The Normal of a Fractal Surface

Wayne O. Cochran* Robert R. Lewis†

John C. Hart‡

November 9, 2000

Keywords: fractal, fractal function, fractal terrain, iterated function system, recurrent modeling

*School of Electrical Engineering and Computer Science, Washington State University; 14204 NE Salmon Creek Avenue; Vancouver, WA 98686-9600.

†School of Electrical Engineering and Computer Science, Washington State University; 2701 University Drive; Richland, WA 99352.

‡Department of Computer Science, University of Illinois, 1304 W. Springfield Ave.; Urbana, IL 61801.

Abstract

We have discovered a class of fractal functions that are differentiable. Fractal interpolation functions have been used for over a decade to generate rough functions passing through a set of given points. The integral of a fractal interpolation function remains a fractal interpolation function, and this new fractal interpolation function is differentiable. Tensor products of pairs of these fractal functions form fractal surfaces with a well defined tangent plane. We use this surface normal to shade fractal surfaces, and demonstrate its use with renderings of fractal mirrors.

1 Introduction

There are many definitions of the term *fractal*. Most involve some property of detail at all levels of magnification. This property tends to imply that any fractal function is not differentiable. Given a real function $f(x)$ of a single real variable x, its derivative is commonly defined as

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}. \tag{1}$$

The graph of a fractal function f has detail at all levels of magnification. As h approaches zero (from either direction), the limit in (1) may not converge.

Fractals have found numerous applications in computer graphics as a representation for rough surfaces, particularly for modeling terrain [FFC82, Man82, Kaj83, Bou85, Mil86, MKM89]. In the absence of an analytic derivative, the shading of these fractal surfaces is typically approximated using the surface normals from the terminal level of surface subdivision. However, the shading from the normals at a given level of surface subdivision is not nec-

essarily representative of reflectance of the rough surface found at the limit of subdivision.

Fractal surfaces can also be shaded using a weighted average of surface normals taken over a variety of subdivision levels [HD91]. However, this normal was designed to visually indicate the scalar self-symmetry of the fractal surface, and was not meant to approximate the shading of the limit surface.

Instead, we consider an alternative implicit definition of the derivative as the solution f' of

$$f(x) = y_0 + \int_0^x f'(t)dt. \tag{2}$$

In this ordinary differential equation form, one can describe a function f by its derivative f' and an initial value $f(0) = y_0$.

In this manner, we model a fractal surface by modeling its derivative. We use the representation of fractal interpolation functions [Bar86], developed over a decade ago, along with an associated calculus that shows that the fractal interpolation function representation is closed under integration [BH89]. In this manner,

we have a well-defined representation for a fractal surface and an equally well-defined representation for its derivative, and thus its surface normal.

Fractal terrain surfaces based on fractional Brownian motion, which is a self-affine process, result in functions that appears steeper at finer levels of magnification. While this simulates natural structures well, in the mathematical limit we find that the slope tends to infinity. Alain Fournier [Fou] observed that such surfaces are blackbodies, absorbing all incoming light in a myriad of self-shadowed interreflections. The fractal terrain models derived in this paper do not have such slopes than tend to infinity across finer scales, and are perhaps a more appropriate model for determining bidirectional reflectance distributions for rough surface shading models.

As for increasing slope variance at finer scales to simulate physical structural characteristics, the terrain model in this paper is offered as a primitive. These primitives represent a uniformly

distributed slope variance across all scales. This characteristic is appropriate for typical terrain databases that use many primitives to interpolate dense arrays of data points, as opposed to fractal terrain models that use few primitives to synthesize typical features that could possibly be found in reality.

2 Fractal Interpolation Functions

Let $x_0 < x_1 < \cdots < x_N$ be a partitioning of the interval $\mathbf{X} = [x_0, x_N]$. Let $L_j : \mathbf{X} \to \mathbf{X}$ be the affine map

$$L_j(x) = a_j x + b_j \tag{3}$$

where $0 \leq a_j < 1$ and b_j are such that

$$L_j(x_0) = x_{j-1}, \tag{4}$$

$$L_j(x_N) = x_j, \quad j = 1, 2, \ldots, N, \tag{5}$$

yielding

$$a_j = (x_j - x_{j-1})/(x_N - x_0), \tag{6}$$

$$b_j = x_{j-1} - a_j x_0, \quad j = 1, \ldots, N. \tag{7}$$

Let $y_0, y_1, \ldots, y_N \in \mathbb{R}$ (the points (x_i, y_i) are often referred to as *knots* or *control points*). Let $F_j : \mathbf{X} \times \mathbb{R} \to \mathbb{R}$ be

$$F_j(x, y) = \alpha_j y + q_j(x) \tag{8}$$

where $j = 1, 2, \ldots, N$, $-1 < \alpha_j < 1$, and q_j is a Dth degree polynomial

$$q_j(x) = \sum_{i=0}^{D} q_{ji} x^i, \; q_{ji} \in \mathbb{R} \tag{9}$$

where

$$F_j(x_0, y_0) = y_{j-1}, \tag{10}$$

$$F_j(x_N, y_N) = y_j, \quad j = 1, 2, \ldots, N. \tag{11}$$

We refer to a_j as the *horizontal contraction factor* or *horizontal Lipschitz constant* since

$$|L_j(x_1) - L_j(x_2)| \leq a_j |x_1 - x_2|, \tag{12}$$

for $x_1, x_2 \in \mathbf{X}$. The parameter α_j is termed here as the *vertical contraction factor* or *vertical Lipschitz constant* since

$$|F_j(x, y_1) - F_j(x, y_2)| \leq |\alpha_j| |y_1 - y_2|, \tag{13}$$

for $x \in \mathbf{X}, y_1, y_2 \in \mathbb{R}$.

Each map $w_j = (L_j(x), F_j(x,y))$ can be defined to operate on a set of points A

$$w_j(A) = \bigcup_{(x,y)\in A} w_j(x,y), \tag{14}$$

and the simultaneous operation of all maps on a set A is defined by the *Hutchinson operator*

$$\mathbf{W}(A) = \bigcup_{i=1}^{N} w_j(A). \tag{15}$$

The attractor $\mathcal{A}$ is the fixed point

$$\mathbf{W}(\mathcal{A}) = \mathcal{A}, \tag{16}$$

so that every point on $\mathcal{A}$ maps to some (usually) other point on $\mathcal{A}$. $\mathcal{A}$ is the graph of the associated *fractal interpolation function* (FIF) which is the unique function $f : \mathbf{X} \to \mathbb{R}$ satisfying

$$f(L_j(x)) = F_j(x, f(x)),\ j = 1, 2, \ldots, N, x \in \mathbf{X}. \tag{17}$$

Two fractal interpolation functions are illustrated in Figure 1. The graph of the function f on the left interpolates the points (0,0), (1,1), and (2,0) using the functions

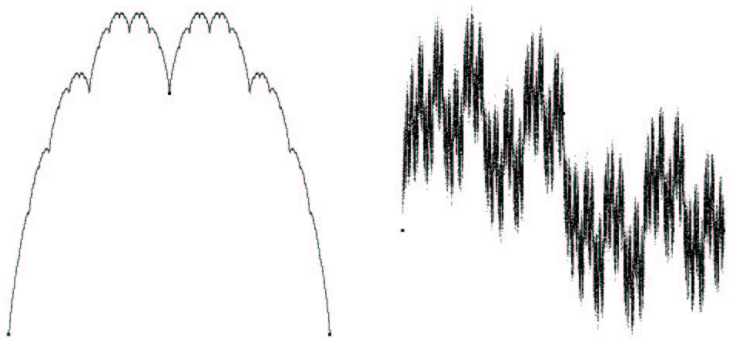

Figure 1: Example two-map fractal functions interpolating three control points: f (left) and g (right).

$$L_1(x) = \frac{1}{2}x \tag{18}$$

$$L_2(x) = \frac{1}{2}x + 1 \tag{19}$$

$$F_1(x,y) = \frac{1}{2}y + \frac{1}{2}x \tag{20}$$

$$F_2(x,y) = \frac{1}{2}y + 1 - \frac{1}{2}x \tag{21}$$

The graph of the function g on the right interpolates the points (0,0), (1,1), (2,0) using the same horizontal contractions, and the

vertical contractions

$$G_1(x,y) = \frac{17}{20}y + \frac{1}{2}x \tag{22}$$

$$G_2(x,y) = -\frac{17}{20}y + 1 - \frac{1}{2}x \tag{23}$$

2.1 Evaluating FIF's

Algorithm 1 FIF chaos game.

Input: $\{w_j\}_{j=1}^{N}$, seed (x_0, y_0)

$(x, y) \leftarrow (x_0, y_0)$

for $i = 1$ to *iterations* **do**

$j \leftarrow$ random index in $\{1, \ldots, N\}$

$(x, y) \leftarrow w_j(x, y)$

plot(x, y)

end for

We can use the *chaos game* to generate points on the attractor as shown in Algorithm 1. To actually evaluate $f(x)$ we must use an iterative approach by composing the sequence of transformations which (approximately) map some initial point (x_0, y_0) to $(x, f(x))$ as described in Algorithm 2. We can form the composite $w_c =$

$w_i \circ w_j$ where $L_c(x) = L_i \circ L_j(x)$ and $F_c(c, y) = F_i(L_j(x), F_j(x, y))$ yielding

$$a_c = a_i a_j, \tag{24}$$

$$b_c = a_i b_j + b_i, \tag{25}$$

$$\alpha_c = \alpha_i \alpha_j, \tag{26}$$

$$q_c(x) = \alpha_i q_j(x) + q_i(a_j x + b_j). \tag{27}$$

By using the binomial expansion of $(ax + b)^n$ we note that

$$q(ax + b) = \sum_{m=0}^{n} q_m \sum_{k=0}^{m} \binom{m}{k} a^{m-k} b^k x^{m-k} \tag{28}$$

where $q(x) = \sum_{m=0}^{n} q_m x^m$. From this we can compute $p(x) = q(ax+b)$ by summing coefficients for x^{m-j} :

$p_0, \ldots, p_m \leftarrow 0$

for $m = 0$ to n **do**

  for $k = 0$ to m **do**

    $p_{m-k} \leftarrow p_{m-k} + q_m \binom{m}{k} a^{m-k} b^k$

  end for

end for

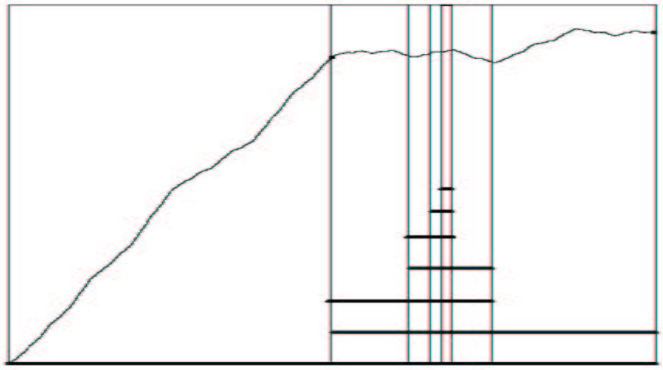

Figure 2: The horizontal contractions converge on the evaluation point. The contractions shown correspond to the map $L_2 \circ L_1 \circ L_2 \circ L_1 \circ L_2 \circ L_2$. The composition of corresponding vertical contractions yields the result of the fractal function evaluation

Each of the terms a^i and b^j can be recomputed and $\binom{m}{j}$ can be retrieved via lookup into a table referencing Pascal's triangle. The rapid convergence of Algorithm 2 is apparent from Equation 24 and Figure 2. For example, if each of the horizontal contraction factors were $a_j = 1/2$, then $\lceil -log_2 \epsilon \rceil$ iterations would be required (ϵ controls the accuracy of the computation and represents an upper bound on the horizontal contractivity factor a for the composite transformation w formed during iteration).

Algorithm 2 Evaluation of FIF $f(x)$.

Input: $x \in [x_0, x_N],\ \{w_j = (L_j(x), F_j(x,y))\}_{j=1}^{N}$

Output: $y \cong f(x)$

$w \leftarrow (L(x) = x,\ F(x,y) = y)$ {identity transformation}

repeat

$j \leftarrow 1$

while $j < N$ and $x > L \circ L_j(x_N)$ **do**

$j \leftarrow j + 1$

end while

$w \leftarrow w \circ w_j$

until $a \leq \epsilon$ {composite $L(x) = ax + b$ is sufficiently contractive}

$k \leftarrow \lfloor N/2 \rfloor$

$y \leftarrow F(x_k, y_k)$ {transform median control point}

3 Integrating FIF's

In [BEH89] the construction of a FIF $\hat{f}(x)$ that is the definite integral of another FIF $f(x)$ on the interval $[x_0, x]$ is given. In other words, the maps $\{(L_j(x), \hat{F}_j(x,y))\}_{j=1}^N$ for

$$\hat{f}(x) = \hat{y}_0 + \int_{x_0}^{x} f(t)dt, \tag{29}$$

can be computed from the maps $\{(L_j(x), F_j(x,y))\}_{j=1}^N$ for $f(x)$. Given the initial value $\hat{y}_0$ we can determine the last interpolation value as

$$\hat{y}_N = \hat{y}_0 + \frac{\sum_{n=1}^{N} a_n \int_{x_0}^{x_N} q_n(t)dt}{1 - \sum_{n=1}^{N} a_n \alpha_n}. \tag{30}$$

Note that the denominator in Equation 30 can never be zero since $\sum a_n = 1$ and $|\alpha_n| < 1$. The remaining knot y-values are

$$\hat{y}_j = \hat{y}_0 + \sum_{n=1}^{j} a_n \left[\alpha_n(\hat{y}_N - \hat{y}_0) + \int_{x_0}^{x} q_n(t)dt \right]. \tag{31}$$

Both sets of maps for $f(x)$ and $\hat{f}(x)$ share each L_j while $\hat{F}_j$ is now determined by

$$\hat{F}_j(x,y) = a_j \alpha_j y + \hat{q}_j(x), \tag{32}$$

where

$$\hat{q}_j(x) = \hat{y}_{j-1} - a_j \alpha_j \hat{y}_0 + a_j \int_{x_0}^{x} q_j(t)dt. \tag{33}$$

Equations 30 and 31 can be derived by summing the integrals over each partition and using the substitution provided by Equation 17. By using the definition of $\hat{f}(x)$ we can determine an equation for $\hat{f}(L_j(x))$ and then Equations 32 and 33 follow from Equation 17. The proof for this construction was first given in [BEH89] (for a more detailed proof see [Coc98]).

Note that the vertical contractivity factor for $\hat{F}_j$ in Equation 32 is $a_j \alpha_j$ which is smaller in magnitude than the vertical contractivity factor α_j for F_j. As this term approaches zero the function $\hat{F}_j$ can be approximated by $\hat{q}_j(x)$. Intuitively, this demonstrates the "smoothing" of the fractal function as we integrate.

The FIF model is general enough to include many "smooth" functions (*e.g.*, we can represent polynomials). Since we have a closed form solution $\int_{x_0}^{x} f(x)dx$ for any FIF $f(x)$ implies that we can create a wide variety of differentiable FIF's according to the funda-

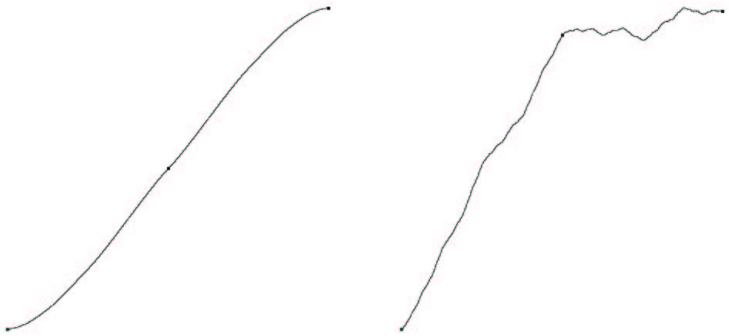

Figure 3: The integrals of the fractal interpolation functions f (left) and g (right) from Figure 1. Both curves are C^1 and fractal.

mental theorem of calculus. More precisely, $\hat{f}'(x) = f(x)$ if and only if $\hat{f}$ is the function associated with $\{(L_j(x), \hat{F}_j(x,y))\}_{j=1}^N$ where

$$\hat{F}_j(x,y) = \hat{\alpha}_j y + \hat{q}_j(x), \tag{34}$$

$$\hat{\alpha}_j / a_j = \alpha_j, \tag{35}$$

$$\hat{q}_j'(x) = a_j q_j(x), \quad j = 1, \ldots, N. \tag{36}$$

The integral of $f(x)$ is a fractal function interpolating (0,0), (1,1), (2,2), using the same horizontal contractions as f, with the new vertical contractions

$$F_1(x,y) = \frac{1}{4}y + \frac{1}{8}x^2 \tag{37}$$

$$F_2(x,y) = \frac{1}{4}y + 1 + \frac{1}{2}x - \frac{1}{8}x^2 \tag{38}$$

The integral of g interpolates (0,0), $(1,\frac{37}{40})$, (2,1) with the new vertical contractions

$$G_1(x,y) = \frac{17}{40}y + \frac{1}{8}x^2 \tag{39}$$

$$G_2(x,y) = -\frac{17}{40}y + \frac{37}{40} + \frac{1}{2}x - \frac{1}{8}x^2 \tag{40}$$

The graphs of these integral curves are shown in Figure 3.

3.1 The FIF Tensor Product

The non-differentiable property of most fractal curves and surfaces created with affine transformations can be problematic in computer graphics especially when it comes to computing surface normals for lighting calculations. Typically, surface normals

are approximated by averaging normals of bounding volume hierarchies (as in [HD91]) or low-pass filtering the surface at some prescribed resolution and inferring a normal from the low resolution result. Surfaces created from differentiable FIF's (*e.g.,* via tensor products) are both fractal and have a well defined surface normal.

We can create a FIF *height field* $h(x, y)$ that has well defined surface normals by choosing differentiable FIF's $f(x)$ and $g(x)$ and defining the tensor product

$$h(x, y) = f(x)g(y), \quad x_0 \leq x \leq x_N, \; y_0 \leq y \leq y_M. \tag{41}$$

Using the partial derivatives

$$h_x(x, y) = f'(x)g(y) \tag{42}$$

$$h_y(x, y) = f(x)g'(y) \tag{43}$$

the tangent plane at $h(x, y)$ is spanned by the vectors $(1, 0, h_x(x, y))$ and $(0, 1, h_y(x, y))$ and the normal direction $\mathbf{N}(x, y)$ can be com-

puted via the cross product of these two vectors:

$$\mathbf{N}(x,y) = (-h_x(x,y), -h_y(x,y), 1) \quad (44)$$

$$= (-f'(x)g(y), -f(x)g'(y), 1). \quad (45)$$

4 Ray Intersection

In order to ray trace the FIF tensor product height field $h(x,y)$ we construct an algorithm to find the intersection of a ray with the surface $(x,y,h(x,y))$ for $(x,y) \in [x_0, x_N] \times [y_0, y_M]$. The ray $\mathbf{r}(t)$ is defined by the *ray origin* $\mathbf{o}$ and the *ray direction* vector $\mathbf{d}$ as

$$\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}, 0 \leq t < \infty \quad (46)$$

.

Following previous fractal terrain [Kaj83, Bou85] and IFS [HD91] ray tracing algorithms, we use a depth first search of a hierarchy of procedural bounding volumes that converge to the surface, as illustrated in Figure 4. For heightfields we can impose an ordering

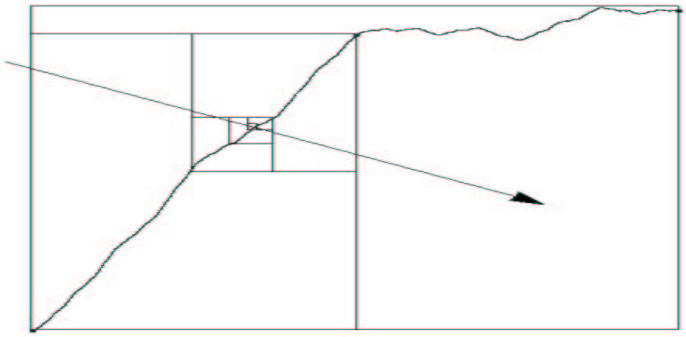

Figure 4: A 2-D illustration of the procedurally-generated bounding volumes used to determine ray-FIF intersection.

on the bounding boxes we inspect in such a way that the first intersection we find is indeed the closest intersection.

In order to construct our bounding volumes, we need the bounds for the functions $f(x)$ and $g(y)$ on the intervals $[x_0, x_N]$ and $[y_0, y_M]$ respectively. It is difficult in general to find the optimal bounds of an attractor [Ric96]. One suboptimal but reasonable solution is to use the chaos game (see Algorithm 1) to find approximate bounds and inflate these bounds by some small factor.

Given the corners (x_0, f_{min}) and (x_N, f_{max}) of the bounding rect-

angle containing the graph of $f(x)$, we need to be able to construct a smaller bounding rectangle that bounds the attractor after it has passed through some contraction $w = (L(x), F(x, y))$. Clearly the bounds on two of the sides will be $x_{min} = L(x_0)$ and $x_{max} = L(x_N)$. To find the other two sides we compute the extrema of the polynomials

$$F(x, f_{min}) = \alpha f_{min} + q(x), \tag{47}$$

$$F(x, f_{max}) = \alpha f_{max} + q(x) \tag{48}$$

on the interval $x \in [x_0, x_N]$.

We construct a bounding volume (3-D oriented box) for the portion of the surface $(x, y, f(x)g(y))$ corresponding to the two bounding rectangles with corners (x_{min}, f_{min}), (x_{max}, f_{max}) and (y_{min}, g_{min}), (y_{max}, g_{max}). The corners of this bounding box are $(x_{min}, y_{min}, h_{min})$ and $(x_{max}, y_{max}, h_{max})$ where h_{min} and h_{max} are the minimum and maximum of the set

$$\mathcal{S} = \{f_{min}g_{min},\ f_{min}g_{max},\ f_{max}g_{min},\ f_{max}g_{max}\}. \tag{49}$$

We can find the points of intersection $(\mathbf{r}(t_0), \mathbf{r}(t_1), t_0 < t_1)$ be-

tween the ray and the bounding box by finding the ray intersections with each pair of parallel planes defining the box. Given the six intersection t values for each plane the endpoints of the interval

$$[t_0, t_1] = [t_{x_{min}}, t_{x_{max}}] \cap [t_{y_{min}}, t_{y_{max}}] \cap [t_{h_{min}}, t_{h_{max}}] \tag{50}$$

yield the appropriate values. If this interval is empty or $t_1 < 0$ then the ray does not intersect the box. Since these planes are parallel to the coordinate axes the intersection computation is simple. For example, the value of t at the intersection of the ray $\mathbf{r}(t)$ with the plane $x = x_0$ is

$$t = \frac{x_0 - \mathbf{o}_x}{\mathbf{d}_x} \tag{51}$$

In the special case where the ray is parallel to one of the planes then its corresponding intersection interval becomes either $(-\infty, +\infty)$ or empty depending on whether or not the ray origin is between the planes in question.

The method for finding a ray intersection with the surface first attempts to intersect the ray with the surface's global bounding

volume as described above. If the ray misses this box then we conclude that the ray does not intersect the surface. Otherwise we recursively generate child bounding boxes and carry on until either we have concluded that there is no intersection or we have intersected a very small box. As we noted earlier, we can choose the order in which we generate these child bounding boxes to ensure that the first intersection we encounter is indeed the closest intersection to the ray origin. This ordering is determined by projecting the ray onto the xy-plane and classifying its direction into one of four cases: $((+x,+y), (+x,-y), (-x,+y), (-x,-y))$. Figure 5 illustrates each case and shows possible orderings for six child bounding boxes. Note that no child box can occlude another child box with a lower ordering value.

An outline of the recursive search through the bounding boxes is given in Algorithm 3. Given the ray $\mathbf{r}(t)$, the maps and bounding rectangles for f and g, the child ordering list for the current ray direction, the routine generates composite maps w^g and w^f (which are each initialized to the identity transformation

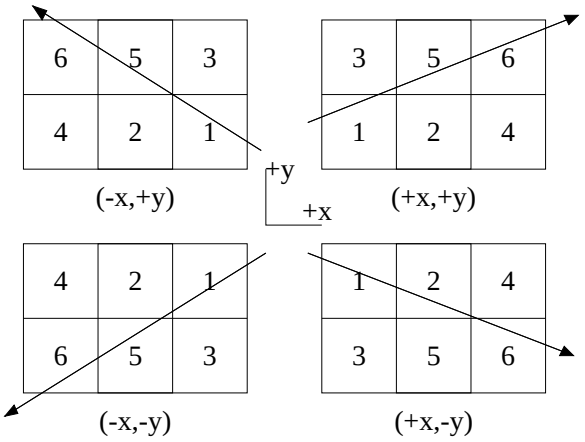

Figure 5: Four cases of ordering child bounding boxes based on the direction of the ray projected on to the xy-plane.

$(L(x) = x, F(x, y) = y)))$ that produce a bounding box that is small enough to approximate the intersection point.

5 Results

Figure 6 illustrates the rendering of fractal surfaces with fractal surfaces with well-defined surface normals. Although the surface on the left appears smooth, it is fractal and its roughness is subtle and detectable in its shading. The curve of the derivative f can

Algorithm 3 (**rayIntersection**) Find ray intersection with height field $h(x, y) = f(x)g(y)$ defined by FIF's f and g.

Input: $\mathbf{r}(t)$, $\{w_j^f\}_{j=1}^N$, $\{w_j^g\}_{j=1}^M$, $rect_f$, $rect_g$, *childOrdering*, w^g, w^f

Output: t : ray intersection parameter, $t < 0$ implies no intersection

{Form 2D bounding rectangles that contain the attractors for f and g
once they have passed through their respective composite transformations w^f and w^g.}
$rect'_f \leftarrow$ **childBoundingRectangle**$(w^f(rect_f))$
$rect'_g \leftarrow$ **childBoundingRectangle**$(w^g(rect_g))$

{Build the resulting bounding volume for the corresponding portion of the surface $h(x, y)$.}
$box \leftarrow$ **boundingVolume**$(rect'_f, rect'_g)$

if ray $\mathbf{r}(t)$ intersects *box* **then**
  Set ray entry and exit parameters $t_0 < t_1$.
  if composite transformations are sufficiently contractive **then**
    $t \leftarrow (t_0 + t_1)/2$ {Approximate intersection found, use average t-values.}
  else
    for $(i, j) \in$ *childOrdering* **do**
      $w_c^f \leftarrow w^f \circ w_i^f$
      $w_c^g \leftarrow w^g \circ w_j^g$
      $t \leftarrow$ **rayIntersection**$(\ldots, w_c^f, w_c^g)$ {recurse with new maps}
      if $t > 0$ **then**
        Intersection found, unwind recursion.
      end if
    end for
  end if
else
  $t \leftarrow -1$ {no intersection}
end if

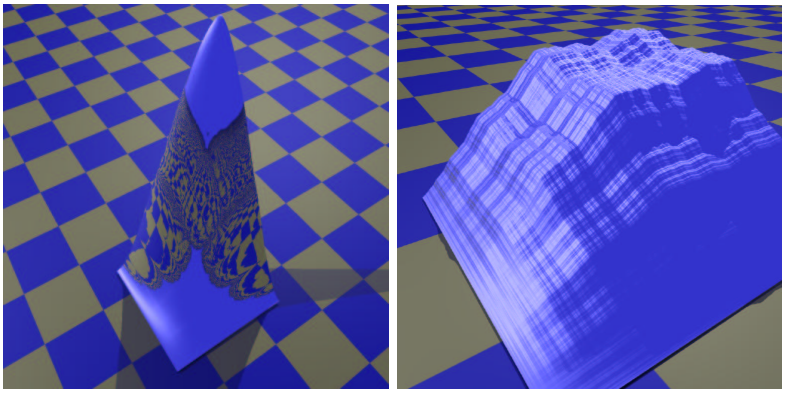

Figure 6: FIF tensor product surfaces: $f \times f$ (left) and $g \times g$ (right), rendered as reflective mirrors.

be visualized in the reflection of straight lines.

We have adapted a conventional raytracer originally developed for [Lew90] to include FIF objects along with the usual polygons and implicit surfaces. To specify the relation between incident light and reflected radiance L_r, the user can assign to a FIF object one of several illumination models: diffuse, Phong, Torrance-Sparrow, and others[1]. FIF objects may also be transparent and have mirror reflection.

[1] See [Lew94] for additional details.

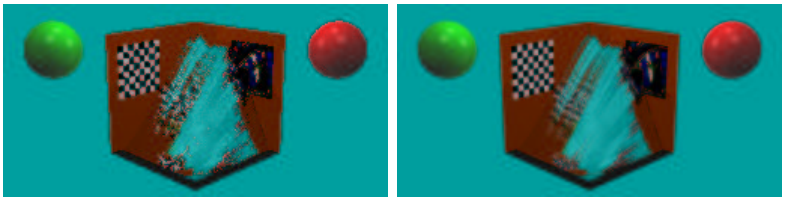

Figure 7: A FIF object rendered by a conventional raytracer. Left: one sample per pixel. Right: 16 jittered samples per pixel. The FIF object filling the corner includes a mirror reflection component.

As with any raytracer, it is important to consider the sampling process. Each pixel value may be considered the integral of the reflected radiance L_r times a weighting function $w(u,v)$ over a sample area (which may or may not extend outside the pixel):

$$L_{\text{pixel}} = \int\int_{\text{area}} L_r(u,v)\, w(u,v)\, du\, dv \tag{52}$$

where $w(u,v)$ is defined so that $\int\int_{\text{area}} w(u,v)\, du\, dv = 1$. A ray tracer may take a single ray per pixel so that $w(u,v) = \delta(u-u_c)\,\delta(v-v_c)$, where (u_c, v_c) is the center of the pixel. Alternatively, it may cast $N \times N$ rays per pixel, so that

$$w(u,v) = \sum_{i=0}^{N-1}\sum_{j=0}^{N-1} \delta(u-u_i)\,\delta(v-v_j) \tag{53}$$

where the (u_i, v_j) values may be on a uniform or randomly-"jittered" grid. It is also possible to incorporate a Gaussian dropoff into $w(u, v)$.

Figure 7 shows two images created with this raytracer. The left-hand image uses a single ray per pixel and the right-hand image uses 4×4 jittered rays per pixel. The differences between the two demonstrate the increased susceptibility of FIF objects to aliasing effects.

6 Conclusion

While not all fractal functions are differentiable, they are all integratable, and their integral is often itself a fractal function that is therefore differentiable. We have used this property to define a class of fractal surfaces with a well-defined surface normal by creating a tensor product of integrals of fractal functions.

6.1 Future Work

The tensor product is in itself not an acceptable terrain model. All of the examples we have constructed have obvious grain lines parallel to the coordinate axes. These grain lines can be eliminated by creating a non-separable fractal function height field, similar to the non-separable fractal function curves in [CHF98]. A non-separable fractal height field is made from smaller self-replicas, but cannot be separated into a tensor product of two individual self-similar fractal functions.

Removing the tensor-product artifacts will also produce a more realistic rough surface model. The surface normals of a sample area of a given fractal surface can be collected into a normal distribution function and/or a BRDF. A fractal BRDF can be used to shade arbitrary geometries as rough surfaces.

Aliasing is a problem, especially for fractal surfaces. While the normals of our fractal surfaces are well defined and continuous, they vary widely over small regions and are in fact fractal. These

normals could be sampled analytically by integrating over a region that projects to a pixel, as this integration is similar to the integration performed to generate the geometry of the surface. This integration method could be used to construct a normal distribution for regions on the surface, and could also yield a BRDF model for rough surfaces.

Fractal functions with the same horizontal and vertical scale factors are closed under addition and scalar multiplication. We suspect this property could be used to define a fractal function of the height field normal over a given domain. This normal in fractal function form could then be integrated to produce an area average of the surface normals, and perhaps even an area sample of the shading.

Height fields are an explicit geometric representation. We are pursuing methods for using fractal functions for implicit and parametric geometric representations as well. If these methods use fractal functions that are integrals of other fractal functions,

then the geometry they represent will also be shaded as directly as the height fields in this paper.

6.2 Acknowledgments

This research was funded in part by the National Science Foundation under grant #CCR-9529809 and a gift from Intel.

References

[Bar86] Michael F. Barnsley. Fractal functions and interpolation. *Constructive Approximation*, 2:303–329, 1986.

[BEH89] Michael F. Barnsley, John H. Elton, and D. P. Hardin. Recurrent iterated function systems. *Constructive Approximation*, 5:3–31, 1989.

[BH89] Michael F. Barnsley and A. N. Harrington. The calculus of fractal interpolation functions. *Journal of Apoximation Theory*, 57:14–34, 1989.

[Bou85] Christian Bouville. Bounding ellipsoids for ray-fractal intersection. *Computer Graphics*, 19(3):45–51, 1985.

[CHF98] Wayne O. Cochran, John C. Hart, and Patrick J. Flynn. On approximating rough curves with fractal functions. In *Proc.* Graphics Interface, pages 65–72, June 1998.

[Coc98] Wayne O. Cochran. *A Recurrent Modeling Toolset.* PhD thesis, EECS Dept, Washington State University, December 1998.

[FFC82] Alain Fournier, Donald Fussel, and Loren Carpenter. Computer rendering of stochastic models. *Communications of the ACM*, 25(6):371–384, June 1982.

[Fou] Alain Fournier. personal communication.

[HD91] John C. Hart and Thomas A. DeFanti. Efficient antialiased rendering of 3-D linear fractals. *Computer Graphics*, 25(3):91–100, 1991.

[Kaj83] James T. Kajiya. New techniques for ray tracing procedurally defined objects. *ACM Transactions on Graphics*, 2(3):161–181, 1983. Also appeared in *Computer Graphics 17,* 3 (1983), 91–102.

[Lew90] Robert R. Lewis. Three-dimensional texturing using lattices. *Eurographics '90*, pages 439–48, September 1990.

[Lew94] Robert R. Lewis. Making shaders more physically plausible. *Computer Graphics Forum*, 13(2):109 – 120, June 1994.

[Man82] Benoit B. Mandelbrot. *The Fractal Geometry of Nature.* W.H. Freeman, San Francisco, 1982.

[Mil86] Gavin S. P. Miller. The definition and rendering of terrain maps. *Computer Graphics*, 20(4):39–48, Aug. 1986.

[MKM89] F. Kenton Musgrave, Craig E. Kolb, and Robert S. Mace. The synthesis and rendering of eroded fractal terrains. *Computer Graphics*, 23(3):41–50, July 1989.

[Ric96] Jonathan Rice. Spatial bounding of self-affine iterated function system attractor sets. In *Proc.* Graphics Interface '96, pages 107–115, May 1996.

List of Figures