn you this would get weird?)

Finally, I'd like to note that all fractals for computer graphics must also be band-limited, for two reasons: First, spatial frequencies that are higher than half our pixel frequency (the screen width divided by resolution) may violate the Nyquist sampling limit and cause aliasing (a highly technical point; feel free to ignore it here and when it comes up again later, but hugely important to our ultimate goal of building the realistic fractal planets that will become cyberspace). Second, we generally wish for our computations to terminate, so it's poor programming practice to put in the kind of infinite loop that would be required to construct non-band-limited fractals.

WHAT ARE FRACTALS GOOD FOR?

Again, the world we inhabit is visually complex. When synthesizing worlds of our own, *complexity* equals *work*. This work can be on the part of the programmer/artist or the computer; in fact it will always be some of each. But there is a balance to be struck, and personally I prefer to have the computer do most of the work. Usually, it seems to have nothing better to do, while I generally can find other mischief to make. One of the defining characteristics of procedural modeling, in general, is that it tends to shift the burden of work from the programmer/artist to the computer. Complexity is vital to realism in synthetic images. We still have a long way to come in this area: I claim that you'd be hard put to find any scene in your everyday environment, other than a clear blue sky, that's as visually simple as the best, most detailed synthetic image of the same thing (if for no other reason than that reality is higher

resolution). How, then, can we close the gap? To date, fractals are our best tool. While not all complexity in nature comprises the repetition of form over different scales, much of it is. It is not so much true that nature is fractal as that fractals are natural. Fractal models can be used to construct scenes with good realism and a high degree of visual complexity, but they effectively address only a limited range of phenomena. People often ask me if I can model dogs and people with fractals and the answer is "no." Dogs and people simply aren't self-similar over a range of scales. Fractals, then, allow us to model certain things well: mountains, clouds, water, even planets. Non-self-similar complexity such as hair and grass require other methods. Things not visually complex, obviously, do not require fractal geometry for their reproduction.

All my time with fractals has led me to view them as "the simplest conceivable form of complexity." (If you can think of a simpler way to give rise to complexity than to simply repeat the same thing over and over at a variety of scales, I'd love to hear about it!) This simplicity is a very good thing, since computers are simpletons, and simpler programs are better programs, too: they are quicker to write, (usually) easier to understand, easier to trust, and take up less memory.[3]

Now let me answer the question "What are fractals good for?" Many natural phenomena are fractal. Mountains are perhaps the best-known example: a smaller part of a mountain looks just as mountainlike as a larger part. They're not exactly the same; this is the distinction between *statistical self-similarity,* where only the statistics of a random geometry are similar at different scales, and *exact self-similarity,* where the smaller components are exactly the same as the larger ones, as with the equilateral triangles forming the famous von Koch snowflake fractal. Trees, river systems, lightning, vascular systems in living things—all have the character of statistical self-similarity: smaller branches tend to resemble larger branches quite closely. Another example, on which we capitalize quite heavily in this book, is turbulent flow: the hierarchy of eddies in turbulence was known to be self-similar long before Mandelbrot elucidated the concept of "fractal." There's even a little poem about it:

> Bigger swirls have smaller swirls,
> That feed on their velocity,
> And smaller swirls have smaller swirls,
> And so on, to viscosity
> —Lewis F. Richardson, 1922

3. This bias in favor of simplicity is canonized in Occam's Razor: "The simpler model is the preferred model." This principle has guided much of science and engineering through the centuries. More on this later.

The essential fractal character of turbulence allows us to use fractal models to represent clouds, steam, global atmospheric circulation systems on planets, even soft-sediment deformation in sedimentary rock. Again, the pictures attest to the success of these models, as well as the limits to that success.[4]

FRACTALS AND PROCEDURALISM

How are fractals and proceduralism related? Very closely indeed. When we describe how to build fBm, it will be with an iterative, constructive procedure. The simplicity of fractals resonates well with computers, too, as they are simple-minded devices. The sort of simple, tedious, repetitive operations required to build a fractal are exactly the kind of thing computers do best. As we will see, fractal constructions can provide potentially unlimited visual complexity that issues from a relatively small amount of code.

Alvy Ray Smith called this complexity-from-simplicity *amplification* (Smith 1984)—a small input provides a wealth of output. We engineer it in the classic proceduralist method: by shifting the burden of work from the human designer to the computer. Mandelbrot himself points out that fractals have been evident to mathematicians for some time, but not until the advent of computer graphics did they have the tools necessary to investigate them. This is almost certainly why fractals were only recently "discovered"—perhaps more like "fleshed out"—and why it was a researcher (Mandelbrot) at IBM (a computer company) who did so. Fractals and computers have always been inextricably linked. The fruit of this symbiosis is illustrated in the realistic synthetic imagery issuing from fractal models and by the observation that most other complexity in computer graphics derives either from captured data from the real world (cheating!) or from the hard labor of constructing detailed models by hand (which is antiproceduralist).

PROCEDURAL fBm

But enough lecturing already. Let's see exactly how to build the archetypal fractal procedural texture: fBm (also known as "plasma" in some software applications). It's pretty simple:

---

4. For instance, as our poem points out, turbulence is actually composed of a hierarchy of *vortices,* not simply lumps like the *noise* function provides. To my knowledge, no one has yet developed a procedural vortex turbulence model that competes well with physical simulations using the Navier-Stokes equations that yield nice fractal vortices, but at a high computational cost.

```
/*
 * Procedural fBm evaluated at "point".
 *
 * Parameters:
 * "H" is the fractal increment parameter
 * "lacunarity" is the gap between successive frequencies
 * "octaves" is the number of frequencies in the fBm
 */
double fBm( Vector point, double H, double lacunarity, double octaves )
{
     double         value, remainder, Noise();
     int            i;

     value = 0.0;

     /* inner loop of fractal construction */
     for (i=0; i<octaves; i++) {
     value += Noise( point ) * pow( lacunarity, -H*i );
          point *= lacunarity;
     }

     remainder = octaves - (int)octaves;
     if ( remainder ) /* add in "octaves" remainder */
          /* 'i' and spatial freq. are preset in loop above */
          value += remainder * Noise3( point ) * pow( lacunarity, -H*i );
     return value;
}
```

Note the simplicity of this routine: the fractal itself is constructed in the two-line inner loop; the rest of it is concerned with the picayune point of dealing with the remainder of `octaves`. (This will be important when building fractals with continuous level of detail, or LOD, in later chapters.[5]) Again, this routine is a generalization of Perlin's original "chaos" function. I've added new parameters to control lacunarity (which in most cases can simply be fixed at 2.0, as Perlin originally did), the fractal increment parameter H, and the number of octaves in the construction.

---

5. For a properly band-limited fBm with pixel-level detail, imaged from a distance of 1.0, the correct number of octaves to use is

$$\texttt{octaves} = \log_2(\texttt{screen\_resolution}) - 2$$

or a value of about 6 to 10. The -2 term in this expression has to do with the facts 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 $\log_2(1/4) = -2$. Reducing the number of octaves can speed up rendering time in exchange for less detail.

To accommodate LOD in advanced applications, this function is designed to accommodate real-valued octaves. The fractional part of the value of parameter `octaves` is added in, linearly, after the main loop. We'll use this to adaptively band-limit textures where we want to link the number of octaves to distance to avoid exceeding the Nyquist limit and subsequent aliasing. (We will utilize this in the QAEB algorithm in Chapter 17.) This little trick with the octaves remainder avoids discontinuities in the texture that would be introduced were the number of octaves to abruptly change at some threshold.

If you implement something like adaptive band-limiting, you'll want to store the spectral exponents in a preinitialized array, to avoid repeated calls to `pow()` in the inner loop. This exponent array would store the amplitude-scaling values for the various frequencies in the construction. These weights, a simple function of H and the lacunarity, determine the fractal dimension of the function. (Again, see *The Science of Fractal Images* for details on how and why.)

The parameter H is equal to one minus 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 approaches white noise. Figure 14.2 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`.[6]

MULTIFRACTAL FUNCTIONS

The fBm described above is, at least as closely as we can make it (see Chapter 6 for more on this), statistically *homogeneous* and *isotropic*. Homogeneous means "the same everywhere," and isotropic means "the same in all directions" (note that the two do not mean the same thing). Fractal phenomena in nature are rarely so simple and well behaved. For instance, synthetic fractal mountains constructed with a single, uniform fractal dimension everywhere—as are most fractal mountains in computer graphics—have the same roughness everywhere. Real mountains are never like that: they typically rise out of plains and have rolling foothills at their feet. For

6. A definition of *vector:* an ordered set of numbers, usually three numbers for our purposes, which defines an arrow from the origin (the origin being [0,0,0] in three dimensions) to the point in space that the set of numbers specifies. The dimensionality of the space is the same as the number of elements in the vector. Thus the vector [1,−1,1] defines an arrow in three dimensions that points out from the origin at 45 degrees to each of the x-, y-, and z-axes. Its length is $\sqrt{x^2 + y^2 + z^2} = \sqrt{1^2 + (-1)^2 + 1^2} = \sqrt{3}$.

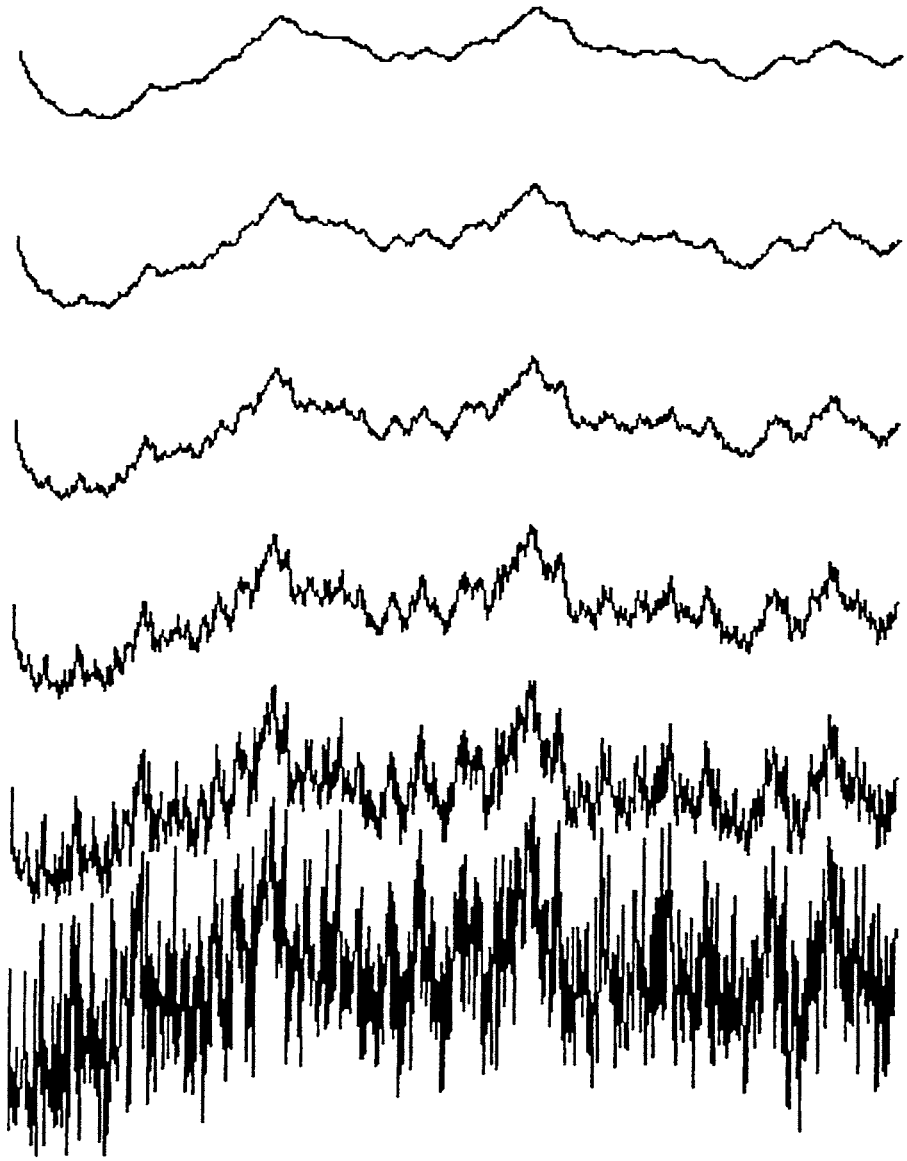

FIGURE 14.2 Traces of fBm for H varying from 1.0 to 0.0 in increments of 0.2.

this and other concerns of realism, we desire some more interesting, heterogeneous fractal functions.

Enter *multifractals*. Multifractals may be heuristically defined as "fractals that require a multiplicity of measures, such as fractal dimension, to characterize them." Another heuristic definition is "heterogeneous fractals, the heterogeneity of which is invariant with scale."[7] They are most easily thought of as fractals whose dimension varies with location; the key point is that they are heterogeneous—not the same everywhere. Later I will demonstrate some terrain models with plains, rolling foothills, and jagged alpine mountains, all of which issue from a single function that is only a little more complicated than the basic fBm described earlier.

One mathematical definition of multifractals ties them to *multiplicative cascades* (Evertsz and Mandelbrot 1992). The earlier fBm function is built using an *additive cascade*. The formal difference between an additive and a multiplicative cascade is simply that the addition of higher frequencies in the inner loop is replaced by a multiplication. Here is a multiplicative-cascade multifractal variation on fBm:

```
/*
 * Procedural multifractal evaluated at "point."
 *
 * Parameters:
 * "H" determines the highest fractal dimension
 * "lacunarity" is gap between successive frequencies
 * "octaves" is the number of frequencies in the fBm
 * "offset" is the zero offset, which determines multifractality
 */
double
multifractal( Vector point, double H, double lacunarity,
              int octaves, double offset )
{
    double    value, Noise();

    value = 1.0;

    for (int i=0; i<octaves; i++) {
         value *= (Noise( point ) + offset) * pow( lacunarity, -H*i );
         point *= lacunarity;
    }
    return value;
}
```

7. The heterogeneity may be, for instance, that peaks on a terrain are rougher than valleys. This can be true at all scales; then we have a multifractal, by this definition.

Note the addition of one more argument, `offset`, to the function, over the function for ordinary *(monofractal)* fBm; other than this and the multiplication in the inner loop, this function is nearly identical to `fBm( )`. The `offset` controls the multifractality, if you will, of the function. When `offset` equals zero, the function is heterogeneous in the extreme, and as its value increases the function goes from multi- to monofractal, and then approaches a flat plane as `offset` gets large (i.e., around 100 or so). An `offset` value of ~0.8 yields a very nice, heterogeneous terrain model, as seen in Figure 14.3. One thing you must know about this function: its range varies widely with changes to the value of `offset`—by many orders of magnitude. In fact, as the number of octaves goes up, it will either converge to zero or diverge to infinity. It's just plain unstable. (Hence the parameter `octaves` is not real-valued, as this function would behave badly in LOD schemes.) If you ever use this function, you will need to take measures to rescale its output. I have accomplished this by evaluating the function over some finite patch (e.g., a 3 × 3 area sampled at 100 × 100 resolution) and rescaling the function's output by one over the maximum value in the patch.

This function has the abstract advantage of following at least one mathematical definition of "multifractal" closely. This may be desirable for mathematical research into multifractals, but it is really pretty much a red herring for those more interested in applications. Therefore, I suggest that you just ignore this construction and concentrate on the less mathematically well-defined, but better behaved, multifractal

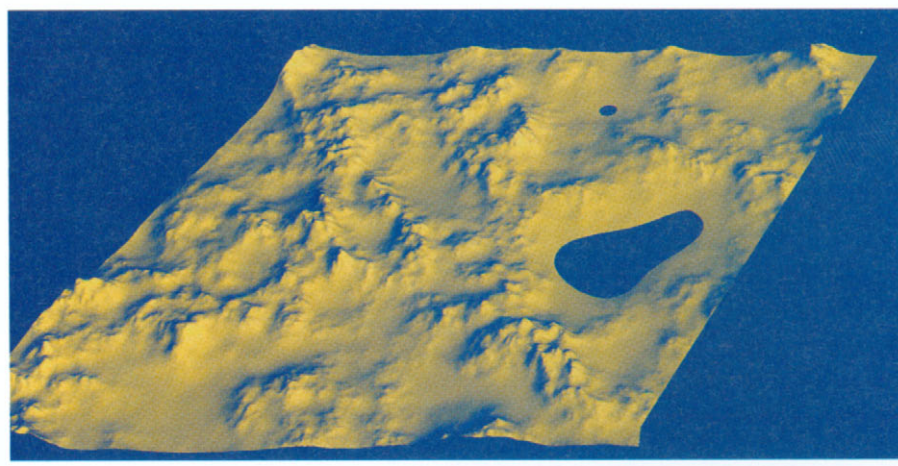

FIGURE 14.3 A multifractal terrain patch, with `offset` parameter set to 0.8. Copyright © 1994 F. Kenton Musgrave.

functions described in Chapter 16. Those functions are additive/multiplicative hybrids that we don't quite know how to characterize mathematically yet, but that have proven their usefulness in applications.

These multifractal models were developed for terrain models, and I have most often interpreted them as height fields for that purpose. But in fact they are really just more examples of procedural textures, and they can be used as such. I encourage you to experiment with them and invent your own new applications—it's a lot of fun! (For example, see Figure 15.17.) Since they were conceived as terrain models, I will explain them in that context, in Chapter 16. But keep in mind that those models can also be used as textures when applications arise.

FRACTALS AND ONTOGENETIC MODELING

Fractal models are sometimes assailed on the grounds that they lack a physical basis in reality. For example, there is no known causal basis for the resemblance between fBm and real mountains. Is this a problem? For the reductionist-minded it is, for such a model violates the reductionist principle that true validity can only be established by showing that the model issues from first principles of physical law. This is a narrow view that Mandelbrot and I, among others, would dispute. To wit, Nobel Prize–winning physicist Richard Feynman once said: "A very great deal more truth can become known than can be proven" (Brown and Rigden 1993). In practice, both physical and nonphysical modeling have their places in computer graphics. To provoke and focus this argument, I coined the term *ontogenetic modeling*. From *Webster's Collegiate Dictionary,* 10th edition:

> *ontogenetic:* . . . 2: based on visible morphological characters.

I coined the term deliberately to contrast with Al Barr's "teleological" modeling (Barr 1991). Again, from *Webster's:*

> *teleology:* 1 a: the study of evidences of design in nature b: a doctrine (as in vitalism) that ends are immanent in nature c: a doctrine explaining phenomena by final causes 2: the fact or character attributed to nature or natural processes of being directed toward an end or shaped by a purpose 3: the use of design or purposes as an explanation of natural phenomena.

The underlying idea of ontogenetic modeling is that, in the field of image synthesis, it is a legitimate engineering strategy to construct models based on subjective morphological (or other) *semblance;* this as opposed to, for instance, pursuing precise (e.g., mathematical) veracity, as is a goal in scientific models, or constructing them such that they and their behavior issue from first scientific principles, in the reductionist tradition.

My point is that we computer graphics professionals are engineers, not scientists. The goal of engineering is to construct devices that do something desirable. The goal of science is to devise internally consistent models that reflect, and are consistent with, the behavior of systems in nature (which are entirely unrelated to such models) and to make verifiable predictions of the behavior of nature based on the behavior of the model. Science informs engineering. But engineering is an ends-driven discipline: if the device accomplishes what it is intended to accomplish, it works and is therefore good. The means by which it accomplishes its end are of secondary importance.

Elegance[8] in the model is nice in engineering, while it is necessary in science. Occam's Razor (i.e., the metaphysical prejudice that "the simpler solution is the preferred solution") is equally applicable in both science and engineering. Engineers call it the KISS principle: "Keep it simple, stupid."[9]

In engineering—specifically, in computer graphics—Occam's Razor often recommends ontogenetic models. Science is going for Truth and Beauty; we're after beautiful pictures that look like things familiar. The degree of preoccupation with accuracy and logical consistency that marks good science is not admissible in our engineering discipline: we have a job to get done (making pictures); getting that job done efficiently takes precedence. Scientific models, or "physical" models in computer graphics parlance, do not generally map well into efficient algorithms for image synthesis. Given the aims and methods of science, this is not surprising. Algorithmic computability and/or efficiency are not considerations in constructing scientific models. And concerns for efficiency invoke a new, different set of constraints that may be orthogonal to the considerations that shaped any given scientific model. For instance, no scientist would hesitate to use an integral with no closed-form solution in a model—the lack of a mechanism to obtain an exact evaluation is

8. Again from *Webster's:* "elegance: scientific precision, neatness, and simplicity."

9. My mentor in landscape photography, Steve Crouch, put it another way: referring to composing an image in the viewfinder, he said "See what you can get out of the picture," not what you can get into it.

orthogonal to, and in no way compromises the validity of, the model. In the engineering discipline of image synthesis, we *must* be able to complete our computations, faster is better, and if it looks good enough, it *is* good enough.

As the late Alain Fournier put it: When you use a physical model for image synthesis, you often waste time computing the answers to questions about which you care not. That is, they do not contribute to the image. Excessive accuracy in illumination calculations is an example, given the fact that we quantize our illumination values to at most 256 levels in most standard output formats.

Given the serious drawbacks and complications of physically based models of natural phenomena, I claim that the ontogenetic approach remains, for the foreseeable future, a viable alternative approach to engineering the synthesis of realistic images. Ontogenetic models tend to be—indeed *ought* to be—simpler and more efficient, both in programming and execution time, than corresponding physical models.

[Editorial note, circa 2002: I got on my soapbox to deliver that rant about 15 years ago, when I was a graduate student and this was a hot topic. By now it seems pretty dated and my passion, well, quaint. But the basic points remain valid and well taken, I think. —FKM]

CONCLUSION

I hope I've helped you establish an intuitive understanding of fractals in this chapter. The level of understanding presented here is all that is really required to pursue applications. You don't need to be a mathematician to create and use fractals; in fact, my experience indicates that an artistic eye is far more important than a quantitative facility for creating striking fractal models for synthetic imagery. Keep in mind that fractals are the simplest and easiest—for the human operator, at least—way to generate visual complexity, whether it be geometric detail, as in landscapes, or visual detail, as with procedural textures. Fractals represent a first step toward procedurally elegant descriptions of complexity in the natural world.

To work effectively with fractals, you need to be familiar with the heuristic definition of *fractal dimension*—merely that it corresponds to roughness or wiggliness. You need to be aware of the idea of *octaves*—the number of scales at which you're adding in smaller details. Also, you may occasionally find it helpful to be familiar with the concept of *lacunarity*—the change in scale between successive levels of detail—although you probably will never have need to use this knowledge.

Finally, it is helpful to understand that most fractal constructions we use in computer graphics are *monofractal*—that is, they are homogeneous, and for that reason, may become a bit monotonous and boring. *Multifractals* can provide a second step toward capturing more of the true complexity abundantly manifest in nature. Turbulence, for instance, is a decidedly multifractal phenomenon.

With these elements of understanding in place, let us proceed to applications and see how fractals may be used in procedural models.