20

MOJOWORLD: BUILDING PROCEDURAL PLANETS

F. KENTON MUSGRAVE

INTRODUCTION

This chapter is derived from the first chapter of the MojoWorld manual. MojoWorld is a software product that builds and images fractal planets. These planets are entirely procedural, unless augmented with imported data. They are fractal. MojoWorld employs most of the tricks and techniques described in Chapters 14–19 of this book to create images with unprecedented detail (see Figures 20.11, 20.16, 20.17, 20.42, and 20.43). These images are sometimes realistic, sometimes surrealistic, and sometimes outright abstracts. As I've mentioned, my life's work is about revealing a synthetic universe. MojoWorld is our first cut at that.

Are we building this universe, or are we discovering it? That's a fascinating question. Sure, we won't stumble on it by accident; rather, we'll conjure it forth through a lot of intelligent and hard work. But, this "synthetic" universe already exists. It always has and always will. Just as 2 is the eternal result of the computation 1 + 1, presumably for all time and space in this universe, these places, as three-dimensional models, and the images we make of them, are the result of a more complicated but similarly deterministic calculation. They may already have been imaged and explored by other intelligent life-forms elsewhere and elsewhen in this universe. Their complexity and dependence on an enormous series of more or less arbitrary decisions by the programmers who wrote MojoWorld makes it extremely unlikely that any place or image we make/discover in MojoWorld has been seen, made, or discovered anywhere else by anyone else. But they have this eerie quality of "being there," just waiting to be discovered and explored (see Figure 20.1). This is surely the apotheosis of proceduralism.

Why include this chapter in this book? First, it explains fractals yet again. In all my years of experience as a teacher of fractals, I've found it reliably takes about three

FIGURE 20.1 *Metallichron: Brassy Rise* is an image from one of the planets that MojoWorld users shared, explored, and imaged in the weekly challenge at *www.3dcommune.com*. This kind of self-organizing community event is part of the MojoWorld experience. © 2002 Armands Auseklis, MojoWorld by Robert Buttery.

passes over explanations of fractals before you start to "get it." It took me at least that many times, believe me! So this chapter provides a third pass at it. Second, this chapter explains MojoWorld as a learning lab in which to experiment with fractals and procedural textures. MojoWorld is one of the most potent tools to date for that purpose, and a demo version is included on this book's Web site (*www.mkp.com /tm3*) for your pleasure. Using MojoWorld in concert with your readings in this book will turn your experience from a purely intellectual reading exercise to a hands-on practical tutorial.

An inside tip: There's a part of MojoWorld we don't talk about much—the Pro UI. It's a powerful dataflow function graph editor for writing procedural shaders—a kind of visual programming language. We don't talk about it much because it's too advanced for our target market; they'd simply be flummoxed by it. But it's perfect for anyone reading this book, as it allows you to experiment with many of the methods and tricks we've described here. And, if you find you can't do what you want

with the Pro UI as is, MojoWorld has an open architecture allowing you to extend its functionality by writing your own plug-ins. MojoWorld was designed to be a high-tech sandbox for all of us to play in. We encourage you to play and experiment with MojoWorld, and we look forward to seeing what you create/discover.

This chapter ties together most of what I've written in previous chapters and describes it all in the context of a software package that's included on this book's Web site for your use. It also points the way directly to my personal vision of cyberspace, the future of the human-computer interface, as described in the final chapter. So, in a very real sense, this chapter puts it all together and points to the future. I hope you enjoy it. Let's get started with another description of fractals.

FRACTALS AND VISUAL COMPLEXITY

Nature is visually complex. Capturing and reproducing that complexity in synthetic imagery is one of the principal research problems in computer graphics. In recent years we have made impressive progress, but nevertheless, most computer graphics are still considerably less complex and varied than the average scene you see in nature. Personally, I don't expect computer graphics to be able to match the visual richness of nature in my own lifetime—there's just too much complexity and variety to be seen in our universe. But that certainly doesn't mean we shouldn't try, only that we can keep at it for a long time to come.

So how do we make a first stab at creating visual complexity in synthetic imagery? In a word, with fractals. Fractal geometry is a potent language of complex visual form. It is wonderful in that it reduces much of the staggering complexity we see in nature to some very simple mathematics. I'm going to try to convey, as simply as I can, the intuition behind fractals in this chapter. I know it's a little confusing the first time around. It took me several rereadings of the standard texts when I was a graduate student to get it straight in my head. But after I "got it," it became clear that the important parts are very simple. I'm going to try to convey that simple view of fractals in this chapter. First a little motivation, though.

Building Mountains

One of the most common fractals we see in nature is the earth we walk on. Mountains are fractal, and we can make very convincing synthetic mountains with some pretty simple fractal computer programs. Benoit Mandelbrot, the inventor/discoverer of fractals and fractal geometry, calls such imitations "fractal forgeries" of nature. My own career in fractals began with making some new kinds of fractal

FIGURE 20.2 *Dale Stranded* illustrates the extreme depth, with pixel-level detail throughout, that can be imaged with the procedural methods MojoWorld employs. © 2002 Armands Auseklis.

terrains through novel computer programs, to create fractal forgeries the likes of which had not been seen before. Figure 20.2 shows the current state of the art in such "forgeries."

When I began to crank out some very realistic images, people immediately started asking me, "Why don't you animate them?" Well, there aren't many moving parts in a landscape. Landscapes tend to just sit there rather peacefully. What *is* free to roam about is the camera, or your point of view. This in turn begs the questions: Where do you come from? Where do you go? If the point of view is free to roam, the landscapes need to be in a proper global context. What's the global context for a landscape? A globe, of course!

Naturally, the next thing to do was to build a planet so that these realistic landscapes would have a geometrically correct context in which to reside. Fortunately, planets, being covered with terrains that are fractal, are themselves fractal. So we just need to build bigger fractal terrains to cover a planet.

FIGURE 20.3 A preliminary Earth-like planet model.

When I set out to build my first planet, seen here in Figure 20.3, it was apparent that we needed better fractal terrains. The kind of fractals we had all been so pleased with up to that time weren't really up to the job of modeling a whole planet—they were just too monotonous. So I created some new fractals, *multifractals,* that had a little more variety, as seen in Figure 20.4. Here's the difference: See how the coastline in Figure 20.3 has pretty much the same "wiggliness" everywhere on the planet? In Figure 20.4 you can see that the coastline is pretty smooth and uncomplicated in some places and pretty "rough" or "wiggly" in other places. That's pretty much all you need to know about multifractals—no kidding! They're another step toward the true richness and complexity we see in nature. The cool thing is that they don't complicate the math much at all, which is a very good thing, if you ask me.

There are other interesting aspects to modeling a planet. One that is not so obvious is the atmosphere. Landscape painters have known for hundreds of years that the atmosphere gives the only visual indication of truly large scale in a rendering. Leonardo wrote about it in his journal. Computer graphics have used atmospheric effects for as long as we've been making realistic renderings of fractal mountains. But it turns out that, in order to get the atmospherics to work really well, even on a local scale, you can't use the simplest and easiest atmospheric model—a flatland model. You have to use an atmosphere that curves around a planet. You've seen the sun lighting clouds from underneath well after it's set—you can't get that with a flatland model! So for practical reasons of getting things just so, you end up having

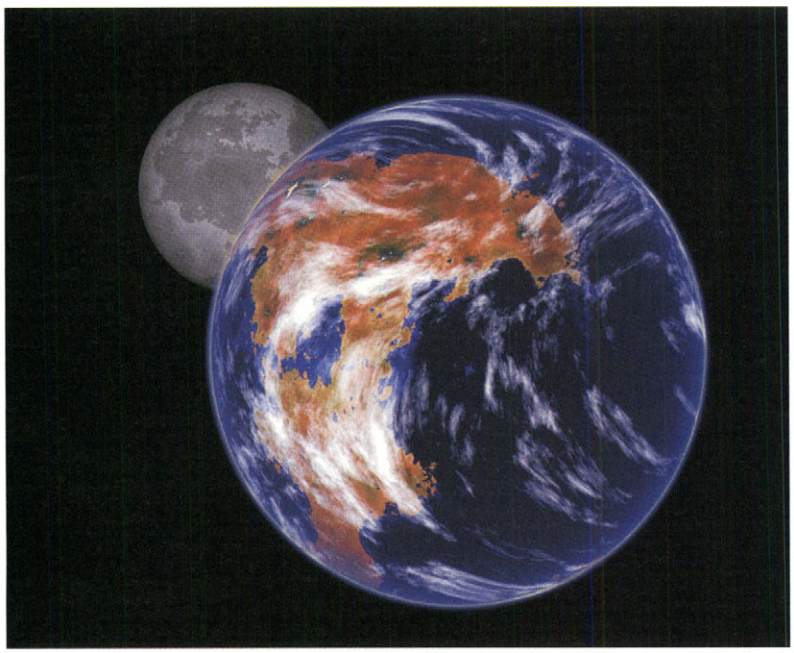

FIGURE 20.4 *Gaea & Selene* shows two procedural planet models. The clouds are the same ones seen in Figure 20.7. Note the multifractal coastline: at some places it is quite smooth, while at others it is quite convoluted, with many islands. Copyright © F. Kenton Musgrave.

to model Earth's atmosphere quite accurately, just to get sunsets to look right. Look up the word "atmosphere"—it literally means "sphere of vapor." Another indication of "global context."

Is the atmosphere fractal? Not in any obvious way, even though clouds certainly are. When I was a graduate student working under Mandelbrot, who was basically paying me to invent new fractal terrain models and make beautiful images of them, I worried that I was spending too much time on my atmosphere models. When I asked him about it, he, in true form, quipped mysteriously, "Well, many things that do not appear to be fractal are, in fact, fractal." It turns out that global circulation and the distributions of pollutants and density layers are fractal. Someday we'll have the power to model these things in MojoWorld, but not yet in this, the second year of the third millennium AD. Also, the path of photons as they are scattered by the atmosphere is fractal, not that it's of any consequence to us in making our pictures. Fractals are, indeed, everywhere.

Building a Virtual Universe

If landscapes need a global context, so do planets. Planets orbit suns in solar systems, stars tend to form in clusters, which in turn reside in and around galaxies, which gather in clusters and superclusters, right up the largest-scale features of our universe, which are in turn attributable to quantum fluctuations in the early universe, according to current cosmological theory. (If you want an explanation of *that,* you'd better ask Jim Bardeen, who wrote the part of MojoWorld that gives us continents with rivers and lakes. Jim is an astrophysicist who helped Stephen Hawking work out the original theory of black holes and now works on exactly that aspect of cosmology. He does rivers for us on the side, in his spare time.) Fortunately, the distribution of stars and galaxies, and the beautiful shapes of the stellar and interstellar nebulae that we're constantly getting ever better pictures of, are all quite fractal. In coming years, we here at Pandromeda have every intention of generating an entire synthetic universe that lives inside your computer. The path is clear as for how to do it. It will just take time to develop it and a lot of computer power to make it so. I, for one, am anxious for that future to arrive already!

Okay, so there's the big picture. Now how are we going to build this universe? What does it take?

Fractals. Lots of fractals.

WHAT IS A FRACTAL?

Let's get to first things first: What exactly is a fractal? Let me offer this definition:

> *fractal:* a complex object, the complexity of which arises from the repetition of a given shape at a variety of scales

Here's another definition, from *www.dictionary.com:*

> *fractal* (noun): A geometric pattern that is repeated at ever smaller scales to produce irregular shapes and surfaces that cannot be represented by classical geometry. Fractals are used especially in computer modeling of irregular patterns and structures in nature. [French from Latin *fractus,* past participle of *frangere,* to break; *see* fraction.]

It's really that simple. One of the easiest examples of a fractal is the von Koch snowflake (Figure 20.5). In the von Koch snowflake the repeated shape is an

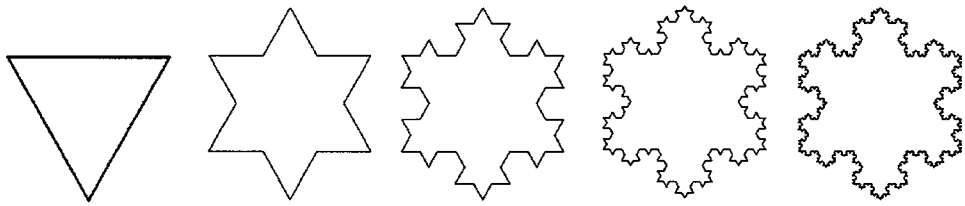

FIGURE 20.5 The von Koch snowflake: a canonical fractal.

equilateral triangle. Each time it is repeated on a smaller scale, it is reduced in size by exactly one-third. This repetition can continue at ever smaller scales ad infinitum, leading to a curve—the edge of the snowflake—that is wonderfully complex. It also exhibits some bizarre mathematical properties, but we won't go into those here.

There's lots of math we could go into about fractals, but perhaps the neatest thing about fractal geometry is that you don't need to learn any math at all to understand and use it. You can think of it as an artist, entirely in terms of visual form. Let me describe a few easy ways to think of fractals.

Self-Similarity

The repetition of form over a variety of scales is called *self-similarity*: a fractal looks similar to itself on a variety of scales. A little piece of a mountain looks a lot like a bigger piece of a mountain and vice versa. The bigger eddies in a turbulent flow look much the same as the smaller ones and vice versa (see Figure 20.6). Small rocks look the same as big rocks. In fact, in geology textbooks, you'll always see a rock hammer or a ruler in photographs of rock formations, something to give you a sense of scale in the picture. Why? Because rock formations are fractal: they have no inherent scale; you simply cannot tell how big a rock formation is unless you're told. Hence another synonym for the adjective "fractal" is "scaling": a fractal is an object that is invariant under change of scale.

Figure 20.6 shows my favorite example of a fractal in nature. Looks kind of like a puff of smoke from a smoker's mouth at arm's length, or a drop of milk in water, doesn't it? Guess how big it is. It's about 400,000 light years wide—that's roughly four thousand trillion kilometers, or 24 hundred trillion miles from one end to the other. Too big for me to imagine! In the 1970s, when I first saw this picture, taken at radio wavelengths by the Very Large Array radio telescope, I considered it the most mind-blowing image I'd ever seen. I am not sure if Benoit had even coined the term

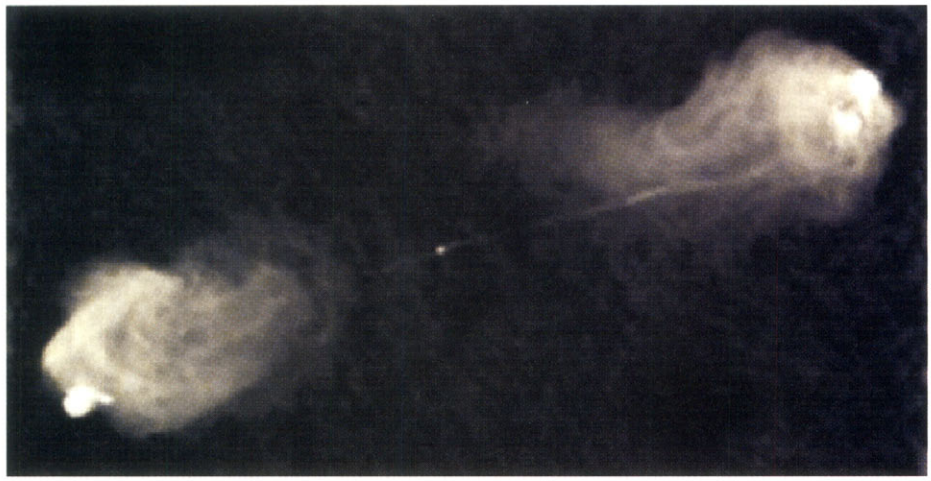

FIGURE 20.6 A fractal on a truly grand scale: jets of gas from an active galactic nucleus. VLA radio image of Cygnus A at 6 cm courtesy of NRAO.

"fractal" yet; if he had, I certainly hadn't heard it yet. But I found it stunning that this incomprehensibly large object looked so ordinary, even small. That's a fractal for you. You can recognize one immediately, even if you don't know it's called a fractal.

Dilation Symmetry

My favorite, easy way to grasp the idea of "fractal" is as a new form of symmetry: *dilation symmetry.* You're probably already familiar with symmetries such as the mirror symmetry by which the human body is pretty much the same on both sides when mirrored across a line down the middle, and perhaps the rotational symmetry whereby a square remains unchanged by a rotation of 90°. Dilation symmetry is when an object is unchanged by zooming in and out. Turbulence is like that; hence we can't tell how big that turbulent puff of gas in Figure 20.6 is until we're told.

Imagine, if you will for a moment, a tree branch. A smaller branch looks pretty much like a larger branch, right down to the level of the twigs. This is dilation symmetry. The same goes for river networks: smaller tributaries and their networks look much like the larger river networks. Figure 20.7 shows this in some of Jim Bardeen's

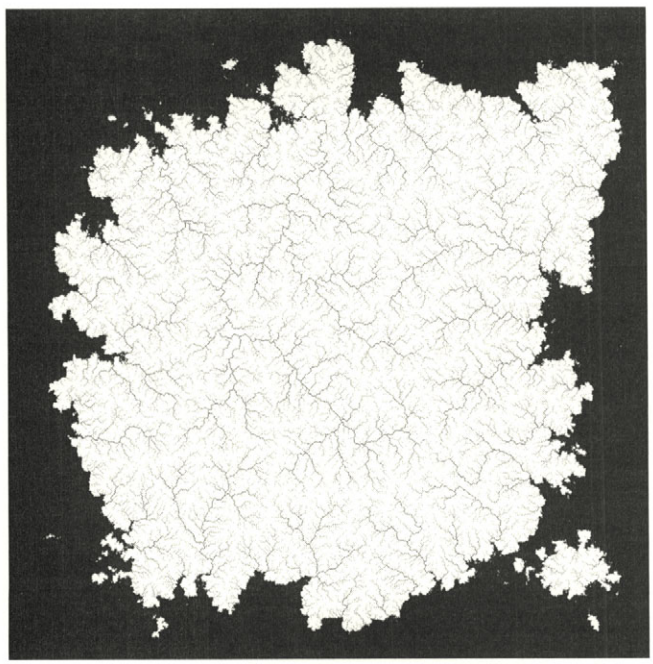

FIGURE 20.7 A fractal river drainage network for a MojoWorld continent.

river networks on a MojoWorld continent. Clouds, mountains, coastlines, and lightning are like that, too; smaller parts look just like larger parts. There is a catch: unlike the Koch snowflake, they aren't *exactly* the same at different scales, only qualitatively so. This leads to our next distinction in fractals: *random fractals*.

Random Fractals

Random fractals may be loosely defined as *fractals that incorporate random variables in their construction.* The random variable may be some quantum process, like the probability of a given air molecule scattering a passing photon, or a pseudo-random variable in a computer program, as we might use to determine the altitude of a point on a fractal terrain. Computers are always deterministic, so we don't have truly random variables in computer programs, only ones that are designed to look random while being, in fact, deterministic. "Deterministic" means that a given input always generates the same output. This determinism is a good thing: it is why we always get the same MojoWorld from a given scene file, even though what we find

there is unpredictable. If the computer were producing truly random variables, we might get slightly better MojoWorlds (for very obscure mathematical reasons), but we wouldn't be able to roam around and come back to the same place again.

The point is that self-similarity comes in at least two flavors: *exact* self-similarity, as in the Koch snowflake where every part is exactly the same as every other if you rotate it properly, and *statistical* self-similarity, as in all the natural phenomena I've mentioned. In nature, self-similarity is usually of the statistical sort, where the statistics of random behaviors don't change with scale. But you needn't worry any about statistics—to the human eye these fractals look similar at different scales, no doubt about it, without any reference to numbers, statistics, or any other fancy mathematics.

A BIT OF HISTORY OF FRACTAL TERRAINS

Like all intellectual revolutions, fractal geometry did not happen overnight. Rather, it was an extension of the work and observations of many people working in many different fields. But let me perform the standard practice of historians and make a complex story simple.

The Mathematics

Fractals were noticed by mathematicians around the turn of the 20th century. They noted their mathematically bizarre behavior, labeled them "monsters," and left them for posterity.

Benoit Mandelbrot had an uncle who was a famous mathematician. He assured the young Benoit that the person who cracked this mathematical case could make a name for himself. Benoit was not immediately interested, but the ideas festered (my word, not his!) in his mind, and he eventually came to work on such things as a researcher at IBM. In 1982 he published his classic book *The Fractal Geometry of Nature,* which introduced the world to fractals. In 1987 I was fortunate to be hired as Benoit's programmer in the Yale math department. I was to work on fractal terrain models that included river networks, a research project that didn't pan out for us. In 1988 *The Science of Fractal Images* was published, edited by Heinz-Otto Peitgen and Dietmar Saupe, whom I had gotten to know at UC Santa Cruz in 1986. In it Mandelbrot issued a challenge to the world to solve the difficult problem of creating fractal terrains with river networks. In 1989 Craig Kolb, Rob Mace, and I published a paper titled "The Synthesis and Rendering of Eroded Fractal Terrains," which described a way to create rivers in fractal terrains, although that method

remains mathematically and computationally intractable to this day. A little later Jim Bardeen solved the problem in the way that Benoit had in mind; his latest solution appears in MojoWorld. In 1993 I completed my doctoral dissertation "Methods for Realistic Landscape Imaging," which was pretty much a compilation of the various papers and course notes that I had published over the last six years on various aspects of modeling and rendering realistic forgeries of nature. In it I described my multifractal terrain models.

That's a very brief sketch of the academic mathematical history of fractal terrains. Now for the mathematical imaging track.

Mathematical Imaging of Fractal Terrains

Mandelbrot divides the history of computer images of fractal terrains into three eras: the Heroic, the Classical, and the Romantic. The Heroic Era is characterized by the work of Sig Handelman, who made the first wire frame renderings of Benoit's terrain models. According to Benoit, it was a heroic effort in the 1970s to get even a wire frame image out of the computer, hence the name of that era. Handelman's images are mostly lost in the mists of time, alas, although one or two appear in *The Fractal Geometry of Nature* (Mandelbrot 1982). Next came the work of Richard Voss, who made the first realistic (for that time, at least) images, such as the classic *Fractal Planetrise,* which graces the back cover of *The Fractal Geometry of Nature.* Voss's work comprises the Classical Era. Richard fleshed out the mathematics and rendering algorithms required to make beautiful and convincing forgeries of nature. Next came my work, in which I brought various artistic subtleties to the forefront. As I went on at length about artistic self-expression, Benoit calls my work the Romantic Era. Benoit has generously credited me with being "the first true fractal-based artist." Figures 20.8–20.10 are a few of my personal favorites from my own body of work of that era. You may notice that I was fascinated with planets right from the start.

The Computer Graphics Research Community

Then there's the computer graphics track. In 1979 Loren Carpenter, then at Boeing, made the groundbreaking computer animation *Vol Libre,* the first animated flyby of a synthetic fractal terrain. In 1982 Loren, now senior scientist at Pixar and a member of Pandromeda's distinguished board of advisors, published with Alain Fournier and Donald Fussell the paper "Computer Rendering of Stochastic Models," which introduced the commonly used polygon subdivision method for generating fractal

FIGURE 20.8 *Blessed State* is also a polygon subdivision terrain—compare it to the terrain models seen in Figures 15.7 and 20.9. The water is the "ripples" texture. The moon is a very simple fBm bump map applied to a white sphere; a more realistic moon model would be too busy for the visual composition. Copyright © F. Kenton Musgrave.

FIGURE 20.9 *Zabriskie Point* illustrates distorted fBm clouds and the rounded character of terrains constructed from the Perlin *noise* function. A vector-valued fBm function, with the vector interpreted as an RGB color value, has been used to perturb the color of the terrain. Such coloring is also seen in Figure 17.11. It is a prototype of the more sophisticated coloring seen in Figures 15.15, 16.6, 20.18, and 20.20. Copyright © F. Kenton Musgrave.

FIGURE 20.10 *Pleiades* has all the major elements of a procedural fractal universe. Copyright © 1996 F. Kenton Musgrave.

terrains. That precipitated a bit of a feud with Mandelbrot, but it was before my time in the field, so I'll say no more about that. Also in 1982, Loren and the rest of the distinguished crew at Pixar showed us the first MojoWorld (if I may be so presumptuous) on the big screen in the "Genesis Sequence" in *Star Trek II: The Wrath of Kahn*. That blew our minds. In 1985 the late Alain Fournier came to UC Santa Cruz to teach a course, "Modeling Natural Phenomena," that changed my life and set the course of my career.

Alain mentored me in the area and advised me in my master's research. Later in 1985 Ken Perlin and Darwyn Peachey published twin papers that introduced the procedural methods that have pretty much driven my entire career. Thanks, guys! They had some really cool pictures in those papers; I saw a world of possibility (so to speak) in their methods, and the rest is, well, history. In 1986 Dietmar Saupe and Heinz-Otto Peitgen came to the UC Santa Cruz math department, and I took Dietmar's course "Fractals in Computer Graphics." In 1987 Dietmar recommended me to Mandelbrot for the job at Yale. (I always tell my students: "There's no substitute for dumb luck.") In 1993 I finally graduated from Yale with a terminal degree—

a Ph.D., but I prefer that other term for it—at the ripe old age of 37. Can you say "professional student?" In 1994 Matt Pharr, Rob Cook, and I created the *Gaea Zoom* computer animation (see Figure 16.3 and *www.kenmusgrave.com /animations.html*). I believe it was the first MojoWorld with adaptive level of detail, that is, a synthetic fractal planet that you could zoom in and out from, without nasty artifacts described by obscure mathematics that I won't go into here. Suffice it to say, it's not so easy to make a planet that looks good from both near and far, doesn't overload your computer's memory, and renders in a reasonable amount of time. The *Gaea Zoom* took two weeks to render on four supercomputers, so we weren't quite there yet in 1994. Finally, in 2001, we were. Hence MojoWorld launched that year—it really couldn't have been launched even a year earlier.

The Literature

And then there's the track of explanations of fractal terrains. First came the technical papers that even *I* never could really understand. Then came *The Science of Fractal Images* in 1988 (which I even wrote a tiny part of), in which Richard Voss and Dietmar Saupe cover everything you'd ever want to know about the mathematics of these and other kinds of fractals. Next came our book, *Texturing & Modeling: A Procedural Approach,* first edition in 1994, second in 1998, third edition circa 2003 in your hands now, by David Ebert, Darwyn Peachey, Ken Perlin, Steve Worley, and me. In it I explain how to build fractal terrains from a programming perspective. Still pretty technical, but the standard reference on how to program fractal terrains and even entire MojoWorlds. And now there's this little exposition, in which I'm trying to explain it all to the nontechnical reader.

The Software

Last but not least, there's the software track of the history of fractal terrains. For some time there existed only the various experimental programs created by us academics. They couldn't be used by the average person or on the average computer. The first commercial software that included fractal terrains was high-end stuff like Alias. I'm afraid I must plead ignorance of those high-end packages. I was just a lowly graduate student at the time and, while I had access to some mighty fancy computers to run my own programs on, we certainly couldn't afford such top-of-the-line commercial software, and they weren't giving it away to university types like us despite our constant and pathetic pleas of poverty and need.

The first affordable commercial fractal terrain program that came to my attention was Vistapro, around 1992. With its flight simulator interface for making animations, it was really cool. You can still buy Vistapro, although it's a bit quaint by today's technological standards. Next, I believe, came Bryce 1.0, in 1994, written primarily by Eric Wenger and Kai Krause for what was then HSC Software, then MetaTools, then MetaCreations, now defunct. (Corel now owns the Bryce name and is carrying the product forward.) Bryce 1.0 was Macintosh-only software and mighty cool. It put a user-friendly interface on all the procedural methods that my colleagues and I had been going on about in the academic literature for years, and made it all accessible to the average home computer user. Since then, Animatek's World Builder and 3D Nature's World Construction Set have released powerful, semi-high-end products priced around $1000. Natural Graphics' Natural Scene Designer, E-on Software's Vue d'Esprit, and Matt Fairclough's Terragen shareware program have filled out the low end at $200 and less. Each product has its specialty; each can create and render fractal terrains. Meanwhile, Bryce is up to version 5.0. I personally worked on Bryce 4.0 for MetaCreations until December 1999, when MetaCreations imploded. The very next day FractalWorlds, now Pandromeda, was launched to make MojoWorld a reality. We're the first to come to market with entire planets with level of detail, thus opening the door to cyberspace.

Disclaimers and Apologies

So there's Doc Mojo's *Close Cover Before Striking History of Fractal Terrains.* Sure, it's biased. I worked for Mandelbrot. I come from the academic side and was camped more with the mathematicians than with my real colleagues, the computer science/computer graphics people. I've left out a lot of important contributions by friends and colleagues like Gavin Miller, Jim Kajiya, and many, many more. I'm keeping it brief here; if you want the exhaustive listing of who's done what in the field, see the bibliography of this book or my dissertation. And I must state that, although many people think so, I am *not* a mathematician. Believe me, having a faculty office in the Yale math department for six years drove that point home! My degrees are in computer science. But I'm really a computer artist more than a computer scientist. My contribution has been mostly to the artistic methods, ways to make our images of fractal terrains more beautiful and realistic. I'm a mathematical lightweight; I just do what I have to do to get what I want. And all my reasoning is visual: I think in terms of shape and proportion, even if I do translate it into math in order to make my pictures. To me all the equations just provide shapes and ways to combine them.

The Present and Future

The abstract of my 1993 doctoral dissertation ends with this sentence:

> Procedural textures are developed as models of mountains and clouds, culminating in a procedural model of an Earth-like planet that in the future may be explored interactively in a virtual reality setting.

The planet model I was referring to is Gaea, seen in Figures 20.4 and 16.3, implemented as a C language version of the "terran" texture presented in Chapter 15. In MojoWorld we finally have the interactive planet I envisioned and an endless variety of others as well. It's still not quite what I'd call "virtual reality"—the real-time part is just not that realistic. Yet. But getting there is just a matter of a lot of hardware and software engineering.

Gertrude Stein once said of Oakland, California, "There's no 'there' there." If that's true for Oakland, I say it's even more true of all implementations of virtual reality up to now. MojoWorld, at last, puts the "there" there. This is the main thing that's fundamentally new about MojoWorld.

All other 3D graphics programs build the equivalent of stage sets. Stage sets are designed to be viewed from a distance and at a given resolution, whether it's that of a TV camera, a movie camera, or the human eye. If you get too close, the flaws show. (One problem TV studios face today is that the transition to high-definition, or HD, video requires that they replace their studio sets. The ones that look fine with older video technology won't cut it with HD video: all the defects in the sets show—things like dents and dings, coffee rings, and poor craftsmanship that are invisible at NTSC resolution suddenly are clearly visible. I discovered the same thing early in my career as a computer artist: images that looked great at screen resolution didn't always look so good when rendered at poster resolution.) Also, a set is always *local:* it is of finite size, and if you get too far away, it ends. And generally, if you view it from the wrong angle, the effect it's designed to create breaks down. MojoWorld isn't like that. You can go anywhere. In MojoWorld, as Buckaroo Bonzai said, "Everywhere you go, there you are." You never run out of detail, and there's always someplace new and interesting to visit.

I think of MojoWorld as a window on a parallel universe, a universe that already exists, always has and always will exist, in the timeless truth of mathematical logic. All we've done is create the machinery that reveals it and the beauty to be found there. It's been my great privilege to play a small part in the discovery/creation

FIGURE 20.11 *Southern Parfait Flood* is another scene from the parallel universe we call the "Mojoverse." There are countless other such scenes of beauty and fascination waiting to be discovered there. Copyright © 2002 Armands Auseklis.

of this possibility. It's been my vision for years now to make this experience accessible to everyone. With MojoWorld, we're on our way, and I, for one, am very excited about that. I hope you enjoy it half as much as I will!

Now let's get on to explaining how we build a MojoWorld so that you understand the controls that you'll be using.

BUILDING RANDOM FRACTALS

The construction of fractal terrains is remarkably simple: it is an iterative loop involving only four important factors, one of which is generally a nonissue. First, we have the *basis function,* or the shape that we build the fractal out of, by repeating it at a variety of scales. Next there's the *fractal dimension,* which controls the roughness of the fractal by simply modulating the amplitude or vertical size of the basis

function in each iteration (i.e., each time you go through the loop). Then there are the *octaves,* or the number of times that we iterate in building the fractal. Finally, we have the *lacunarity,* or the factor by which we change the frequency or horizontal size of the basis function in each iteration. Usually, we leave the lacunarity at around two and never think about it again. Let's see what the effect of each of these four factors is and how they all fit together.

The Basis Function

The basis function is perhaps the most interesting choice you get to make when building a random fractal, whether for terrain, clouds, water, nebulae, or surface textures. The shape of the basis function largely determines the visual qualities of the resulting fractal, so "choose wisely." It's fun to experiment and see the subtle and not-so-subtle visual effects of your choice of basis function. I have certainly gotten a lot of artistic mileage over the years through careful choice and modulation of basis functions. And MojoWorld has plenty of basis functions, that's for sure.

For obscure but important mathematical reasons, basis functions should (1) have shapes that are not too complicated, (2) never return values smaller than -1.0 or larger than $+1.0$, and (3) have an average value of 0.0. As Mick once said, "You can't always get what you want," but the basis functions in MojoWorld are designed to obey these constraints in most cases.

The main thing to keep in mind about the basis function is that its shape will show through clearly in the fractal you're building. So your choice of basis function is the most significant decision you make when building a fractal.

Fractal Dimension: "Roughness"

Fractal dimension is a powerful, if slippery, beast. I'll leave mathematical explanations to other texts. In MojoWorld we call it "Roughness." For our purposes, just think of the Roughness control as a slider that controls visual complexity. It does this by varying the jaggedness of terrain, the wiggliness of coastlines, the raggedness of clouds, and the busyness of textures. The Roughness control in MojoWorld is an extremely potent control.[1] Use it a lot! You'll find it a powerful and subtle way to

1. The way we've implemented it in MojoWorld, the Roughness control doesn't necessarily have an accurate relationship to the numerical value of the fractal dimension in the fractal you're building, but that doesn't matter—the numerical value isn't important to us in the context of MojoWorld, only the qualitative effect we're getting, which you can assess visually.

affect the aesthetics of your fractals. And don't be surprised if you find yourself setting its value via text entry, down to the third digit after the decimal point. It's that sensitive and powerful. Play with it and see.

Larger values make for rougher, busier, more detailed fractals (see Figure 20.12). They tend to get visually "noisy" at values over about 0.5. I generally prefer to use smaller values than most people, but, hey, it's strictly a matter of taste.

Octaves: Limits to Detail

We call the number of times we iterate, adding in more detail, the *octaves*. Fractals can have potentially unlimited detail. But that detail has to be built by the computer, so it must have limits if you want your computation to finish. In nature, fractals are always *band-limited:* there is a scale above which the fractal behavior vanishes (this is even true of the largest structures in the universe) and a scale below which it also goes away (as when we get to the scale of quantum physics). For mathematical reasons, MojoWorld has to be in control of the number of times each sample of a fractal goes through the construction loop. The explanation for why this is so is beyond the scope of this presentation. (See Chapter 17 for details on this.) Suffice it to say that controlling the number of times we go through the loop controls the amount of detail, and the amount of detail required at a given point in the image depends on its distance from the camera, the screen resolution, the field of view, and other, more subtle factors as well. Furthermore, too much detail not only wastes computation time, it also causes *aliasing*—nasty visual artifacts that we go to great lengths to eliminate in MojoWorld. So, bottom line, MojoWorld has to control the number of octaves in the fractals. That's just the way it is.

We can, however, play games with the octaves. The Detail control can reduce or increase the number of octaves, and hence the fine detail, in MojoWorld fractals. Its effects can look pretty strange in animations and when you change the rendering resolution, but it can keep your fractals from being annoyingly visually "busy"

FIGURE 20.12 Planets with fractal roughness of −0.5, 0.0, 0.5, 1.0, and 1.5.

everywhere, all the time. The Largest Feature Size and Smallest Feature Size controls set the band limits to the fractal. You'll get no more fractal detail above and below these scales. You have to set the Largest Feature Size to something reasonable. Keep in mind that it shouldn't be any larger than the planet, or you're just wasting computation time. The Smallest Feature Size can be left at zero. MojoWorld will eventually decide that "enough is enough" and stop generating more detail, but you'll probably get tired of zooming in long before that.

Tip: *If you build a terrain or texture that aliases in the high-quality MojoWorld renderings, use the Detail control to reduce the number of octaves until the aliasing goes away at the quality level you're using.*

Note: *The MojoWorld RTR (real-time renderer) uses a very different sampling method than the MojoWorld photorealistic renderer. It will usually suffer significant to severe aliasing when viewing a planet from a distance, whereas the photorealistic renderer will not.*

Lacunarity: The Gap between Successive Frequencies

This one is usually a nonissue, but we've made it an input parameter in MojoWorld just to be thorough, as you can get certain unique artistic effects using lacunarity. When going through the iterative loop that builds the fractal, the frequency or lateral scale of features must change at each iteration because that's how we get features at a variety of scales. The *lacunarity* determines how much the scale is changed at each iteration. Since "scale" is in this case synonymous with "spatial frequency" (of the features in the basis function), it's easiest to think of the lacunarity as the gap between successive spatial frequencies in the construction of the fractal. Indeed, "lacuna" is Latin for "gap."

Usually, we double the frequency at each iteration, corresponding to a lacunarity of 2.0. Musically, this corresponds to raising the frequency by one octave, hence the term "octaves" for the number of scales at which we create detail. Why the value of 2.0 and not something else? Well, it has to be bigger than 1.0, or you go nowhere or even backward. On the other hand, the bigger you make it, the faster you can cover a given range of scales because you're taking a bigger step at each iteration. Each iteration takes time, and when you're building a planet, you have a big range of scales to cover. So a clever person might think, "Well, then, just crank up the lacunarity!" Not so fast, Bucko. It turns out that for lacunarity values much over 2.0, you start to see the individual frequencies of the basis function. It just looks bad, as Figure 20.13

FIGURE 20.13 Planets made with a Perlin basis and lacunarities of 2.7, 5.0, and 10.0.

FIGURE 20.14 Planets made with a Perlin basis and lacunarities of 1.5, 1.9, and 2.7.

demonstrates. Similarly, Figure 20.14 shows that lacunarities less than 2.5 pretty much look alike, although close inspection will reveal artifacts at 2.7. We've gone with a default lacunarity just over 2.2 in MojoWorld, to eek out a little more speed. If you want images that are as good as they can be, I'd recommend a value more like 1.9.[2] My best advice: Don't mess with lacunarity until you really know what you're doing.

2. For obscure technical reasons, it's best not to use a lacunarity of exactly 2.0, but something close to it, like 1.9 or 2.1. Transcendental numbers are best. MojoWorld's default is the natural logarithm *e* minus one-half, or 2.718 . . . −0.5 = 2.218. . . . You might try changing the 2 after the decimal point to 1. Keep in mind you should set your lacunarity permanently before you use your fractal because changing it will change all the features except those on the largest scale, and this could completely disrupt some specific feature in your MojoWorld that you've become interested in.

ADVANCED TOPICS

That covers the fundamentals of where MojoWorld comes from and how it works. Now we'll go into some of the more esoteric aspects of MojoWorld.

Dimensions: Domain and Range

The various functions used to create textures and geometry (for example, mountains) in MojoWorld are implemented in several dimensions. What does this mean? Pay close attention, as this can be a little confusing and even counterintuitive. A *function* is an entity that maps some input, called the *domain* of the function, to some arbitrary output, called the *range*. Inside the MojoWorld program, the domain and range are all a bunch of numbers. As users, we usually think of the output as color, the height of mountains, the density of the atmosphere, and other such things. We also think of the input as things like position in space, altitude, and color, too, or as numbers like the ones we can type into the UI.

Both the domain and range of functions have *dimensions*. One dimension corresponds to a line, two to a plane, three to space, and four to space plus time. When you're creating a new function in MojoWorld, you'll sometimes have to choose the dimensionality of the domain of the function. This seems a little backward, as what you're really interested in is the dimensionality of the output, or the range of the function. Here's the catch: you can't have meaningful output of dimensionality greater than that of the input. There's just no way to make up the difference in information content.

Usually, we're working in three dimensions in MojoWorld, so that's the correct default choice to make when you're confronted with this decision. But, in general, every added dimension doubles the time required to compute the function. So you want to use as few dimensions as you can get away with. You might also want to do some special effects using lower dimensions, like determining the climate zones of your planet (implemented as a three-dimensional texture) by latitude (a one-dimensional variable). Figure 20.15 illustrates MojoWorlds made from the same function, with domains of one, two, and three dimensions.

MojoWorld also has a full complement of functions with four-dimensional domains "under the hood." These will be useful for animating three-dimensional models over time to simulate things like continental drift and billowing volumetric clouds. The user interface for animation is a complicated thing, however, so we decided to leave it for a future version of MojoWorld, when we've had time to do it right.

FIGURE 20.15 Planet made from a sine function with one-, two-, and three-dimensional domains.

Hyperspace

You may have noticed that we go on about *hyperspace* a lot in our MojoWorld propaganda. Hyperspace is whenever you go up one dimension: a plane is a hyperspace to a line, and three dimensions is a hyperspace to a plane. Add time to space, and you have a hyperspace to the three dimensions we're most familiar with. Well, it's really easy to keep adding more dimensions. Take the three dimensions of space and add a color, for example. Because the human eye has three different color receptors in the retina, one each for red, green, and blue light, human vision has three color dimensions—hence the RGB color space used in computer graphics. (Some birds have six; they live in a world with far richer color than we monkeys.) It takes three values to specify a color for the human eye: one each for red, green, and blue. Each is independent—part of the definition of a *dimension*. From *www.dictionary.com:*

> **dimension:** . . . 4. *Mathematics.* a. One of the least number of independent coordinates required to specify a point in space or in space and time.

In our example we have three dimensions for space and three for color, for a total of six dimensions. Presto—a hyperspace! Not hard to do at all, eh?

Now think for a minute of all the values you may assign to make a MojoWorld—things like planet radius, sea level, color, atmospheric density, fractal dimension, and so on. For an interesting planet, there'll be hundreds, even thousands of variables involved. From a mathematical standpoint, each independent variable adds another dimension to the space that the planet resides in. The more complicated the MojoWorld, the higher the dimensionality of the space it resides in. Each

lower-dimensional space, as for the same MojoWorld with one less color specified, is called a *subspace* in mathematics. And, of course, each higher-dimensional space is a *hyperspace*. There's no limit to the amount of complication you can add to a MojoWorld, and so there's no limit to the dimensionality of the master MojoWorld hyperspace. Pretty mind-bending, ain't it? I certainly think so!

The variables used to specify a MojoWorld are called *parameters*. So the master hyperspace spanned by the possible parameters is rightly called *Parametric Hyperspace*. (We took out a trademark on the name because that's what corporations do.) It's simply the most succinct and accurate way to think and talk about how MojoWorld works. A pure MojoWorld scene file, uncomplicated by content such as plants, cities, monkeys, and the like, needs only encode the numbers that specify the parameter settings. Everything else is generated at run time from these values. This set of parameter values specifies the point in Parametric Hyperspace where the MojoWorld resides. Load them into MojoWorld, and MojoWorld "beams" you there—hence we call them *transporter coordinates*. Very sci-fi, but very scientifically and mathematically accurate. MojoWorld Transporter let's you beam yourself to these places and explore them in three dimensions. If you want access to all of

FIGURE 20.16 *At the Edge* illustrates the kind of beautiful surface textures that can be built using the procedural texture engine that is the heart of MojoWorld. Copyright © 2002 Allison Morris.

Parametric Hyperspace, however, and have the ability to mess with all of the parameters, you need MojoWorld Generator.

The Basis Functions

MojoWorld has by far the richest set of basis functions ever seen in a random fractal engine, so don't be surprised to find the choices a bit bewildering at first. They're all based on methods from the academic literature in computer graphics. If you're an advanced graduate student in the field, they should all be familiar. If you're not, don't worry—not everyone needs a Ph.D. in computer graphics! (Very few do, indeed.) You can familiarize yourself with the choices by least two routes: you can plunge right into the lists of basis functions and their controlling parameters in the MojoWorld Texture Editor, or, if you're less patient and intrepid (like myself), you can keep examining the way various MojoWorlds that you like have been built and note the basis functions used in fractals that you particularly like. The variety of basis functions available in MojoWorld far surpasses anything I, personally, have ever had before, and it will be a long time—probably *never*—before I get around to trying every basis function that's possible in there. In fact, once you get to distorting or applying curves to the basis functions, the possibilities are basically infinite, so no one will ever try them all.

Because there are so many, I'm not going to try to describe all of MojoWorld's basis functions here. Rather, I'll describe the basic classes into which they fall and the fundamental visual qualities of each.

Perlin

The best and fastest basis function, in general, is a Perlin *noise*. Ken Perlin introduced this famous basis function in his classic 1985 SIGGRAPH paper "An Image Synthesizer," and it's the basis function that launched a thousand pictures (and my own career). Ken—and everybody else—calls it a "noise function." In the context of fractal mathematics this term is rather misleading, so in MojoWorld we call Perlin *noise* a "Perlin basis." At any rate, there are several flavors of the Perlin basis, and we have them all in MojoWorld, plus a new one.

The Perlin basis consists of nice, smooth, random lumps of a very limited range of sizes. Its output values range between −1.0 and 1.0. It's ideal for building smooth, rounded fractals, such as ancient, heavily eroded terrains as seen in Figures 16.2 and 18.2. One of everyone's favorite terrain models is the so-called "ridged multifractal" seen in Figure 20.18. It looks completely unlike an ordinary Perlin fractal but is

FIGURE 20.17 A planet with relief from a Perlin basis.

closely related: it's made by taking the absolute value of the Perlin basis—that is, by changing the sign of all negative values to positive—and turning that upside down. Figure 20.19 shows visually how this process works. The result is a basis that has sharp ridges. You can use it in a monofractal function to get terrain, as seen in Figure 20.20, or in a multifractal to get terrain, as in Figure 20.18.

Voronoi

The Voronoi, or "cellular," basis functions are cool and useful, but slow. Steve Worley introduced the Voronoi basis in his 1996 SIGGRAPH paper (Worley 1996). It has a cell-like character, kind of like mud cracks with pits drilled into the middle of each tile of the mud (see Figure 20.21.) The pits are conical when you use the "distance" contour and rounded (actually, parabolic) when you use the "distance squared" contour. The mud tiles are flat plateaus of random height when you use the "cell ID" contour. The value of the Voronoi basis at a given sample point derives from the distance of that sample point from a random point in space, called the "seed," and one or more of its neighbors in a stored set of seeds. There is a ridge at the perpendicular bisector of the line between the random point and the chosen neighbor. That neighbor can be the first, second, third, or fourth closest neighbor to the point. You don't have to worry about which number you choose; rather, just look at the quality of the resulting texture and choose one you like. You can also

FIGURE 20.18 *Emil* is a terrain made from a ridged multifractal, a variety of Perlin fractal.

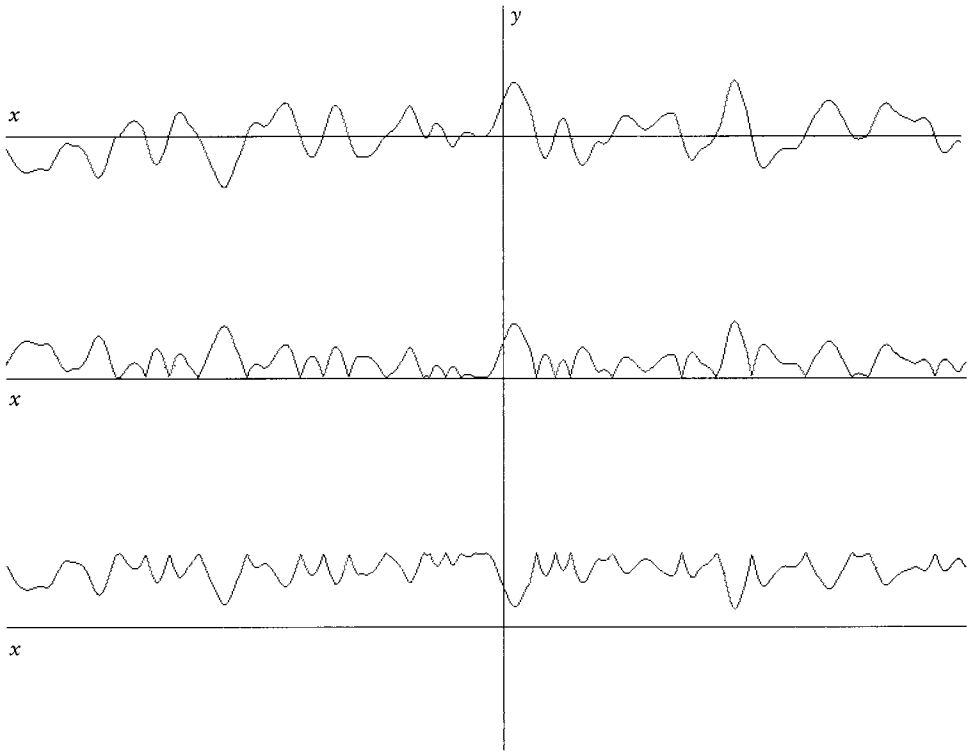

FIGURE 20.19 Building a ridged basis function from a Perlin basis: the Perlin basis, its absolute value, and one minus the absolute value.

choose the differences between the first and second, second and third, or third and fourth neighbors. Again, figure out what that means only if you want to; otherwise, just examine the quality of the texture you get and choose one you like for aesthetic reasons.

Voronoi basis functions have ridges like the ridged Perlin basis, only they're all straight lines. Usually, you'll want to apply a fractal domain distortion to Voronoi fractals, to make those straight lines more natural (read: wiggly).

Sparse Convolution

In 1989 John Lewis introduced the sparse convolution basis: the slowest, technically "best," and most flexible of all (Lewis 1989). I'd say its time has not yet come—we simply need faster computers before this basis is going to be practical. But I'm

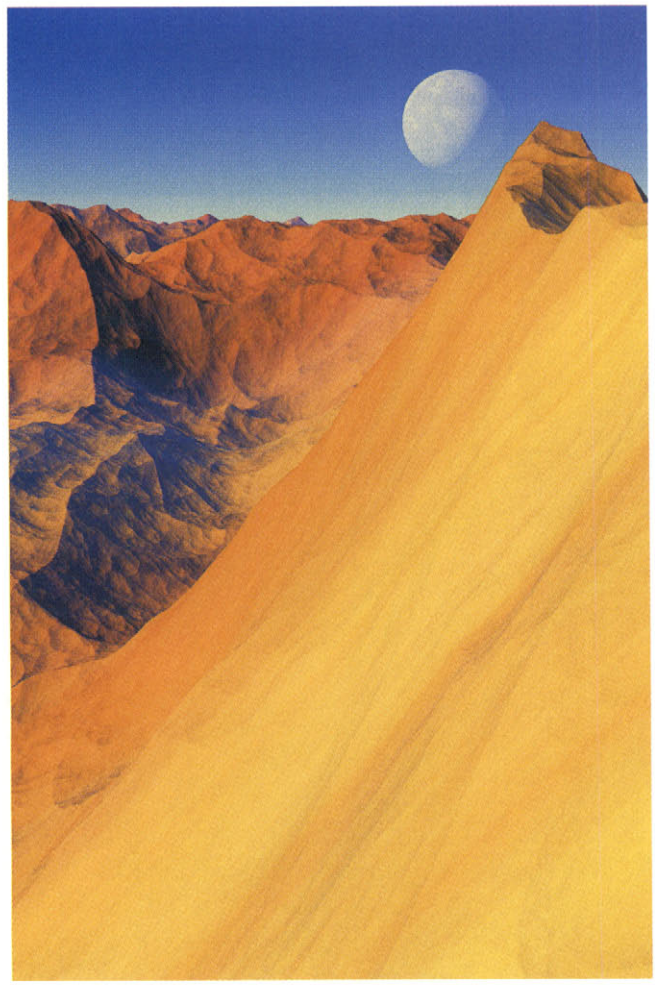

FIGURE 20.20 *Slickrock* is a monofractal built from a ridged Perlin basis. Copyright © 1993 F. Kenton Musgrave.

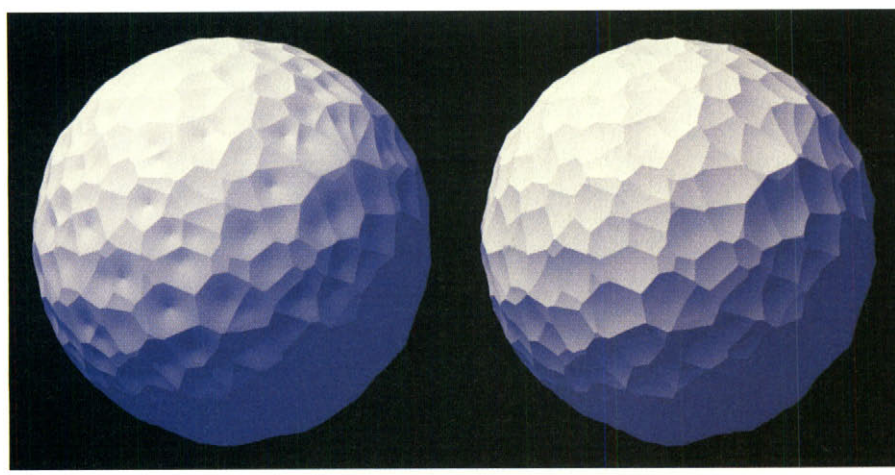

FIGURE 20.21 Planets made from the Voronoi basis, first neighbor. "Distance" Voronoi on the left, "distance squared" on the right. "Distance" has conical pits; "distance squared" has smooth pits.

personally obsessed with basis functions and I wanted it in there, so I put it in there. And to my surprise, people are using it! (See Figure 20.22.)

The sparse convolution basis generates random points in space and splats down a *convolution kernel* around those points. (That funny name comes from some more pointy-headed math terminology.) The kernel can be literally anything. In MojoWorld version 1.0 it can be a simple, radially symmetric shape that you choose from a list of options or building the curve editor. In later versions, we'll get into some bizarre and powerful kernels like bitmaps. Personally, I look forward to building planets out of Dobbsheads.

Various

Then there are the various other basis functions that I've thrown in for fun. See if you can find a creative use for them! Some of you may wonder why I haven't included some of the ones found in other texture engines such as Bryce's (which was written by Eric Wenger and myself). Well, some of those "noises" are really textures, not basis functions. You can obtain similar results, with far more flexibility, using MojoWorld's *function fractals*, which will build a fractal from whatever function you pass to them—and probably alias like crazy while they're at it. Such aliasing is why many things that are used as basis functions in Bryce shouldn't be used for that. Others, like Eric's "techno noise," don't lend themselves to the level-of-detail schemes used in MojoWorld. As a rule, for deep mathematical reasons that I

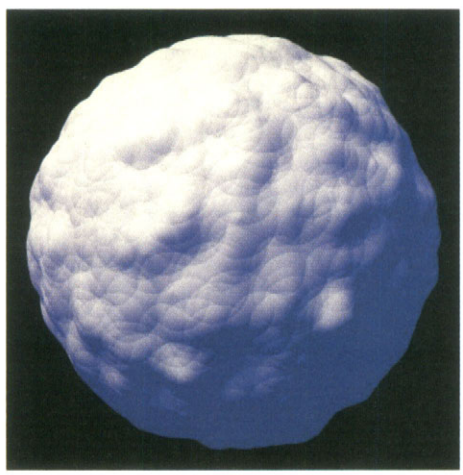

FIGURE 20.22 A planet made from the sparse convolution basis, using a cone for the kernel.

FIGURE 20.23 A planet made from a 3D sine basis.

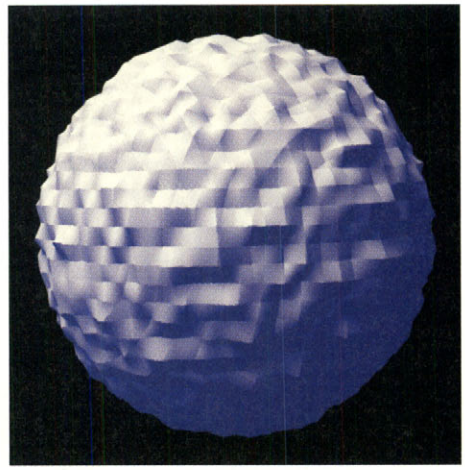

FIGURE 20.24 A planet made from the linear basis function.

won't go into here, basis functions should be visually simple. So all of MojoWorld's basis functions are—with the exception of sparse convolution when using a complex kernel. Keeping to this constraint of simplicity, here are a few more basis functions.

First there's the venerable *sine* wave. As a function of more than one variable, it's a little ambiguous what a sine wave should be. In MojoWorld we multiply sine waves along the various dimensions. As the sine function is periodic, anything built using the sine basis will be periodic (see Figure 20.23). Periodic phenomena are quite common in nature, but they tend to look unnatural in synthetic images. Nature can get away with things that we can't.

The *linear* basis function is a simpler version of the simplest Perlin basis. It uses linear interpolation of random offsets, rather than the smooth cubic spline used in the Perlin bases. It will give you straight, sharp creases (see Figure 20.24). Not very natural-looking, but I'm sure someone will find a use for it.

The *steps* basis is simpler still: it's just a bunch of random levels in a cubic lattice (see Figure 20.25). And then there's the procedural texture that any student of computer graphics programs first: the common *checkerboard*. In MojoWorld, the checkerboard basis alternates between 1.0 and −1.0. The little sawtooth artifacts you see on the cliff faces in the steps and checkerboard planets (Figures 20.25 and 20.26) are the micropolygons that compose them. They can be made smaller, at the expense of

FIGURE 20.25 A planet made from the steps basis function.

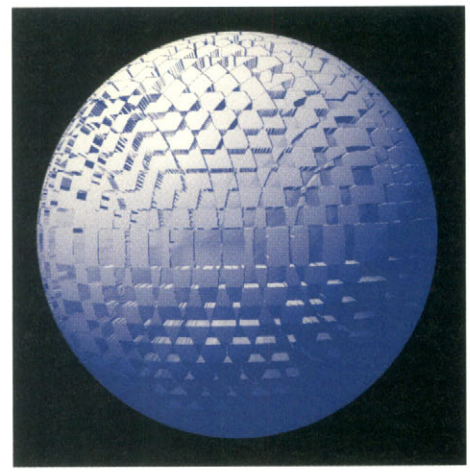

FIGURE 20.26 A planet made from the checkerboard basis function.

longer render times, but they'll never go away completely. Discontinuous basis functions like these are mathematically evil. And so they really shouldn't be used, but what the heck, MojoWorld is for play as well as serious work, so not everything has to be perfect.

The Seed Tables

Here's another MojoWorld first. Call me a noise dweeb, a basis function junkie, or just plain ill advised, but . . . "I just had to." The Voronoi and sparse convolution basis functions are built from tables of random points in space. It just turns out that there are many forms "random" can take. (Take a class in probability or statistics and learn to hate them.) So I implemented a mess of different ones and whacked 'em into MojoWorld.

Look at them closely and try to distinguish the subtle visual differences between them (see Figure 20.27). In general, the ones with larger numbers have a denser spatial distribution. The ones with "uniform distribution" are more dense in some areas and less dense in others—more heterogeneous, in a word. Uniform distribution means that each cell has an equal probability of having any of the possible range of seeds per cell in each cell. Thus the "0–2 per cell, uniform distribution" seed table has cells that have zero, one, or two seeds per cell, with equal probability of each.

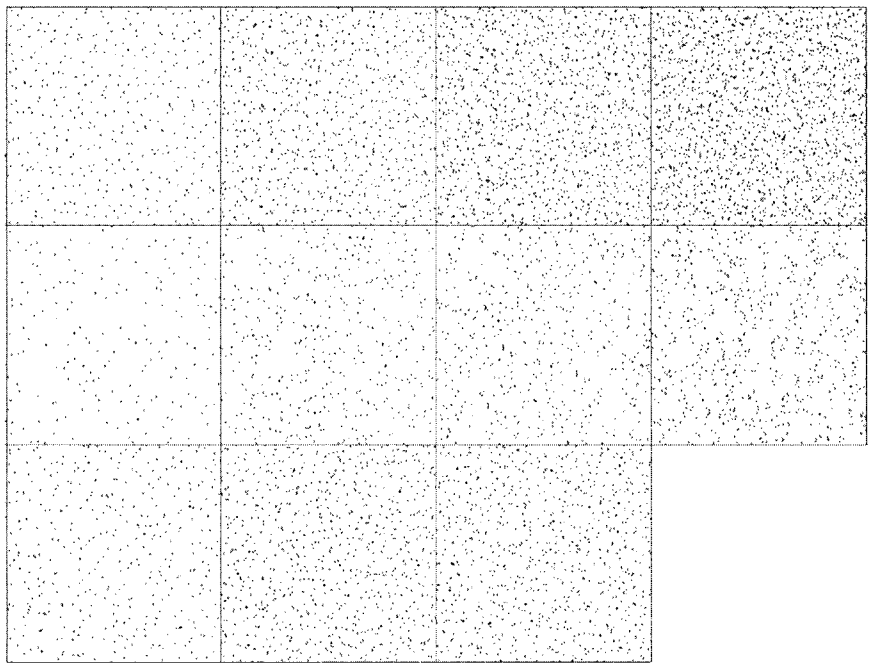

FIGURE 20.27 The distributions of seeds in the 11 seed tables available in MojoWorld: 1, 2, 3, and 4 seeds per cell; 0–1, 0–2, 0–3, and 0–4 seeds per cell, uniform distribution; and 0–2, 1–3, and 0–4 seeds per cell, Gaussian distribution.

The seed tables with "Gaussian distribution" are more subtle: there is a higher probability of a given cell having the middle of the possible number of seeds per cell than the greatest or smallest possible number. This is the good old "bell curve" that statisticians are so fond of. So the "0–4 per cell, Gaussian distribution" seed table will have mostly cells with two seeds each, very few with zero or four seeds, and an intermediate number of cells with one or three seeds.

At any rate, just evaluate the seed tables visually and use them if you like, or just ignore them—they are a *very* advanced feature for subtle effects. Figure 20.28 illustrates some of the extremes in the visual consequences available with different random seed tables. Note that even these "extremes" are only subtly different; there are other tables with intermediate values to make the possible transitions all but imperceptible. A few perfect masters of MojoWorld will someday find these subtle differences useful, I predict.

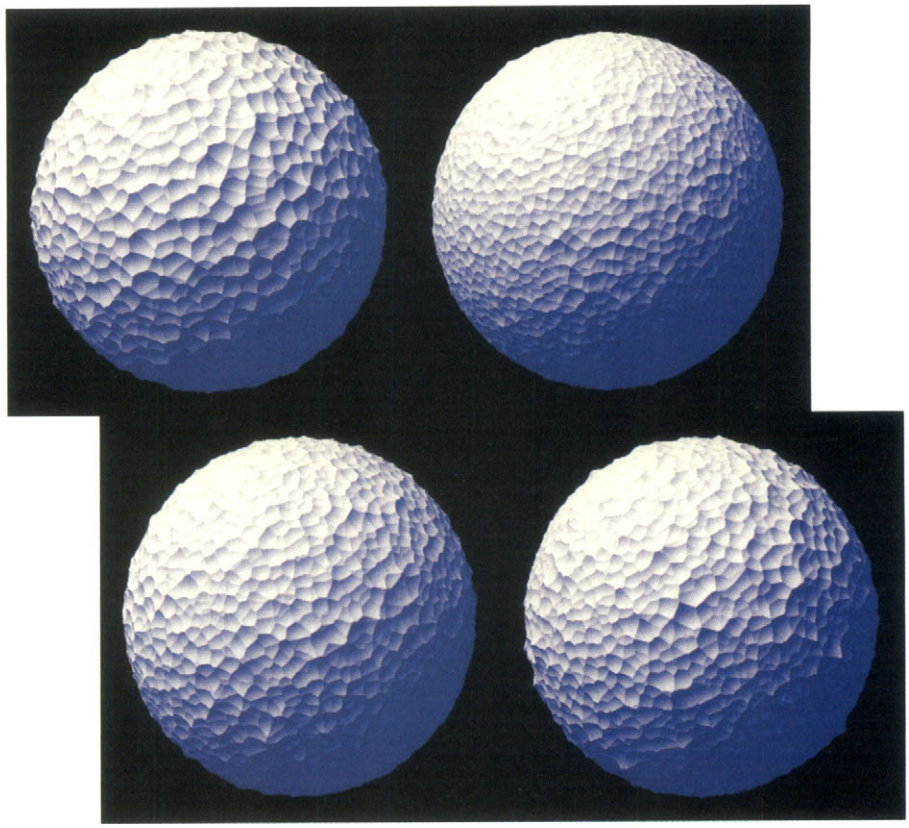

FIGURE 20.28 Planets made from Voronoi first neighbor with, going clockwise from upper left, seed tables with 1 seed per cell; 4 seeds per cell; 0–4 per cell, uniform distribution; and 0–4 per cell, Gaussian distribution.

Monofractals

Much like the various possible distributions of random numbers I just glossed over so quickly, there are even more complicated mathematical measures that characterize the randomness in random fractals. I'll do my best to simplify and clarify the complicated and obscure here, so please bear with me.

Early fractal terrains were derived from a mathematical function called *fractional Brownian motion* (fBm). fBm is a generalization of Brownian motion, which

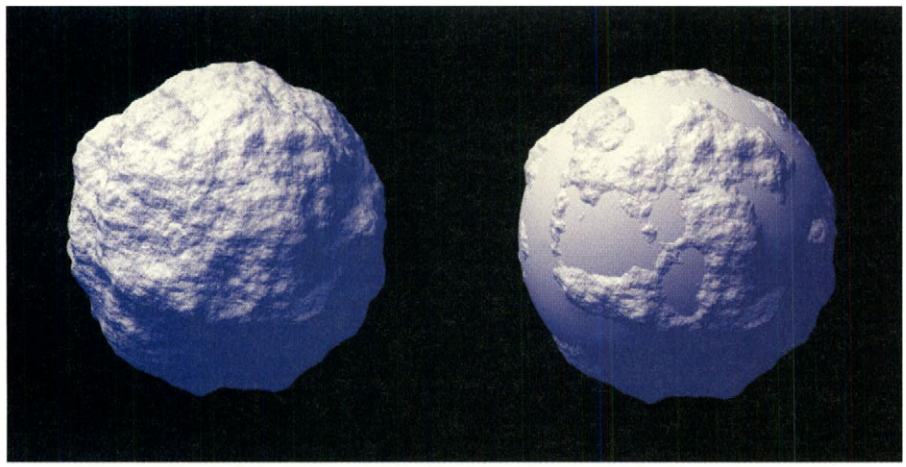

FIGURE 20.29 A planet made from monofractal fBm, a Perlin basis, and roughness of 0.3, with and without "oceans."

you may remember from your high school science classes: Brownian motion is the random walk of a very small particle being jostled about by the thermal motions of the much smaller particles comprising the fluid in which it is suspended. It's a lot like the random walk of an aimless drunk staggering about on a flat plain. fBm has a bunch of specific properties, and foremost among them is its uniformity: it's designed to look the same at all places and in all directions.[3] You can think of it as "statistically monotonous"—hence the name *mono*fractal.

This was a good start, and variations on fBm, generated from a variety of different basis functions, remain the standard random fractal used in MojoWorld and other fractal programs that create models of natural phenomena like mountains, clouds, fire, and water (see Figure 20.29).

Multifractals

Real terrains are not the same everywhere. Alpine mountains can rise out of flat plains—the eastern margin of America's Rocky Mountains being a conspicuous example. Early in my work with Mandelbrot, I wanted to capture some more of

3. In math lingo, this property is called *statistical stationarity.*

that kind of variety in fractal terrains, without complicating the very simple mathematical model that gives us fBm. As usual, I was reasoning as an artist, not a mathematician. I had some ideas about how the fractal dimension, or roughness, of terrain should vary with altitude and slope. For example, I knew that lakes fill in with sediment and become meadows as geologic time passes, so I thought, "Low areas should remain smooth, while peaks should be rough." Interestingly, the opposite appears to be more common in nature, but what the heck, I was working in a dark closet (no kidding) in the Yale math department at the time, so I was just working from memory, not active field work. I fiddled with my math—more, with my programs that implemented the math.

In order to escape Yale with my Ph.D. and not get roasted like a pig on a spit at my thesis defense, I tightened up my descriptions of these multifractal functions and did some interesting experiments that led to dead ends, from the standpoint of my ultimate goal, making MojoWorlds. I did get a little attention from some physicists interested in multifractals, which I thought was cool, as an artist. At any rate, I packaged up the two best-behaved multifractals I had devised and stuck them in MojoWorld.

"What do I care?," you ask. Well, monofractals get pretty boring pretty fast because they're the same everywhere, all the time. Multifractals are a little more interesting, visually: they're *not* the same everywhere. They're smoother in some places and rougher in others. Nature, of course, is far more complex than this, but hey, it's a second step in the right direction. In general, I use multifractals for my terrains and usually use monofractals for clouds and textures.

The Heterofractal Function

The first of the two multifractal functions in MojoWorld I call *heterofractal*. Its roughness is a function of how far it is above or below "sea level" (where it tends to be quite smooth and boring). You can add in another function to a heterofractal terrain to move the terrain around vertically, so that the smooth areas don't always occur at the same altitude (see Figure 20.30).

The Multifractal Function

The second of the two multifractal functions in MojoWorld is simply named *multifractal*. Back when I was still trying to figure out the Byzantine mathematics of multifractals, as in my dissertation and the first edition of this book, I called this

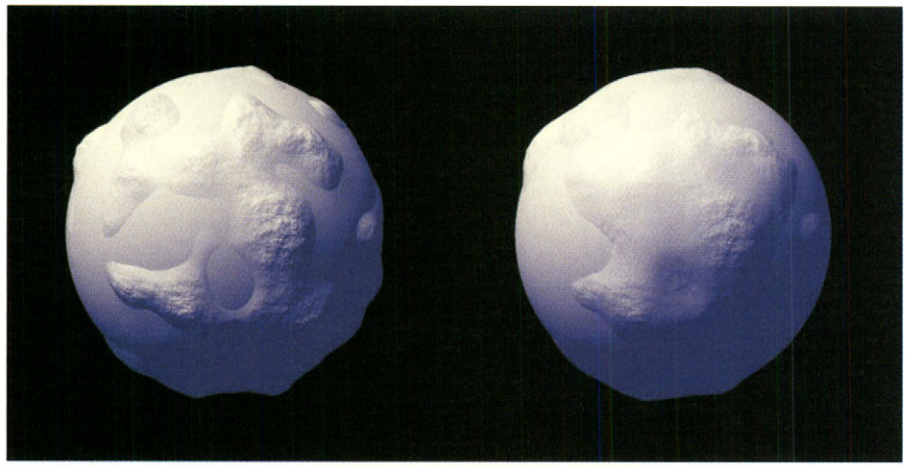

FIGURE 20.30 Planets made from a heterofractal with a Perlin basis and an "ocean" sphere at sea level. Straight heterofractal on the left, heterofractal plus a random vertical displacement on the right.

function a "hybrid multifractal," for technical reasons. What I was then calling "multifractal" turned out to be a dead end, for practical purposes, so I've since promoted "hybrid multifractal" to "multifractal" in my personal lexicon, and in MojoWorld's, by transitivity. It's a good workhorse and my first choice for terrains on a planetary scale most every time (see Figure 20.31).

Function Fractals

A fractal consists of some form repeated over a range of scales. There is no inherent restriction on what that form might be. In the good, safe practice of image synthesis, however, there are some fundamental restrictions. More math being glossed right over: There is a highest spatial frequency that can be used when synthesizing images on the computer. There is a mathematical theorem that tells us what that highest frequency is—the *Nyquist sampling theorem.* Frequencies higher than that limit, the *Nyquist limit,* will *alias* (turn into undesirable artifacts) in our images. The upshot: You don't want to go building fractals out of just anything; you really want to know what the highest spatial frequency is in your basis function. But we know you most likely care more about making interesting pictures than hearing about mathematical limitations. Hence we have *function fractals* in MojoWorld (see Figure 20.32). Rather than limiting you to using the carefully crafted basis functions built by well-

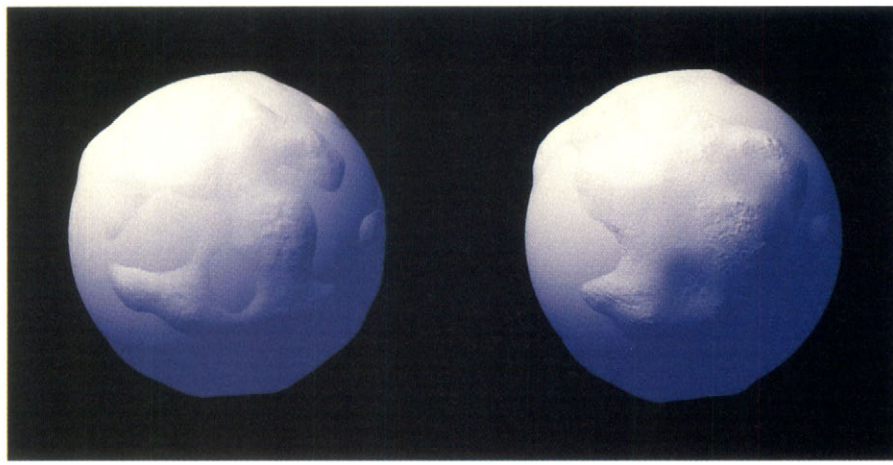

FIGURE 20.31 Planets made from multifractal 3 with a Perlin basis and a smooth sphere at sea level. Straight multifractal on the left, multifractal plus a random vertical displacement on the right.

FIGURE 20.32 A planet made from a 3D function fractal.

informed professional engineers, with function fractals you can build them out of whatever ill-informed, ill-advised basis function you can come up with.

But be warned, there are two predictable consequences of using function fractals: First, they will tend to be slow. The arbitrary basis function you put in may already be a fractal that requires thousands of CPU cycles to evaluate. Repeat that at a

hundred different scales, and rendering your image will redefine the word "slow." Second, such fractals are very likely to alias badly. Such aliasing will usually look kind of like sandpaper in a still image—not too evil. But when you animate it, it can sizzle in a most annoying way. Also, the way MojoWorld is designed, if you have aliasing anywhere in your image, you're likely to have it everywhere, whereas in ordinary 3D graphics software aliasing generally increases with distance.

Don't say we didn't warn you. But then, we have a motto here at Pandromeda: "Give 'em enough rope."

Domain Distortion

When I first started playing with procedural textures like we're using in MojoWorld, one of the first things I tried is distorting one texture with another. Because we accomplish this by adding the output of one fractal function to the input of another, we call it *domain distortion*. Imagine a function with a one-dimensional domain, say, the sine wave. Undistorted, it looks like Figure 20.33.

Imagine adding the cosine to x before we pass x to the sine function. What we're computing is then not the sine of x, but the sine of x plus the cosine of x. As the value of the cosine function varies from -1.0 to 1.0 and is added to x, it has the effect of displacing the x value that gets passed to the sine function, moving it back and forth from its undistorted value (see Figure 20.34). Distorting the input of a function has the effect of distorting the output. We see such a result in Figure 20.35.

Of course, this example is what mathematicians call "trivial." It's just to illustrate the process simply and clearly. Domain distortion in MojoWorld will generally involve more complex functions and take place in higher dimensions, but the way it's done is exactly the same. Figure 20.36 shows planets made from an undistorted fractal and the same fractal with domain distortion. The domain distortion has the effect of stretching the distorted texture out in some places and pinching it together in others.

MojoWorld's Texture Editor allows domain distortion both to the basis functions and to the aggregate fractal. For efficiency and to avoid severe aliasing, basis

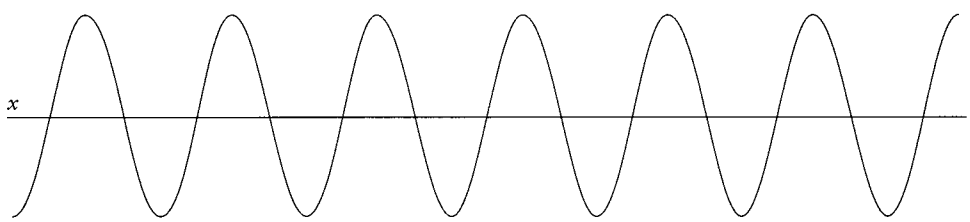

FIGURE 20.33 An undistorted sine wave: $y = \sin(x)$.

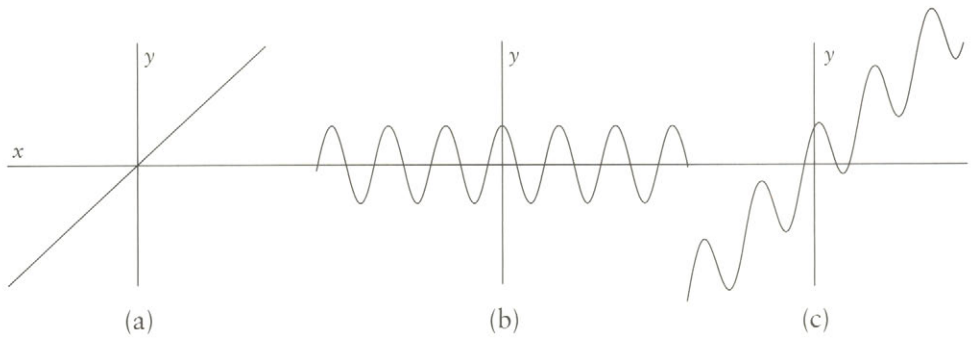

FIGURE 20.34 (a) The undistorted domain $y = x$, (b) a distortion function $y = \cos(x)$, and (c) the distorted domain $y = x + \cos(x)$.

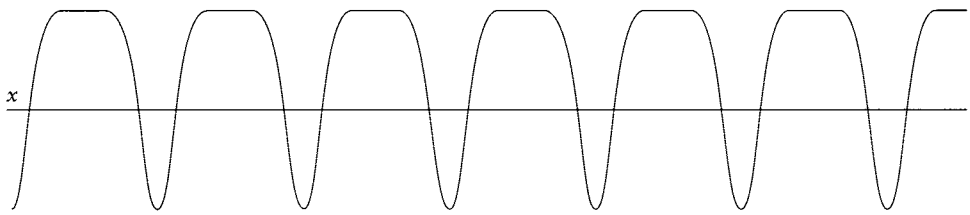

FIGURE 20.35 A distorted sine wave $y = \sin(x + \cos(x))$.

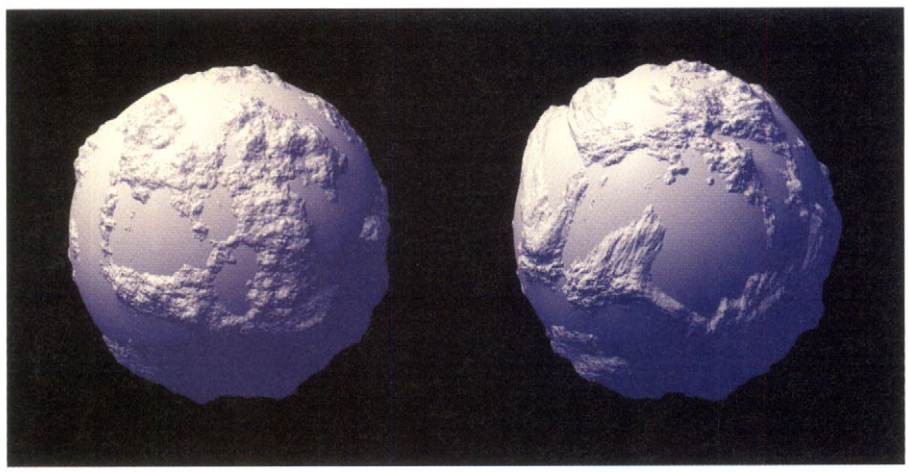

FIGURE 20.36 A planet with undistorted and distorted fractal relief.

functions can only be distorted with other basis functions. On the other hand, fractal functions can be distorted with other fractals—indeed, with any function. There are at least two reasons for the two being treated differently. The first is that, for three-dimensional functions, the distortion function is three times as expensive to evaluate as the distorted function. This is because the distortion function has to be evaluated once for each of the three input dimensions to the distorted function, while the distorted function only has to be evaluated once, using that distorted three-dimensional input. The second is that the distortion function has to be evaluated once per octave of the distorted function, in the case of distortion applied to the basis function, but only once for distorting the aggregate fractal. As MojoWorld is churning through around 25 to 30 octaves—and potentially many more—when you're down close to the surface of a planet, performing anything other than simple distortion on a per-octave basis is simply not computationally practical—yet. No doubt it will become so in years to come, with faster computers, but it will still present aliasing problems.

Distorted Fractal Functions

You can use domain distortion to make long, linear mountain ranges in the areas where the distorted texture, stretched in one direction and pinched in the other. But when you zoom in to those mountains, it's not very realistic to have all the little details stretched and pinched in the same way—real mountains just don't look like that. So I borrowed an idea from turbulent flow, *viscous damping,* to get around that. Turbulent flow is damped, or slowed, by viscosity in the fluid at small scales.[4] Since domain distortion is kind of like turbulence, I thought, "Why not do the same thing with the domain distortion? Taper it off at smaller scales." So MojoWorld has what I call *distorted fractal functions* available in the Graph Editor, or Pro UI. These are complicated little beasts—definitely a very advanced feature. Beginners will find them hopelessly confusing, I fear. But they have two fields, *onset* and *viscosity,* in which you can specify where the viscous damping begins and is total (no distortion), respectively. The scales are specified in meters, the default unit of scale in MojoWorld.

Crossover Scales

This idea of scales, for the onset of viscosity and where viscous damping is complete, leads to our next and last advanced feature in MojoWorld fractals: *crossover scales.*

4. "Small" is a relative term here; the scales at which viscosity damps turbulence depends on the viscosity of the turbulent fluid, which can be anything from molten rock, as in plumes in the Earth's mantle, to the tenuous gas in a near-perfect vacuum that we see in Figure 20.6.

A crossover scale is simply a scale where the behavior of a fractal changes. The simplest examples are the *upper crossover scale* and *lower crossover scale,* above and below which fractal behavior vanishes.

MojoWorld has such crossover scales. There's always a largest feature size for any MojoWorld fractal, and if you don't explicitly set a lower crossover scale, MojoWorld will eventually say "enough" and quit adding detail. (That limit is up to the MojoWorld programmers.) In the *distorted fractal* functions we employ another kind of crossover scale, in another very advanced MojoWorld feature. I figured that zooming in on a single fractal, from the scale of continents to that of dirt, is neither very interesting nor realistic. In Nature, the character of terrain is different at different scales. So, in MojoWorld's distorted fractals, I included the ability to use different basis functions at different scales. You can use up to three different basis functions in these fractals. When you use more than one, you have to specify the scale, in meters, where the crossover between basis functions begins and ends. It's not easy to show how this works, other than in an animated zoom. So please accept my apologies; no illustration here.

Driving Function Parameters with Functions

One of the most powerful features of the MojoWorld Graph Editor, or Pro UI, is the ability to drive the value of almost any parameter of any function with the output of any other function. This can give some really wild and complicated results! Once you've become an advanced MojoWorld user, I recommend going into the Pro UI and playing with this. (It will probably be hopelessly confusing until you learn to think in the new, purely procedural MojoWorld paradigm.)

For example, using a "blend" node you can easily make a texture whose color is white above a certain altitude—the snow line. But a straight, horizontal snow line is not very natural-looking. So you might create an "add" node, with "altitude" as one input and a fractal as the other. The add node is then a function with inputs and an output. You can hook the output of the add node to the parameter that controls where the blend node makes the transition to white. Now the snow line will be fractal and quite natural-looking!

A potent hidden aspect of the MojoWorld Graph Editor is that each node knows the dimensionality of the input it needs. All nodes will automatically provide output of the dimensionality requested by the parameter they are hooked into. Note that this doesn't free the user from having to make the right choice of dimensionality for certain function modes, as MojoWorld can't know what kind of effect you're out to create, and so it can't always make the choice for you.

USING FRACTALS

I've talked a lot about fractals and all their wonderful complexities in MojoWorld. Now let me talk a little about the specific uses for fractals.

Textures

Perhaps the main use for fractals in MojoWorld is in surface textures. Sure, MojoWorld can create some very complex geometry, but you can still get most of your interesting visual information from textures applied to surfaces. The procedural methods used in MojoWorld were originally designed by Darwyn Peachey and Ken Perlin for creating such surface textures. You can create some really beautiful effects in color alone, using fractal procedural textures. See Figures 20.37 and 20.38 for examples from MojoWorld.

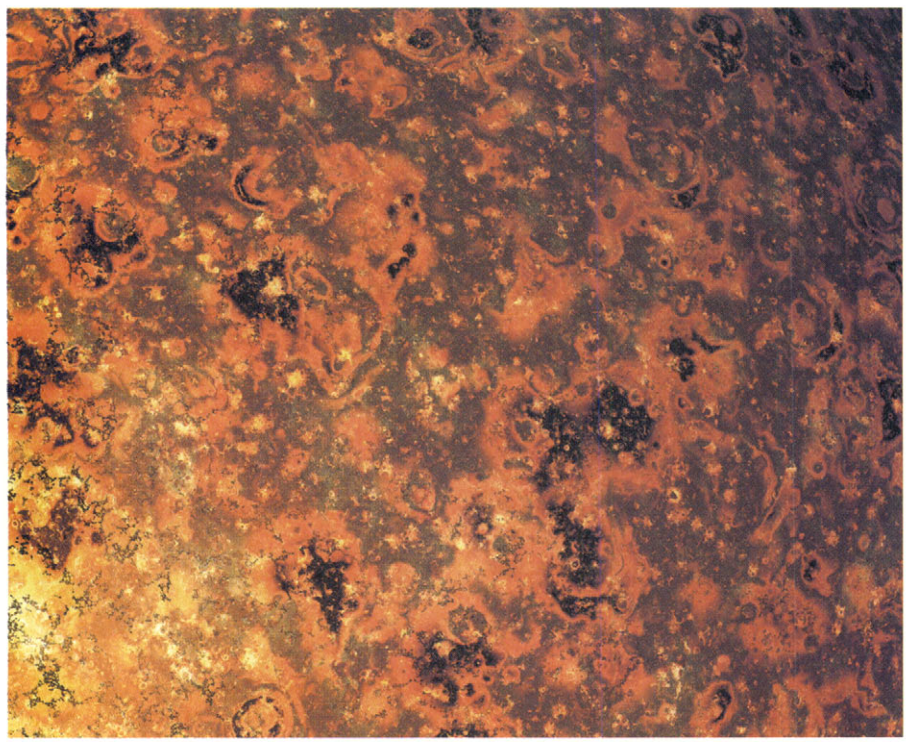

FIGURE 20.37 This MojoWorld texture study by Jonathan Allen illustrates the kind of complex and abstract beauty that can issue from procedural textures. Copyright © 2002 Jonathan Allen.

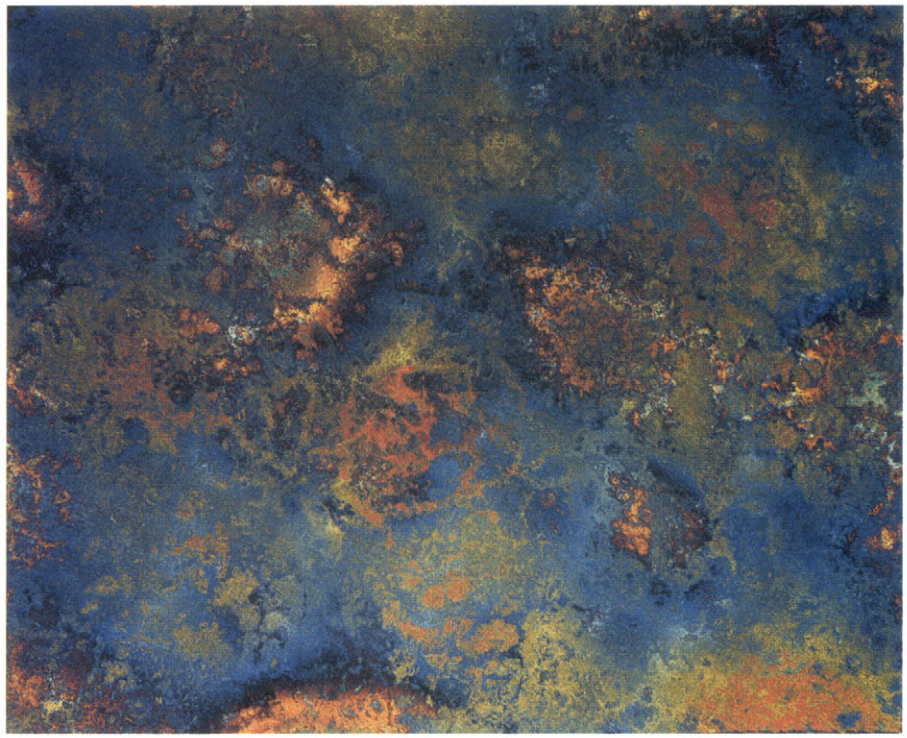

FIGURE 20.38 Another MojoWorld texture study: Complex behaviors result from fractals constructed from different basis functions, blended using various combination modes, with surface displacements for additional subtle detail. This and all the MojoWorld images in this chapter can be reproduced from the mjw files available at this book's Web site.

The development and artful use of fractal textures has pretty much made my career. Such texture functions can be used to control surface color, shininess, displacement, transparency—you name it. Pretty much everything that makes a MojoWorld interesting and beautiful is some form of a fractal procedure, or *procedural texture*. That's why the Texture Editor is the very heart of MojoWorld. Doc Mojo's advice: Spend time mastering the Texture Editor. It is by far the most powerful tool in MojoWorld.

Terrains

Even the terrains that comprise a MojoWorld's planetary landscape are just procedural textures, used in this case to determine elevation. For consistency, the terrain

texture is evaluated on the surface of a sphere of constant radius, and the result is used to raise or lower the planet's terrain surface. The multifractal functions in MojoWorld were originally designed for modeling terrain, so I recommend using them when you're creating the terrain for a MojoWorld.

Displacement Maps

MojoWorld also features *displacement maps*—textures that can actually displace the surfaces they're applied to. Yes, a MojoWorld is just a displacement-mapped sphere. But the algorithm used to displace the planet's surface is a special one, designed for speed (at the expense of memory). There are two consequences to this: First, you can make displacement-mapped spheres for moons, but you can't zoom into them like you can a MojoWorld. (Well, you *could,* but the renderer would crawl to a halt, as you got close. We'll fix this in a future version.) Second, you can do lateral displacements on the MojoWorld terrain to get overhangs. Figure 20.39 shows an example of this. This is a majorly cool feature of MojoWorld, although extreme

FIGURE 20.39 *Peeking* demonstrates an extreme example of displaced displacements in MojoWorld: first, the planet sphere is displaced using sparse convolution to create the spires, then a Voronoi-based displacement material is applied to the surface, creating overhangs on the near-vertical surfaces. Copyright © 2002 Irene Alora.

displacements can throw MojoWorld's photorealistic renderer, producing unpredictable artifacts. MojoWorld's real-time renderer can only handle one displacement—that of the terrain on the planet sphere—so such lateral displacements aren't visible there at all.

Clouds

MojoWorld 1.0 Near Space only has two-dimensional clouds, mapped onto spheres concentric with the planet. You can put any texture you like on those spheres to represent your clouds. (You can do some really wild clouds—go for it!) Future versions will have full volumetric three-dimensional clouds. "We have the technology." See the "Great Balls of Fire" section of my Web site—*www.pandromeda.com /musgrave*—for some animated examples of what I've done in the past. We'll include those and more in future versions of MojoWorld.

More exciting still, in a future version we'll put in four-dimensional clouds. This will allow you to animate volumetric clouds over time. That will be fun!

Planets

Obviously, MojoWorld is an exercise in modeling entire planets with fractals. The possibilities are endless. I'm looking forward with great anticipation to seeing what people create and find in Parametric Hyperspace. They're all out there, virtually, just waiting to be discovered. Figure 20.40 is one of may favorite early examples.

Nebulae

Figure 20.6 illustrates rather convincingly that astrophysical nebulae can be fractal in nature. Figure 20.41 illustrates an early experiment of mine in modeling with a multifractal texture—interstellar dust clouds like we see in the dark lanes in the Milky Way.

Planets reside in solar systems, solar systems in galaxies, and galaxies in clusters. In future versions of MojoWorld we plan to model all these things. Volumetric nebulae are something we're all looking forward to seeing, playing with, and zipping through!

THE EXPRESSIVE VOCABULARY OF RANDOM FRACTALS

People often ask me, "Can you do people with fractals?" The answer is no. Not everything is fractal. People don't look the same on a variety of scales (although parts of us, like our lungs and vascular systems, do). There is a limit to what can be done

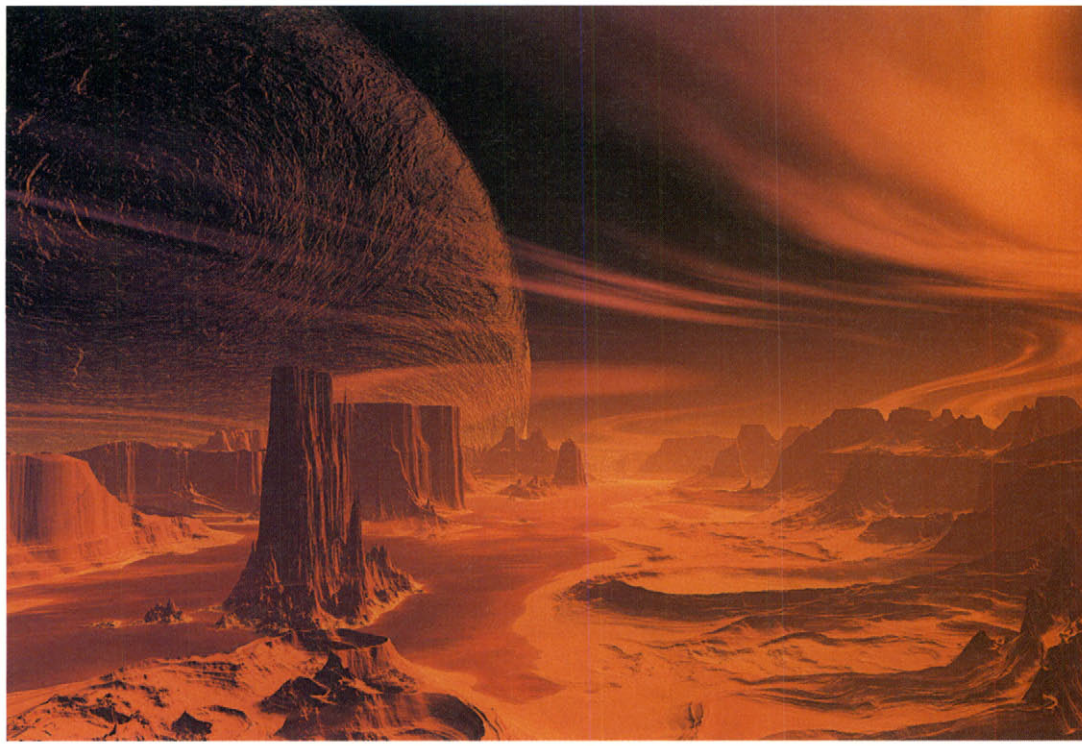

FIGURE 20.40 *Planet Copperwine* is probably my favorite early MojoWorld image. The fantastic terrain, colors, moon atmosphere, clouds, and sea all make this a world that calls to me in a deep way. Copyright © 2001 Armands Auseklis.

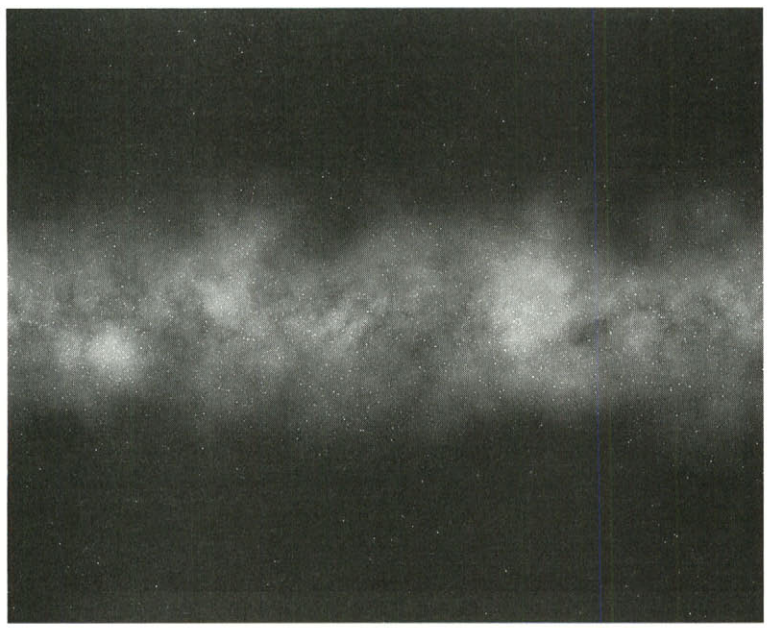

FIGURE 20.41 A multifractal texture as a model of the Milky Way.

with fractals. We can do a lot, but certainly far from *everything*. MojoWorld is designed to do most of what can be done with random fractals today. So while we can't do everything we'd ever want to do in MojoWorld, what we *can* do will keep us busy and entertained for some time to come. MojoWorld reveals a rich, new creative space for us to explore, and that exploration will keep us entertained for the rest of my life, I'm quite certain.

EXPERIMENT!

Get in there and experiment. Although Parametric Hyperspace as spanned by MojoWorld version 1.0 certainly doesn't include every world we'd ever like to see and explore, it does span an infinitely vast virtual universe, much larger in fact than the one we inhabit, simply because it has so many dimensions. Get in there and find/create cool images/places and *share them* with the rest of us! You never know:

FIGURE 20.42 *Dale—Winds of Interference* illustrates a MojoWorld—derived from *Planet Copperwine*—that calls for exploration, imaging, and perhaps colonization. Copyright © 2002 Armands Auseklis.

FIGURE 20.43 The parallel universe of Parametric Hyperspace is a place where your imagination can run wild. *Charon* is an example of the kind of place that you can construct/find there.

planets are big places; someone else is very likely find a more beautiful view on a planet you've created than any you've yet found. If you become a master builder of MojoWorlds, you won't have the time you need to explore them properly. You'll need help. We foresee at least two kinds of MojoWorld users: creators and explorers. The creators will do the work of constructing new planets, while the explorers will be the landscape photographers, the journalists who find and document in images the beauty in these worlds. A true team effort—that's another fun aspect of MojoWorld.

THE FUTURE

We've put into MojoWorld everything I can think of that's practical and even a few things—like the "distorted fractal" functions—that probably aren't. Yet. What's practically doable on your home computer is a fast-moving target: they get faster at an amazing rate. We've designed MojoWorld to push the state of the art. And

MojoWorld can bring any processor in the world to its knees quite easily. Of course, we have other algorithms like radiosity (superaccurate illumination calculations), physically based erosion, and fluid dynamics that we could stick in MojoWorld, just in case we need to slow things down some more.

The Holy Grail we seek is virtual reality. Not the lame, anything-but-real stuff we've seen called "VR" to date, but *believable* virtual reality—interactive MojoWorlds as beautiful and realistic as those we can render currently in non–real time. It will happen. And not too long from now. It's only a matter of engineering.