Chapter 2

Algorithms for random fractals

Dietmar Saupe

2.1 Introduction

For about 200 years now mathematicians have developed the theory of smooth curves and surfaces in two, three or higher dimensions. These are curves and surfaces that globally may have a very complicated structure but in small neighborhoods they are just straight lines or planes. The discipline that deals with these objects is differential geometry. It is one of the most evolved and fascinating subjects in mathematics. On the other hand fractals feature just the opposite of smoothness. While the smooth objects do not yield any more detail on smaller scales a fractal possesses infinite detail at all scales no matter how small they are. The fascination that surrounds fractals has two roots: Fractals are very suitable to simulate many natural phenomena. Stunning pictures have already been produced, and it will not take very long until an uninitiated observer will no longer be able to tell whether a given scene is natural or just computer simulated. The other reason is that fractals are simple to generate on computers. In order to generate a fractal one does not have to be an expert of an involved theory such as calculus, which is necessary for differential geometry. More importantly, the complexity of a fractal, when measured in terms of the length of the shortest computer program that can generate it, is very small.

Fractals come in two major variations. There are those that are composed of several scaled down and rotated copies of itself such as the von Koch snowflake curve or the Sierpinski gaskets. Julia sets also fall into this general category, in the sense that the whole set can be obtained by applying a nonlinear iterated map to an arbitrarily small section of it (see Chapter 3). Thus, the structure of Julia sets is already contained in any small fraction. All these fractals can be termed *deterministic fractals*. Their computer graphical generation requires use of a particular mapping or rule which then is repeated over and over in a usually recursive scheme.

In the fractals discussed here the additional element of randomness is included, allowing simulation of natural phenomena. Thus we call these fractals *random fractals*.

Given that fractals have infinite detail at all scales it follows that a complete computation of a fractal is impossible. So approximations of fractals down to some finite precision have to suffice. The desired level of resolution is naturally given by constraints such as the numbers of pixels of the available graphics display or the amount of compute time that one is willing to spend. The algorithms presented below fall into several categories.

(C1) An approximation of a random fractal with some resolution is used as input and the algorithm produces an improved approximation with resolution increased by a certain factor. This process is repeated with outputs used as new inputs until the desired resolution is achieved. In some cases the procedure can be formulated as a recursion. The *midpoint displacement methods* are examples.

(C2) Only one approximation of a random fractal is computed namely for the final resolution. Pictures for different resolutions are possible but require that most of the computations have to be redone. The *Fourier filtering method* is an example.

(C3) In the third approach the approximation of a fractal is obtained via an iteration. After each step the approximation is somewhat improved, but the spatial resolution does not necessarily increase by a constant factor in each iteration. Here the allowed compute time determines the quality of the result. The *random cut method* described below or the computation of Julia sets as preimages of repellers are examples.

Almost all of the algorithms contained in this paper are based on methods and ideas discussed in Chapter 1 as well as in [103] and ultimately in [68], where also some background information and further references can be found. Please note also the following two disclaimers:

It is not the intent of this chapter to give a comprehensive account of the theory that leads to the algorithms. Instead we have to direct the interested reader to Chapter 1 and the references therein. Secondly, we describe only a few of the aspects of the rendering of fractals. It must be acknowledged that the generation of data of a fractal alone is not sufficient to obtain a good picture. It is likely that computer programmers will spend far more time writing software that can render e.g. a fractal landscape with some reasonable coloring and shading (if that work has not already been done) than for the coding of fractal data generation routines. In any case the computer will surely use much more CPU time for rendering than for data generation. This is a good reason not to be satisfied with poor approximations of fractals due to insufficient algorithms.

In the following sections we describe a number of algorithms with increasing complexity. They are documented in the form of pseudo code which we hope is self-explanatory. This code is included to clarify the methods and therefore is not optimized for speed. Also, it is not complete. Some routines such as those for fast Fourier transformation and multilinear interpolation have been omitted. If they are not available in the form of library routines, they can easily be written using the indicated literature.

In Section 2.7 we explain two methods to graphically display fractal surfaces:

a. flat two-dimensional top view and parallel projection for three-dimensional view using color mapped elevation, Gouraud shaded polygons, and the painters algorithm for hidden surface elimination.

b. three-dimensional perspective representation with points as primitives, using an extended floating horizon method.

In Section 2.8 we list a number of definitions from probability theory and related to random processes. Notions such as "random variables", "correlation" etc. are marked with a dagger (†) where they first appear in the main text, and they are then briefly explained in Section 2.8 for the convenience of those readers who would like to remind themselves about these terms.

2.2 First case study: One-dimensional Brownian motion

2.2.1 Definitions

Brownian motion in one variable constitutes the simplest random fractal, and also it is at the heart of all the following generalizations. Small particles of solid matter suspended in a liquid can be seen under a microscope to move about in an irregular and erratic way. This was observed by the botanist R. Brown around 1827. The modeling of this movement is one of the great topics of statistical mechanics (see e.g. [90]). Occasionally Brownian motion is also referred to as "brown noise". In one dimension we obtain a random process† $X(t)$, i.e. a function X of a real variable t (time) whose values are random variables† $X(t_1), X(t_2)$, etc.

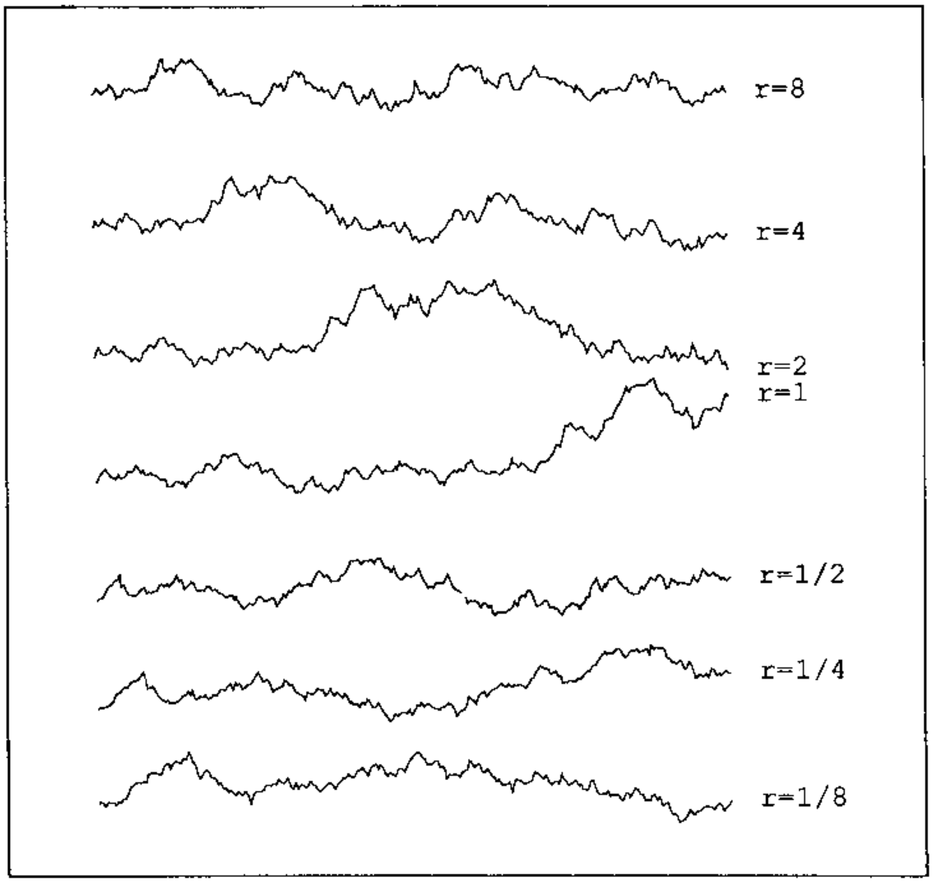


Fig. 2.1: Properly rescaled Brownian motion. The graph in the center shows a small section of Brownian motion $X(t)$. In the other graphs the properly rescaled random functions of the form $r^{-\frac{1}{2}}X(rt)$ are displayed. The scaling factor r ranges from $r = \frac{1}{8}$ to $r = 8$ corresponding to expanding and contracting the original function in the time direction. Note, that the visual appearance of all samples is the same.

The increment

$$X(t_2) - X(t_1) \text{ has Gaussian distribution†} \tag{2.1}$$

and the mean square increments† have a variance proportional to the time differences, thus

$$E\left[|X(t_2) - X(t_1)|^2\right] \propto |t_2 - t_1|. \tag{2.2}$$

Here E denotes the mathematical expectation† of a random variable, or, in other words, the average over many samples. We say that the increments of X are *statistically self-similar* in the sense that

$$X(t_0 + t) - X(t_0) \quad \text{and} \quad \frac{1}{\sqrt{r}}(X(t_0 + rt) - X(t_0))$$

have the same finite dimensional joint distribution† functions for any t_0 and $r > 0$. If we take for convenience $t_0 = 0$ and $X(t_0) = 0$ then this means that the two random functions

$$X(t) \text{ and } \frac{1}{\sqrt{r}}X(rt)$$

are statistically indistinguishable. The second one is just a *properly rescaled* version of the first. Thus, if we accelerate the process $X(t)$ by a factor of 16, for example, then we can divide $X(16\,t)$ by 4 to obtain the same Brownian motion that we started with. We will return to this important characterization, when we discuss fractional Brownian motion, and it will be crucial in understanding the spectral synthesis method.

2.2.2 Integrating white noise

The integral of uncorrelated white Gaussian noise W satisfies (2.1) and (2.2):

$$X(t) = \int_{-\infty}^{t} W(s)ds. \tag{2.3}$$

The random variables $W(t)$ are uncorrelated† and have the same normal distribution† $N(0,1)$. Moreover, the graph of a sample of Brownian motion $X(t)$ has a fractal dimension of 1.5, and the intersection of the graph with a horizontal line has a dimension of 0.5 . Formula (2.3) gives rise to the first little algorithm *WhiteNoiseBM()*.

```
ALGORITHM WhiteNoiseBM (X, N, seed)
Title        Brownian motion by integration of white Gaussian noise

Arguments    X[]        array of reals of size N
             N          size of array X
             seed       seed value for random number generator
Variables    i          integer

BEGIN
    X[0] := 0
    InitGauss (seed)
    FOR i := 1 TO N-1 DO
        X[i] := X[i-1] + Gauss () / (N-1)
    END FOR
END
```

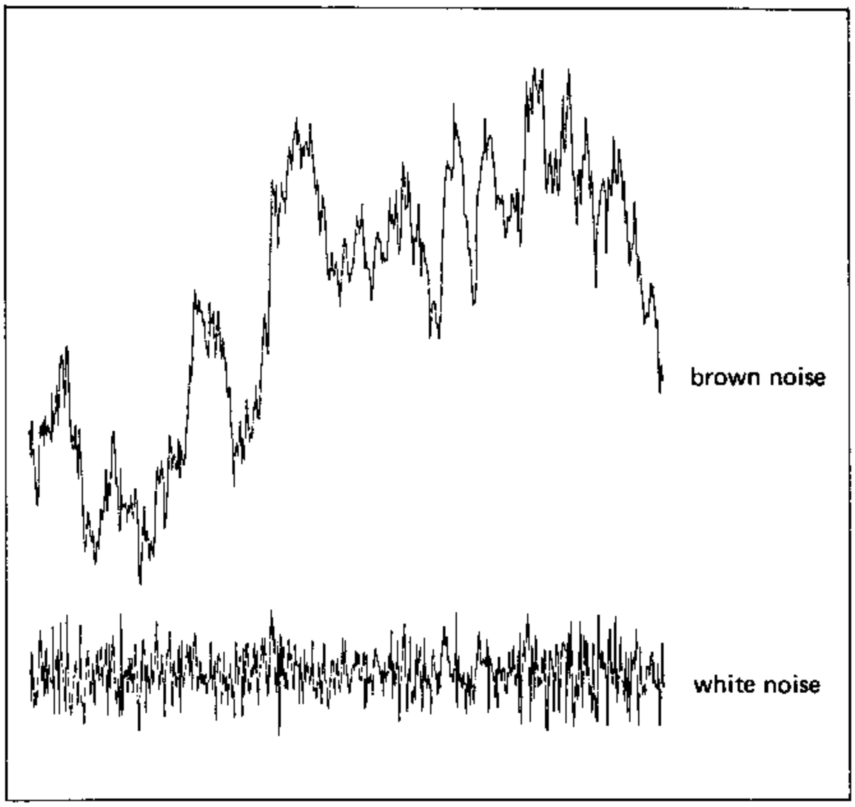


Fig. 2.2: Brownian motion in one dimension. The lower graph shows a sample of Gaussian white noise, its time integral yields the brown noise above it (see algorithm *WhiteNoiseBM()*).

2.2.3 Generating Gaussian random numbers

In the above algorithm we perform a simple numerical integration of white Gaussian noise. It requires a routine called *Gauss()*, which returns a sample

```
ALGORITHM InitGauss (seed)
Title          Initialization of random number generators

Arguments      seed          seed value for random number generator
Globals        Arand         rand() returns values between 0 and Arand, system dependent
               Nrand         number of samples of rand() to be taken in Gauss()
               GaussAdd      real parameter for the linear transformation in Gauss()
               GaussFac      real parameter for the linear transformation in Gauss()
Functions      srand()       initialization of system random numbers

BEGIN
    Nrand := 4
    Arand := power (2, 31) - 1
    GaussAdd := sqrt (3 * Nrand)
    GaussFac := 2 * GaussAdd / (Nrand * Arand)
    srand (seed)
END
```

```
ALGORITHM Gauss ()
Title          Function returning Gaussian random number

Globals        Nrand         number of samples of rand() to be taken in Gauss()
               GaussAdd      real parameter for the linear transformation in Gauss()
               GaussFac      real parameter for the linear transformation in Gauss()
Locals         sum           real
               i             integer
Functions      rand()        system function for random numbers

BEGIN
    sum := 0
    FOR i := 1 TO Nrand DO
        sum := sum + rand ()
    END FOR
    RETURN (GaussFac * sum - GaussAdd)
END
```

of a random variable with normal distribution (mean 0 and variance 1). Let us briefly describe an elementary method for generating such numbers. On most machines a pseudo random number generator will be available. So let us assume that we have a routine *rand()*, which returns random numbers uniformly distributed† over some interval $[0, A]$. Typically A will be $2^{31} - 1$ or $2^{15} - 1$. Also we assume that a routine *srand(seed)* exists, which introduces a seed value for *rand()*. Taking certain linearly scaled averages of the values returned by *rand()* will approximate a Gaussian random variable as follows:

A random variable Y is standardized by subtracting its expected value and dividing by its standard deviation†:

$$Z = \frac{Y - E(Y)}{\sqrt{\operatorname{var} Y}}.$$

The Central Limit Theorem states, that if Z_n is the standardized sum of any n identically distributed random variables, then the probability distribution† of Z_n tends to the normal distribution as $n \to \infty$.. Let Y_i be the i-th value returned by *rand()*, $i = 1, \ldots, n$. Then

$$E(Y_i) = \frac{1}{2}A, \text{ var } Y_i = \frac{1}{12}A^2$$

and thus

$$E\left(\sum_{i=1}^{n} Y_i\right) = \frac{n}{2}A, \text{ var}\left(\sum_{i=1}^{n} Y_i\right) = \frac{n}{12}A^2.$$

With these formulas we obtain

$$Z_n = \frac{\sum_{i=1}^{n} Y_i - \frac{n}{2}A}{\sqrt{\frac{n}{12}}A} = \frac{1}{A}\sqrt{\frac{12}{n}}\sum_{i=1}^{n} Y_i - \sqrt{3n}$$

as our approximate Gaussian random variable. In practice $n = 3$ or 4 already yields satisfactory results for our purposes.

Let us remark that there are other Gaussian random number generators that require only one evaluation of *rand()* for each Gaussian random number returned. They use a transformation method, and the underlying distribution is exactly the normal distribution. We refer the interested reader to [8] and [86].

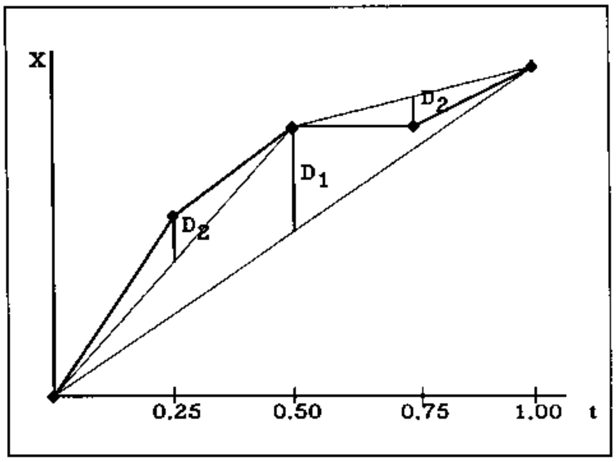


Fig. 2.3: Midpoint displacement.The first two stages in the midpoint displacement technique as explained in the text.

2.2.4 Random midpoint displacement method

Another straightforward method to produce Brownian motion is random midpoint displacement. If the process is to be computed for times, t, between 0 and

```
ALGORITHM MidPointBM (X, maxlevel, sigma, seed)
Title       Brownian motion via midpoint displacement

Arguments   X[]         real array of size $2^{maxlevel} + 1$
            maxlevel    maximal number of recursions
            sigma       initial standard deviation
            seed        seed value for random number generator
Globals     delta[]     array holding standard deviations $\Delta_i$
Variables   i, N        integers

BEGIN
    InitGauss (seed)
    FOR i := 1 TO maxlevel DO
        delta[i] := sigma * power (0.5, (i+1)/2)
    END FOR
    N = power (2, maxlevel)
    X[0] := 0
    X[N] := sigma * Gauss ()
    MidPointRecursion (X, 0, N, 1, maxlevel)
END
```

```
ALGORITHM MidPointRecursion (X, index0, index2, level, maxlevel)
Title       Recursive routine called by MidPointBM()

Arguments   X[]         real array of size $2^{maxlevel} + 1$
            index0      lower index in array
            index2      upper index in array
            level       depth of the recursion
            maxlevel    maximal number of recursions
Globals     delta       array holding standard deviations $\Delta_i$
Variables   index1      midpoint index in array

BEGIN
    index1 := (index0 + index2) / 2
    X[index1] := 0.5 * (X[index0] + X[index2]) + delta[level] * Gauss ()
    IF (level < maxlevel) THEN
        MidPointRecurs (X, index0, index1, level+1, maxlevel)
        MidPointRecurs (X, index1, index2, level+1, maxlevel)
    END IF
END
```

1, then one starts by setting $X(0) = 0$ and selecting $X(1)$ as a sample of a Gaussian random variable with mean 0 and variance σ^2. Then $\text{var}(X(1) - X(0)) = \sigma^2$ and we expect

$$\text{var}(X(t_2) - X(t_1)) = |t_2 - t_1|\sigma^2 \qquad (2.4)$$

for $0 \leq t_1 \leq t_2 \leq 1$. We set $X(\frac{1}{2})$ to be the average of $X(0)$ and $X(1)$ plus some Gaussian random offset D_1 with mean 0 and variance Δ_1^2. Then

$$X(\frac{1}{2}) - X(0) = \frac{1}{2}(X(1) - X(0)) + D_1$$

and thus $X(\frac{1}{2}) - X(0)$ has mean value 0 and the same holds for $X(1) - X(\frac{1}{2})$. Secondly, for (2.4) to be true we must require

$$\mathrm{var}(X(\frac{1}{2}) - X(0)) = \frac{1}{4}\mathrm{var}(X(1) - X(0)) + \Delta_1^2 = \frac{1}{2}\sigma^2.$$

Therefore

$$\Delta_1^2 = \frac{1}{4}\sigma^2.$$

In the next step we proceed in the same fashion setting

$$X(\frac{1}{4}) - X(0) = \frac{1}{2}(X(0) + X(\frac{1}{2})) + D_2$$

and observe that again the increments in X, here $X(\frac{1}{2}) - X(\frac{1}{4})$ and $X(\frac{1}{4}) - X(0)$ are Gaussian and have mean 0. So we must choose the variance Δ_2^2 of D_2 such that

$$\mathrm{var}(X(\frac{1}{4}) - X(0)) = \frac{1}{4}\mathrm{var}(X(\frac{1}{2}) - X(0)) + \Delta_2^2 = \frac{1}{4}\sigma^2$$

holds, i.e.

$$\Delta_2^2 = \frac{1}{8}\sigma^2.$$

We apply the same idea to $X(\frac{3}{4})$ and continue to finer resolutions yielding

$$\Delta_n^2 = \frac{1}{2^{n+1}}\sigma^2$$

as the variance of the displacement D_n. Thus corresponding to time differences $\Delta t = 2^{-n}$ we add a random element of variance $2^{-(n+1)}\sigma^2$ which is proportional to Δt as expected.

2.2.5 Independent jumps

Our last algorithm for Brownian motion falls into the category (C3). We may interpret Brownian motion as the cumulative displacement of a series of independent jumps, i.e. an infinite sum of functions

$$J_i(t) = A_i\beta(t - t_i)$$

where

$$\beta(t) = \begin{cases} 0, & \text{if } t < 0 \\ 1, & \text{if } t \geq 0 \end{cases}$$

and A_i, t_i are random variables with Gaussian and Poisson distributions respectively. This approach generalizes to circles and spheres. For circles we take time as 2π-periodic and arrive at

$$X(t) = \sum_{i=0}^{\infty} A_i\overline{\beta}(t - t_i), \tag{2.5}$$

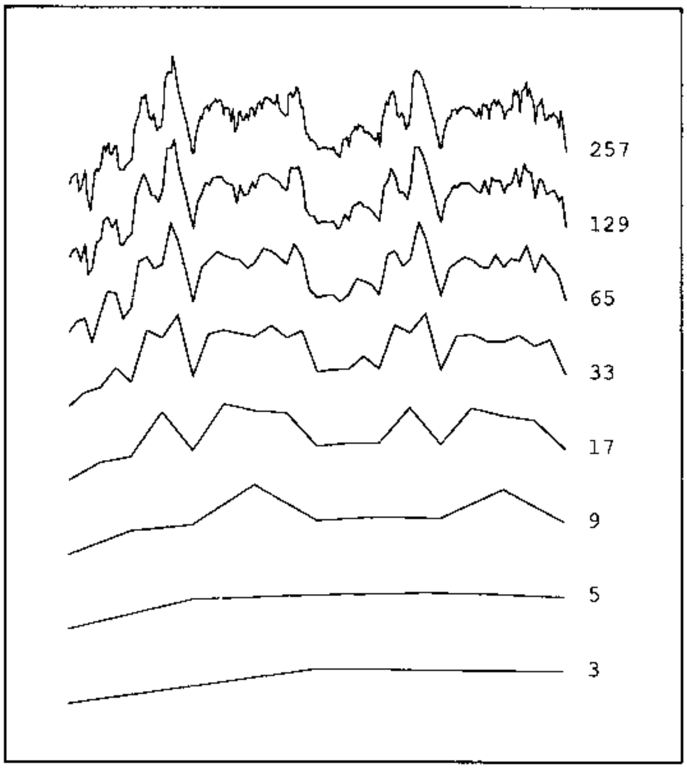


Fig. 2.4: Brownian motion via midpoint displacement method. Eight intermediate stages of the algorithm *MidPointBM()* are shown depicting approximations of Brownian motion using up to 3, 5, 9, ..., 257 points, i.e. the parameter maxlevel was set to 2, 3, ..., 8.

where

$$\overline{\beta}(t) = \begin{cases} 0, \text{if } t \geq \pi (\bmod 2\pi) \\ 1, \text{if } t < \pi (\bmod 2\pi) \end{cases}$$

The A_i are identically distributed Gaussian random variables and the t_i are uniformly distributed random variables with values between 0 and 2π. Each term in the sum (2.5) adds random displacement on half of the circle.

Of course, we can also use midpoint displacement to obtain Brownian motion on a circle. We would just have to require that $X(0) = X(1)$ in order to maintain continuity. However, the midpoint method does not generalize to spheres whereas the random cut method does: In each step a great circle of the sphere is picked at random and one of the two half spheres is displaced by an amount determined by a Gaussian random variable. The pictures of the planets in Figure 1.2 and the Plates 6, 7 and 36 were produced in this way (see also [103,68] and Sections 0.3, 1.1.8).

```
ALGORITHM RandomCutsBM (X, N, maxsteps, seed)
Title       Brownian motion on a circle by random cuts

Arguments   X[]            array of reals of size N
            N              size of array X
            maxsteps       maximal number of displacements
            seed           seed value for random number generator
Globals     Arand          rand() returns values between 0 and Arand
Variables   k,k0,k1,step   integers

BEGIN
    InitGauss (seed)
    FOR k := 0 TO N-1 DO
        X[k] := 0
    END FOR
    FOR step := 1 TO maxsteps DO
        k0 := N * rand () / Arand
        k1 := k0 + N/2 - 1
        FOR k := k0 TO k1 DO
            IF (k < N) THEN
                X[i] := X[i] + Gauss()
            ELSE
                X[k-N] := X[k-N] + Gauss()
            END IF
        END FOR
    END FOR
END
```

2.3 Fractional Brownian motion : Approximation by spatial methods

2.3.1 Definitions

In the last section we studied random processes $X(t)$ with Gaussian increments and

$$\text{var}(X(t_2) - X(t_1)) \propto |t_2 - t_1|^{2H} \qquad (2.6)$$

where $H = \frac{1}{2}$. The generalization to parameters $0 < H < 1$ is called *fractional Brownian motion* (fBm, see [63], [68]). As in the case of ordinary Brownian motion, we say that the increments of X are *statistically self-similar with parameter H*, in other words

$$X(t_0 + t) - X(t_0) \text{ and } \frac{1}{r^H}(X(t_0 + rt) - X(t_0))$$

have the same finite dimensional joint distribution functions for any t_0 and $r > 0$. If we again use for convenience $t_0 = 0$ and $X(t_0) = 0$, the two random functions

$$X(t) \text{ and } \frac{1}{r^H}X(rt) \qquad (2.7)$$

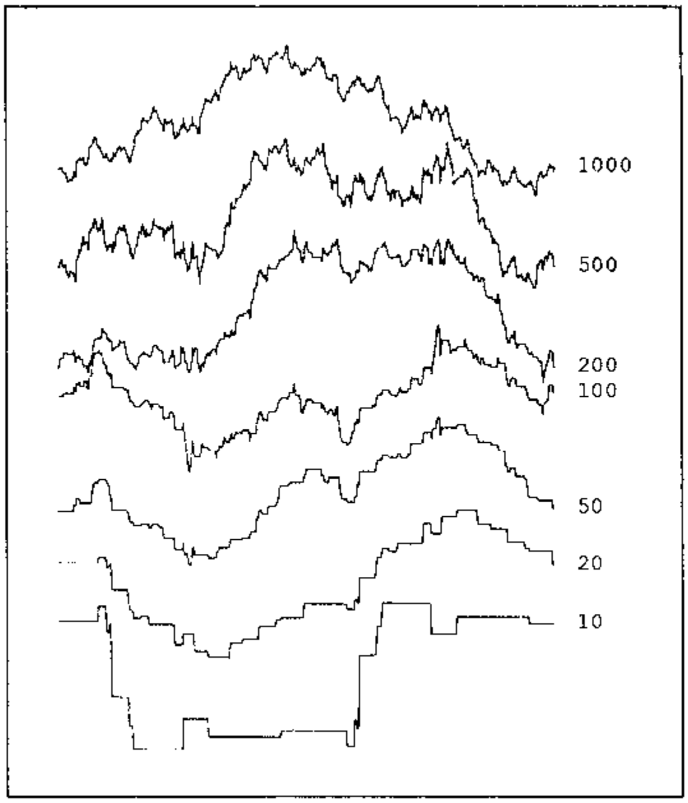


Fig. 2.5: Brownian motion via random cuts. Up to 1000 random cuts have been performed by the algorithm *RandomCutsBM()*. All approximations are step functions which is clearly visible in the lower graphs.

are statistically indistinguishable. Thus "accelerated" fractional Brownian motion $X(rt)$ is *properly rescaled* by dividing amplitudes by r^H.

Let us visualize this important fact (see Figures 2.1 and 2.6). If we set $H = \frac{1}{2}$ we obtain the usual Brownian motion of the last section. For $H = 0$ we get a completely different behavior of X: We can expand or contract the graph of X in the t-direction by any factor, and the process will still "look" the same. This clearly says that the graph of a sample of X must densely fill up a region in the plane. In other words, its fractal dimension is 2. The opposite case is given by the parameter $H = 1$. There we must compensate for an expansion of the graph in the t-direction by also multiplying the amplitudes by the same factor. It is easy to give an argument showing that the fractal dimension of this graph must be $2 - H = 1$. In fact, graphs of samples of fBm have a fractal dimension of $2 - H$ for $0 < H < 1$. The parameter H thus describes the "roughness" of the function at small scales.

Fractional Brownian motion can be divided into three quite distinct cate-

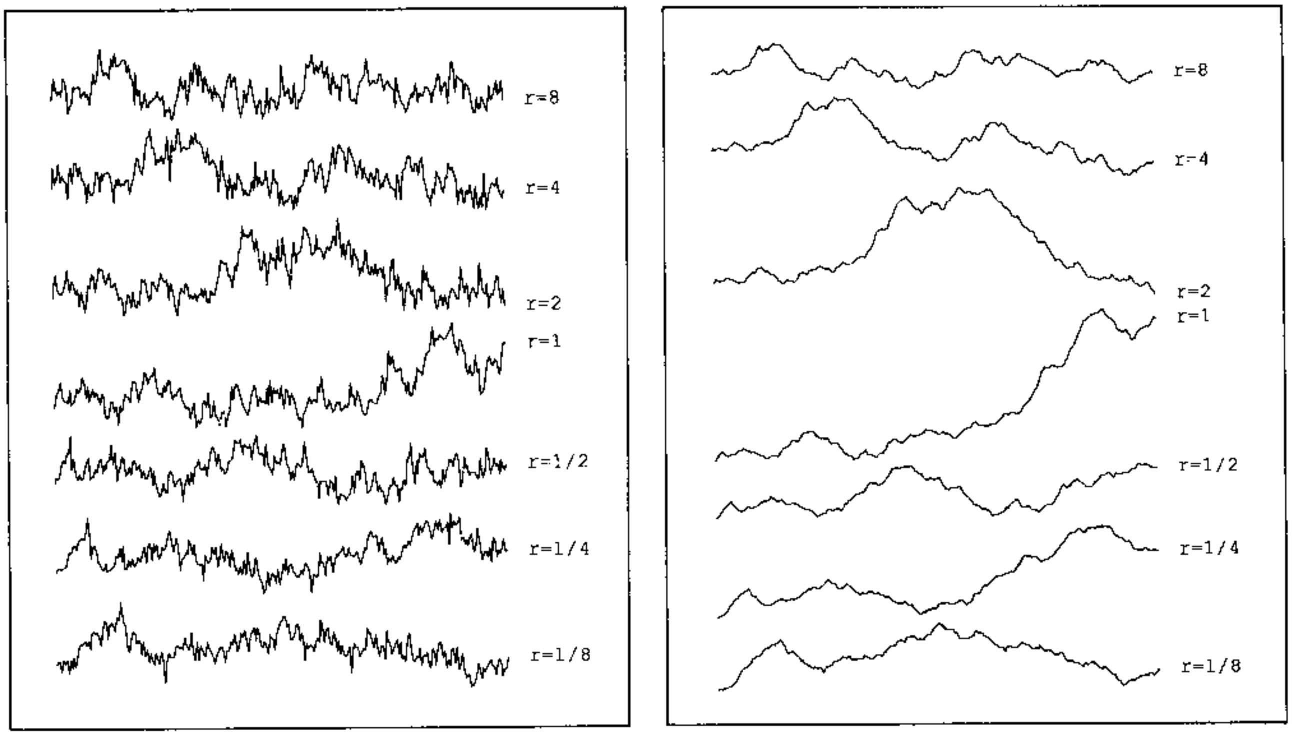


Fig. 2.6: Properly rescaled fractional Brownian motion. The two sets of curves show properly rescaled fractional Brownian motion for parameters $H = 0.2$ (left) and $H = 0.8$ (right). The graphs in the center show small sections of fractional Brownian motion $X(t)$. In the other graphs the properly rescaled random functions $r^{-H}X(rt)$ are displayed. The scaling factor r ranges from $r = \frac{1}{8}$ to $r = 8$ corresponding to expansion and contraction the original function in the time direction. Compare Figure 2.1 for $H = 0.5$.

gories: $H < \frac{1}{2}$, $H = \frac{1}{2}$ and $H > \frac{1}{2}$. The case $H = \frac{1}{2}$ is the ordinary Brownian motion which has independent increments, i.e. $X(t_2) - X(t_1)$ and $X(t_3) - X(t_2)$ with $t_1 < t_2 < t_3$ are independent in the sense of probability theory, their correlation† is 0. For $H > \frac{1}{2}$ there is a positive correlation between these increments, i.e. if the graph of X is increasing for some t_0, then it tends to continue to increase for $t > t_0$. For $H < \frac{1}{2}$ the opposite is true. There is a negative correlation of increments, and the curves seem to oscillate more erratically.

2.3.2 Midpoint displacement methods

For the approximation of fBm the approach taken by the midpoint method can formally be extended to also suit parameters $H \neq \frac{1}{2}$. Here we aim for the equivalent of (2.4)

$$\mathrm{var}(X(t_2) - X(t_1)) = |t_2 - t_1|^{2H}\sigma^2.$$

Using the same line of thought as in the last section, we arrive at midpoint displacements D_n that have variances

$$\Delta_n^2 = \frac{\sigma^2}{(2^n)^{2H}}(1 - 2^{2H-2}). \tag{2.8}$$

Therefore the only changes which are required in the algorithm *MidPointBM()* occur in the computation of Δ_n accommodating $H \neq \frac{1}{2}$.

```
ALGORITHM MidPointFM1D (X, maxlevel, sigma, H, seed)
Title        One-dimensional fractal motion via midpoint displacement

Arguments    X[]         real array of size $2^{maxlevel} + 1$
             maxlevel    maximal number of recursions
             sigma       initial standard deviation
             H           $0 < H < 1$ determines fractal dimension $D = 2 - H$
             seed        seed value for random number generator
Globals      delta[]     array holding standard deviations $\Delta_i$
Variables    i, N        integers

BEGIN
    InitGauss (seed)
    FOR i := 1 TO maxlevel DO
        delta[i] := sigma * power (0.5, i*H) * sqrt (1 - power (2, 2*H-2))
    END FOR
    N = power (2, maxlevel)
    X[0] := 0
    X[N] := sigma * Gauss ()
    MidPointRecursion (X, 0, N, 1, maxlevel)
END
```

It has been shown that the above midpoint displacement technique does not yield true fBm for $H \neq \frac{1}{2}$ [69]. In fact, although

$$\mathrm{var}(X(\frac{1}{2}) - X(0)) = \mathrm{var}(X(1) - X(\frac{1}{2})) = (\frac{1}{2})^{2H}\sigma^2,$$

we do not have

$$\mathrm{var}(X(\frac{3}{4}) - X(\frac{1}{4})) = (\frac{1}{2})^{2H}\sigma^2,$$

as we would like. Thus, this process does not have stationary increments, the times t are not all statistically equivalent. This defect causes the graphs of X to show some visible traces of the first few stages in the recursion. In the two-dimensional extension of this algorithm one obtains results which may exhibit some disturbing creases along straight lines, which are related to the underlying grid of points (compare Section 1.4.3 and the discussion in Appendix A). Nevertheless, this is still a useful algorithm for many purposes. It became most popular after its appearance in [40] and subsequently in some science magazines. Recently it has also been included as a part of some computer graphics text books [49],[50].

One approach to deal with the non-stationarity of the midpoint displacement technique is to interpolate the midpoints in the same way, but then to add a displacement D_n of a suitable variance to all of the points and not just the midpoints. This seems natural, as one would reiterate a measurement at all points in a graph of fBm, when a device is used that allows measurements at a smaller sampling rate Δt and a smaller spatial resolution ΔX. This method is called ***successive random addition.*** The extra amount of work involved as compared with the midpoint method is tolerable, about twice as many displacements are necessary. The actual formula for the variances of the displacements as used in the various stages of this algorithm (see pseudo code) will be derived as a special case of the method dicussed in the next section.

ALGORITHM AdditionsFM1D (X, maxlevel, sigma, H, seed)
Title One-dimensional fractal motion via successive random additions

Arguments	X[]	real array of size $2^{maxlevel} + 1$
	maxlevel	maximal number of recursions
	sigma	initial standard deviation
	H	$0 < H < 1$ determines fractal dimension $D = 2 - H$
	seed	seed value for random number generator
Globals	delta	array holding standard deviations Δ_i
Variables	i, N, d, D	integers
	level	integer

```
BEGIN
    InitGauss (seed)
    FOR i := 1 TO maxlevel DO
        delta[i] := sigma * power (0.5, i*H) * sqrt (0.5) * sqrt (1 - power (2, 2*H-2))
    END FOR
    N = power (2, maxlevel)
    X[0] := 0
    X[N] := sigma * Gauss ()
    D := N
    d := D / 2
    level := 1
    WHILE (level <= maxlevel) DO
        FOR i := d TO N-d STEP D DO
            X[i] := 0.5 * (X[i-d] + X[i+d])
        END FOR
        FOR i := 0 TO N STEP d DO
            X[i] := X[i] + delta[level] * Gauss ()
        END FOR
        D := D / 2
        d := d / 2
        level := level + 1
    END WHILE
END
```

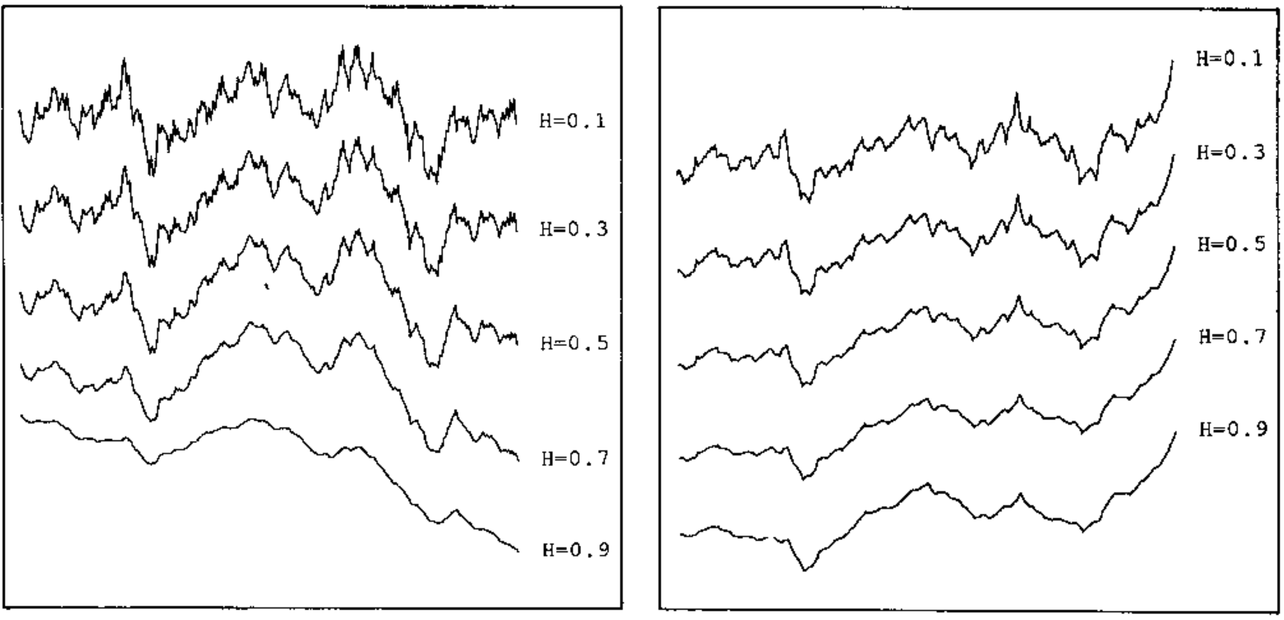


Fig. 2.7: Fractal motion generated by displacement techniques. On the left results of the algorithm *MidPointFM1D()* are shown for various parameters H. On the right the random successive additions were also included. Those curves look smoother because the amplitudes are scaled in order to fit into the graph whereas in the time direction no scaling was done. The effect is especially drastic for $H = 0.1$ where we scaled amplitudes by a factor of 0.7 and thus a properly rescaled graph would have to be contracted by a factor of $0.7^{-0.1} = 35.4$.

2.3.3 Displacing interpolated points

In the midpoint displacement method and the successive random addition method the resolution Δt is improved by a factor of $r = \frac{1}{2}$ in each stage. We can modify the second of the above methods to accommodate other factors $0 < r < 1$. For this purpose one would interpolate $X(t)$ at times $t_i = ir\Delta t$ from the samples one already has from the previous stage at a sampling rate of Δt. Then a random element D_n would be added to all of the interpolated points. In agreement with the requirements for the mean square increments of X we have to prescribe a variance proportional to $(r^n)^{2H}$ for the Gaussian random variable D_n:

$$\Delta_n^2 \propto (r^n)^{2H}. \qquad (2.9)$$

The additional parameter r will change the appearance of the fractal, the feature of the fractal controlled by r has been termed the *lacunarity*. In the algorithm below we use a simple linear interpolation. The reason we include yet another method here is that it generalizes to two, three or more dimensions in a very straight forward way, while the other methods are harder to convert.

ALGORITHM InterpolatedFM1D (X, N, r, sigma, H, seed)
Title One-dimensional Fractal motion with lacunarity

Arguments	X[]	real array of size N
	N	number of elements in X
	r	scaling factor for resolutions ($0 < r < 1$)
	sigma	initial standard deviation
	H	$0 < H < 1$ determines fractal dimension $D = 2 - H$
	seed	seed value for random number generator
Variables	delta	real variable holding standard deviations Δ
	Y[]	real array of size N for the interpolated values of X
	mT, mt	integers, number of elements in arrays X and Y
	t, T	sampling rates for X and Y
	i, index	integer
	h	real

```
BEGIN                         /* initialize the array with 2 points */
    InitGauss (seed)
    X[0] := 0
    X[1] := sigma * Gauss ()
    mT := 2
    T := 1.0
                              /* loop while less than N points in array */
    WHILE (mT < N) DO
                              /* set up new resolution of mt points */
        mt := mT / r
        IF (mt = mT) THEN mt := mT + 1
        IF (mt > N) THEN mt := N
        t := 1 / (mt-1)
                              /* interpolate new points from old points */
        Y[0] := X[0]
        Y[mt-1] := X[mT-1]
        FOR i := 1 TO mt-2 DO
            index := integer (i * t / T)
            h := i * t / T - index
            Y[i] := (1 - h) * X[index] + h * X[index+1]
        END FOR
                              /* compute the standard deviation for offsets */
        delta := sqrt (0.5) * power (t, H) * sigma * sqrt (1.0 - power (t/T, 2-2*H))
                              /* do displacement at all positions */
        FOR i := 0 TO mt-1
            X[i] := Y[i] + delta * Gauss ()
        END FOR
        mT := mt
        T := 1 / mT
    END WHILE
END
```

Following the ideas for midpoint displacement for Brownian motion we set $X(0) = 0$ and select $X(1)$ as a sample of a Gaussian random variable with variance σ^2. Then we can deduce in the same fashion as before that

$$\Delta_n^2 = \frac{1}{2}(1 - r^{2-2H})(r^n)^{2H}\sigma^2 .$$

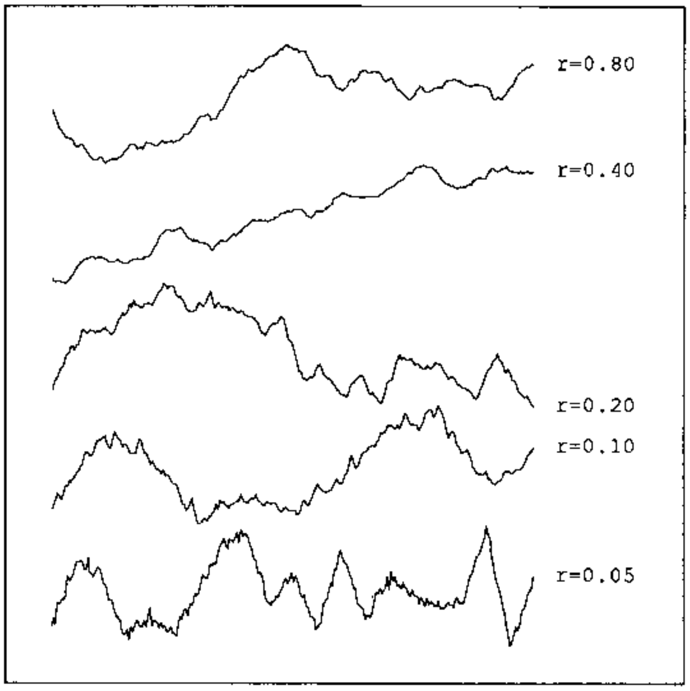


Fig. 2.8: Fractal Brownian motion with varying lacunarity. The algorithm *InterpolatedFM1D()* produced these curves with a parameter $H = 0.8$. The other parameter r is the scaling factor in the method. $r = 0.1$ means that in each stage we have $1/r = 10$ times as many points as in the previous stage. Especially for low r we see a pronounced effect of these high scaling ratios as a modulation remaining from the first one or two stages. This is called the lacunarity.

A little difficulty arises since we cannot work with a continuous variable $X(t)$ in a computer program (X is stored as an array). Thus, generally, one cannot maintain the constant factor of r reducing the resolution Δt in each step. Let us assume for simplicity that we have some $X(0)$ and $X(T)$ already computed where $T \approx r^{n-1}$. Then $X(t)$ with $t \approx r^n$ is first interpolated via

$$X(t) = X(0) + \frac{t}{T}(X(T) - X(0)).$$

Then all points $X(0), X(t), X(2t)$, etc. are displaced by samples of a Gaussian random variable D with variance Δ^2. The old values $X(0)$ and $X(T)$ satisfied

$$\mathrm{var}(X(T) - X(0)) = T^{2H}\sigma^2,$$

and we have to select the variance Δ^2 such that

$$\mathrm{var}(X(t) - X(0)) = t^{2H}\sigma^2$$

holds. Therefore we require

$$t^{2H}\sigma^2 = (\frac{t}{T})^2 T^{2H}\sigma^2 + 2\Delta^2$$

where the first term of the sum stands for the variance already contained in the interpolation, and the other represents the variance due to the perturbations in both $X(0)$ and $X(t)$. Thus

$$\Delta^2 = \frac{1}{2}\sigma^2(1 - (\frac{t}{T})^{2-2H})t^{2H}.$$

To apply this formula to the case of random successive additions we set $T = \frac{1}{2^{n-1}}$ and $t = \frac{1}{2^n}$, and we obtain

$$\Delta^2 = \frac{1}{2}\sigma^2(1 - \frac{1}{2^{2-2H}})\frac{1}{2^{2Hn}}$$

which is, as expected, the quantity Δ_n^2 used in the algorithm *AdditionsFM1D()* where the ratio r is $\frac{1}{2}$.

2.4 Fractional Brownian motion : Approximation by spectral synthesis

2.4.1 The spectral representation of random functions

The spectral synthesis method (also known as the Fourier filtering method) for generating fBm is based on the spectral representation of samples of the process $X(t)$. Since the Fourier transform of X generally is undefined we first restrict $X(t)$ to a finite time interval, say $0 < t < T$:

$$X(t,T) = \begin{cases} X(t) & \text{if } 0 < t < T \\ 0 & \text{otherwise} \end{cases} .$$

We thus have

$$X(t,T) = \int_{-\infty}^{\infty} F(f,T)e^{2\pi i t f}df,$$

where $F(f,T)$ is the Fourier transform of $X(t,T)$

$$F(f,T) = \int_0^T X(t)e^{-2\pi i f t}dt.$$

Now $|F(f,T)|^2 df$ is the contribution to the total energy of $X(t,T)$ from those components with frequencies between f and $f + df$. The average power of X contained in the interval $[0,T]$ is then given by

$$\frac{1}{T}\int_{-\infty}^{\infty} |F(f,T)|^2 df,$$

and the *power spectral density* of $X(t,T)$ is

$$S(f,T) = \frac{1}{T}|F(f,T)|^2.$$

The spectral density of X is then obtained in the limit as $T \to \infty$

$$S(f) = \lim_{T\to\infty} \frac{1}{T}|F(f,T)|^2.$$

The interpretation of $S(f)$ is the following: $S(f)\,df$ is the average of the contribution to the total power from components in $X(t)$ with frequencies between f and $f + df$. $S(f)$ is a nonnegative and even function. We may think of $X(t)$ as being decomposed into a sum of infinitely many sine and cosine terms of frequencies f whose powers (and amplitudes) are determined by the spectral density $S(f)$. For a good introduction into the theory of spectral analysis of random functions we recommend [87] and (shorter) [20]. A version which is more mathematically oriented (but still accessible to readers with some knowledge of probability theory) is contained in Part I of [107].

2.4.2 The spectral exponent β in fractional Brownian motion

The underlying idea of spectral synthesis is that a prescription of the right kind of spectral density $S(f)$ will give rise to fBm with an exponent $0 < H < 1$.

If the random function $X(t)$ contains equal power for all frequencies f, this process is called white noise in analogy with the white light made up of radiations of all wave lengths. If $S(f)$ is proportional to $1/f^2$ we obtain the usual brown noise or Brownian motion. In general, a process $X(t)$ with a spectral density proportional to $1/f^\beta$ corresponds to fBm with $H = \frac{\beta-1}{2}$:

$$S(f) \propto \frac{1}{f^\beta} \sim \text{fBm with } \beta = 2H + 1. \qquad (2.10)$$

Choosing β between 1 and 3 will generate a graph of fBm with a fractal dimension of

$$D_f = 2 - H = \frac{5-\beta}{2}. \qquad (2.11)$$

Let us pause for a moment to explain this relationship between β and H. Mathematically it can be derived from the fact that the mean square increments (which are proportional to Δt^{2H} for fBm with exponent H) are directly related to the autocorrelation function of X, which in turn defines the spectral density by means of a Fourier transform via the Wiener-Khintchine relation (see Section 1.6.8). In place of this "pure" approach we propose a simple and more heuristic argument for the relation $\beta = 2H + 1$. We start out by restating the fundamental property of fBm: If $X(t)$ denotes fBm with exponent $0 < H < 1$ then the properly rescaled random function

$$Y(t) = \frac{1}{r^H}X(rt)$$

for a given $r > 0$ has the same statistical properties as X. Thus it also has the same spectral density. From this basic observation we can deduce the important result (2.10) using only the above definitions and some elementary calculus as follows.

Let us fix some $r > 0$, set

$$Y(t,T) = \begin{cases} Y(t) = \frac{1}{r^H} X(rt), & \text{if } 0 < t < T \\ 0, & \text{otherwise} \end{cases}$$

and adopt the notation

$F_X(t,T), F_Y(t,T)$	Fourier transforms of $X(t,T), Y(t,T)$,
$S_X(f,T), S_Y(f,T)$	spectral densities of $X(t,T), Y(t,T)$,
$S_X(f), S_Y(f)$	spectral densities of $X(t), Y(t)$.

We compute

$$F_Y(f,T) = \int_0^T Y(t) e^{-2\pi i f t} dt = \frac{1}{r^H} \int_0^{rT} X(s) e^{-2\pi i \frac{f}{r} s} \frac{ds}{r},$$

where we have substituted $\frac{s}{r}$ for t and $\frac{ds}{r}$ for dt in the second integral. Thus, clearly

$$F_Y(f,T) = \frac{1}{r^{H+1}} F_X(\frac{f}{r}, rT).$$

Now it follows for the spectral density of $Y(t,T)$

$$S_Y(f,T) = \frac{1}{r^{2H+1}} \frac{1}{rT} |F_X(\frac{f}{r}, rT)|^2$$

and in the limit as $T \to \infty$ or equivalently as $rT \to \infty$ we conclude

$$S_Y(f) = \frac{1}{r^{2H+1}} S_X(\frac{f}{r}).$$

Since Y is just a properly rescaled version of X, their spectral densities must coincide, thus, also

$$S_X(f) = \frac{1}{r^{2H+1}} S_X(\frac{f}{r}).$$

Now we formally set $f = 1$ and replace $1/r$ again by f to finally obtain the desired result (2.10)

$$S_X(f) \propto \frac{1}{f^{2H+1}} = \frac{1}{f^\beta}.$$

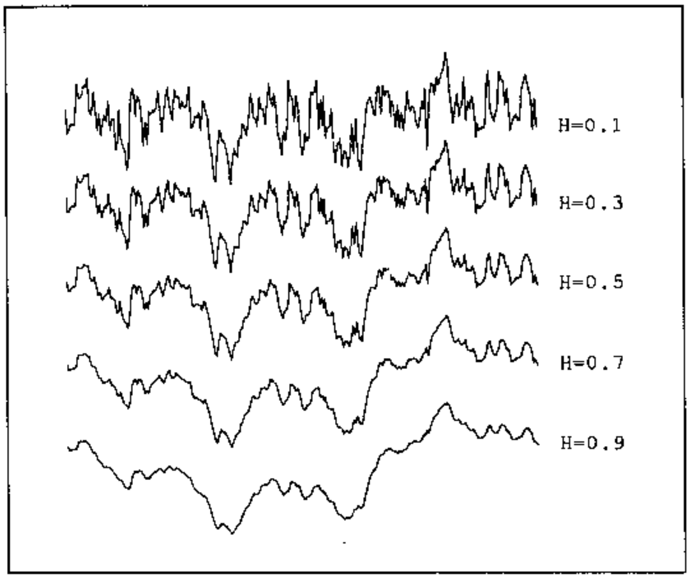


Fig. 2.9: Fractal motion via spectral synthesis. The above curves correspond to spectral density functions of the form $1/f^{\beta}$ where $\beta = 2H + 1$. The fractal dimensions of these graphs are $2 - H$.

2.4.3 The Fourier filtering method

For the practical algorithm we have to translate the above into conditions on the coefficients a_k of the discrete Fourier transform

$$\overline{X}(t) = \sum_{k=0}^{N-1} a_k e^{2\pi i k t} \tag{2.12}$$

The coefficients a_k are in a 1:1 correspondence with the complex values $\overline{X}(t_k)$, $t_k = \frac{k}{N}, k = 0, 1, \ldots, N-1$. The condition to be imposed on the coefficients in order to obtain $S(f) \propto \frac{1}{f^{\beta}}$ now becomes

$$E(|a_k|^2) \propto \frac{1}{k^{\beta}} \tag{2.13}$$

since k essentially denotes the frequency in (2.12). This relation (2.13) holds for $0 < k < \frac{N}{2}$. For $k \geq \frac{N}{2}$ we must have $a_k = \overline{a_{N-k}}$ because $\overline{X}$ is a real function. The method thus simply consists of randomly choosing coefficients subject to the expectation in (2.13) and then computing the inverse Fourier transform to obtain X in the time domain. Since the process X need only have real values it is actually sufficient to sample real random variables A_k and B_k under the constraint

$$E(A_k^2 + B_k^2) \propto \frac{1}{k^{\beta}}$$

and then set

$$\overline{X}(t) = \sum_{k=1}^{N/2} (A_k \cos kt + B_k \sin kt). \tag{2.14}$$

In contrast to some of the previous algorithms discussed this method is not recursive and also does not proceed in stages of increasing spatial resolution. We may, however, interpret the addition of more and more random Fourier coefficients a_k satisfying (2.13) as a process of adding higher frequencies, thus, increasing the resolution in the frequency domain.

ALGORITHM SpectralSynthesisFM1D (X, N, H, seed)
Title: Fractal motion using Fourier filtering method

Arguments	X[]	array of reals of size N
	N	size of array X
	H	$0 < H < 1$ determines fractal dimension $D = 2 - H$
	seed	seed value for random number generator
Globals	Arand	rand() returns values between 0 and Arand
Variables	i	integer
	beta	exponent in the spectral density function ($1 <$ beta < 3)
	rad, phase	polar coordinates of Fourier coefficient
	A[], B[]	real and imaginary parts of Fourier coefficients
Subroutines	InvFFT	fast inverse Fourier transform in 1 dimension

```
BEGIN
    InitGauss (seed)
    beta := 2 * H + 1
    FOR i := 0 TO N/2-1 DO
        rad := power (i+1, -beta/2) * Gauss ()
        phase := 2 * 3.141592 * rand () / Arand
        A[i] := rad * cos (phase)
        B[i] := rad * sin (phase)
    END FOR
    InvFFT (A, B, X, N/2)
END
```

The advantage of this straight forward method is that it is the purest interpretation of the concept of fractional Brownian motion. Artifacts such as those occurring in the midpoint displacement methods are not apparent. However, due to the nature of Fourier transforms, the generated samples are periodic. This is sometimes annoying, and in this case one can compute twice or four times as many points as actually needed and then discard a part of the sequence.

The algorithm for the inverse Fourier transformation *InvFFT* is not included with the pseudo code *SpectralSynthesisFM1D()*. It should compute the sums (2.14) from the given coefficients in A and B. Usually fast Fourier transforms are employed. They require $O(N \log N)$ operations per transform.

2.5 Extensions to higher dimensions

2.5.1 Definitions

In this section we discuss how one can generalize the displacement methods and the spectral synthesis methods to two and three dimensions.

The generalization of fractional Brownian motion itself is straight forward. It is a multidimensional process (a random field) $X(t_1, t_2, \dots, t_n)$ with the properties:

(i) The increments $X(t_1, t_2, \dots, t_n) - X(s_1, s_2, \dots, s_n)$ are Gaussian with mean 0

(ii) The variance of the increments $X(t_1, t_2, \dots, t_n) - X(s_1, s_2, \dots, s_n)$ depends only on the distance

$$\sqrt{\sum_{i=1}^{n}(t_i - s_i)^2}$$

and in fact is proportional to the 2H-th power of the distance, where the parameter H again satisfies $0 < H < 1$. Thus,

$$E(|X(t_1, t_2, \dots, t_n) - X(s_1, s_2, \dots, s_n)|^2) \propto (\sum_{i=1}^{n}(t_i - s_i)^2)^H. \tag{2.15}$$

The random field X again has stationary increments and is isotropic, i.e. all points $(t_1, t_2, \dots, t_n)$ and all directions are statistically equivalent. In the frequency domain we have for the spectral density

$$S(f_1, \dots, f_n) \propto \frac{1}{\left(\sqrt{\sum_{i=1}^{n} f_i^2}\right)^{2H+n}}. \tag{2.16}$$

This fact can be deduced in the exact same fashion as in the last section for the $\frac{1}{f^\beta}$ law in the one-dimensional case. Therefore we skip these details here. This ensures that X restricted to any straight line will be a $\frac{1}{f^\beta}$ noise corresponding to $2H = \beta - 1$ [103]. In analogy with formula (2.10) the fractal dimension of the graph of a sample of $X(t_1, t_2, \dots, t_n)$ is

$$D = n + 1 - H. \tag{2.17}$$

2.5.2 Displacement methods

The midpoint displacement methods can work with square lattices of points. If the mesh size δ denotes the resolution of such a grid, we obtain another square grid of resolution $\frac{\delta}{\sqrt{2}}$ by adding the midpoints of all squares. Of course, the orientation of the new square lattice is rotated by 45 degrees. Again adding the midpoints of all squares gives us the next lattice with the same orientation as the first one and the resolution is now $\frac{\delta}{2}$ (see Figure 2.10). In each stage we thus scale the resolution with a factor of $r = \frac{1}{\sqrt{2}}$, and in accordance with (2.15) we add random displacements using a variance which is r^{2H} times the variance of the previous stage. If we assume that the data on the four corner points of the grid carry mean square increments of σ^2 then at stage n of the process we must add a Gaussian random variable of variance $\sigma^2 r^{2Hn} = \sigma^2 (\frac{1}{2})^{nH}$. For the usual midpoint method random elements are added only to the new midpoints in each stage, whereas in the random addition method we add displacements at all points. Thus we can unify both methods in just one algorithm.

In Figures 2.11 to 2.13 we show topographical maps of random fractal landscapes which were generated using the algorithm *MidPointFM2D()*. Random additions and differing seed values were used, and the parameter H varies from 0.8 to 0.2. At a resolution of 65 by 65 points (*maxlevel* = 6) a total of 4225 data points were generated. The heights of the surface were scaled to the range from -10000 to $+10000$ and then handed to a contour line program which produced the maps. The parts of the surfaces that have a negative height are assumed to be submerged under water and are not displayed. The fractal dimension of these surfaces is $3 - H$. A rendering of a perspective view of these landscapes is also included.

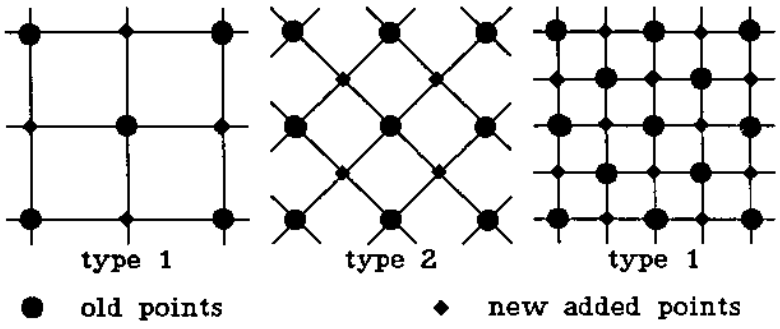


Fig. 2.10: Grid types for displacement methods. In the displacement methods a grid of type 1 is given at the beginning from which a type 2 grid is generated. Its mesh size is $1/\sqrt{2}$ times the old mesh size. In a similar step a grid of type 1 is again obtained as shown in the figure.

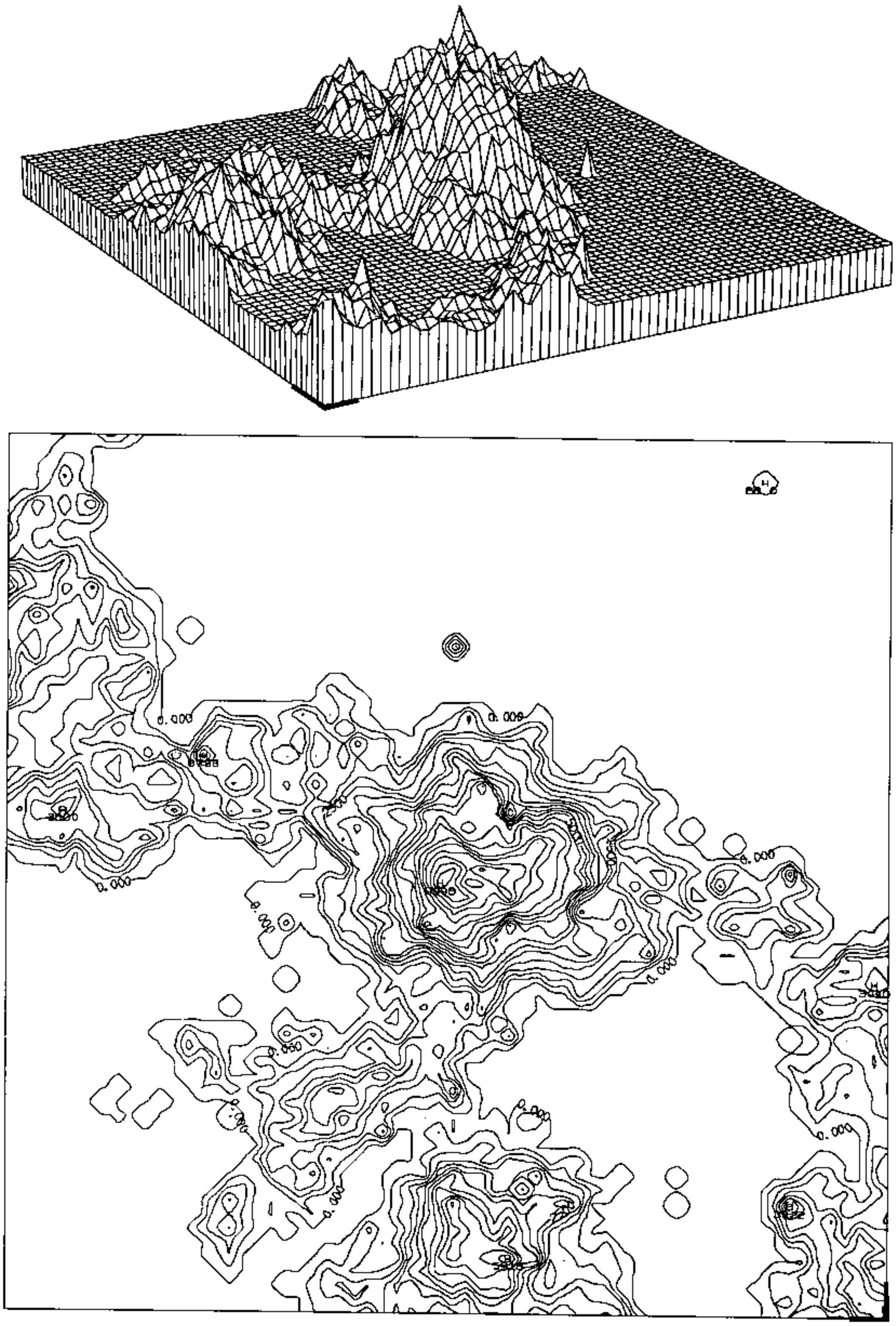


Fig. 2.11: Topographical map and perspective view of random fractal ($H = 0.8$).

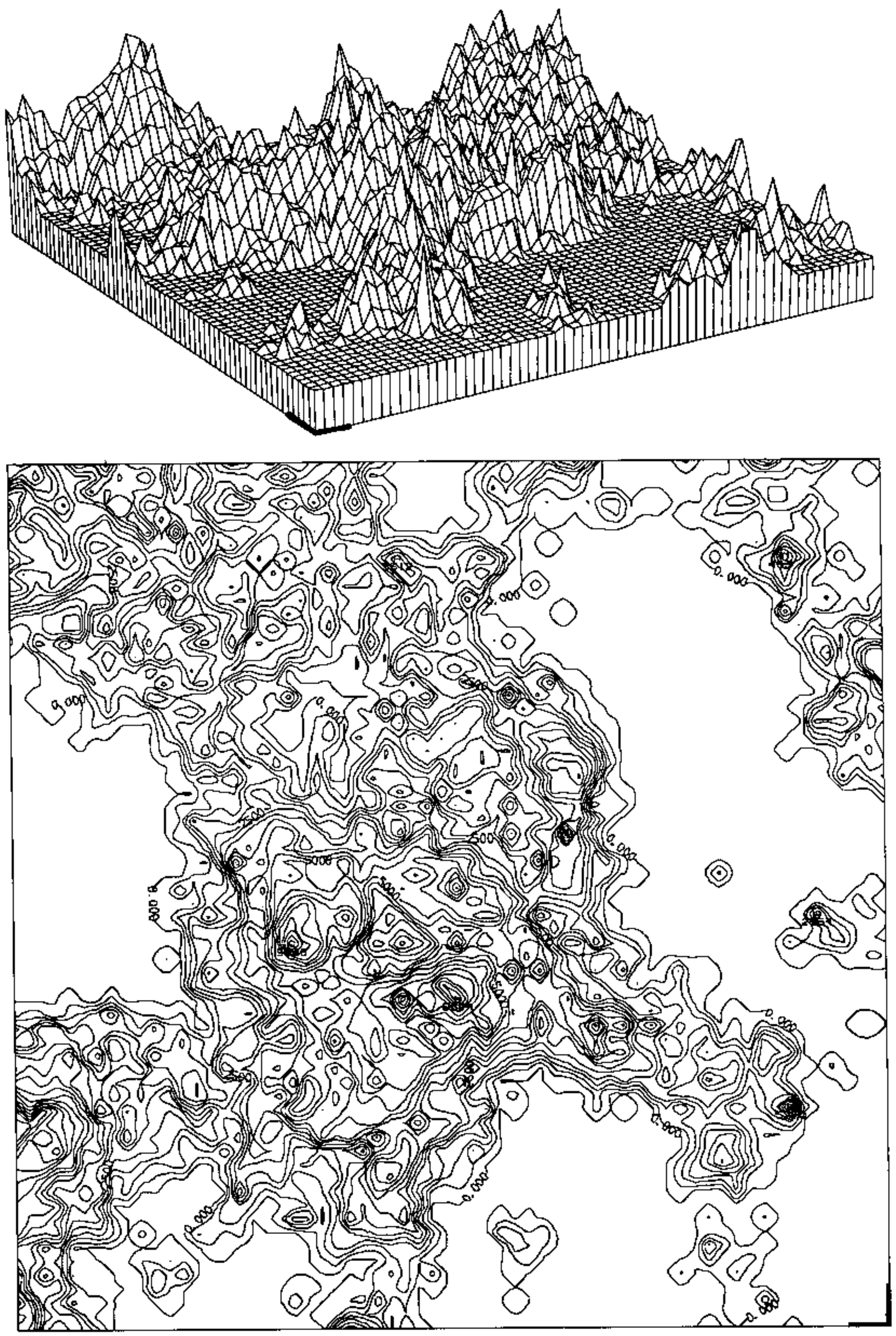


Fig. 2.12: Topographical map and perspective view of random fractal ($H = 0.5$).

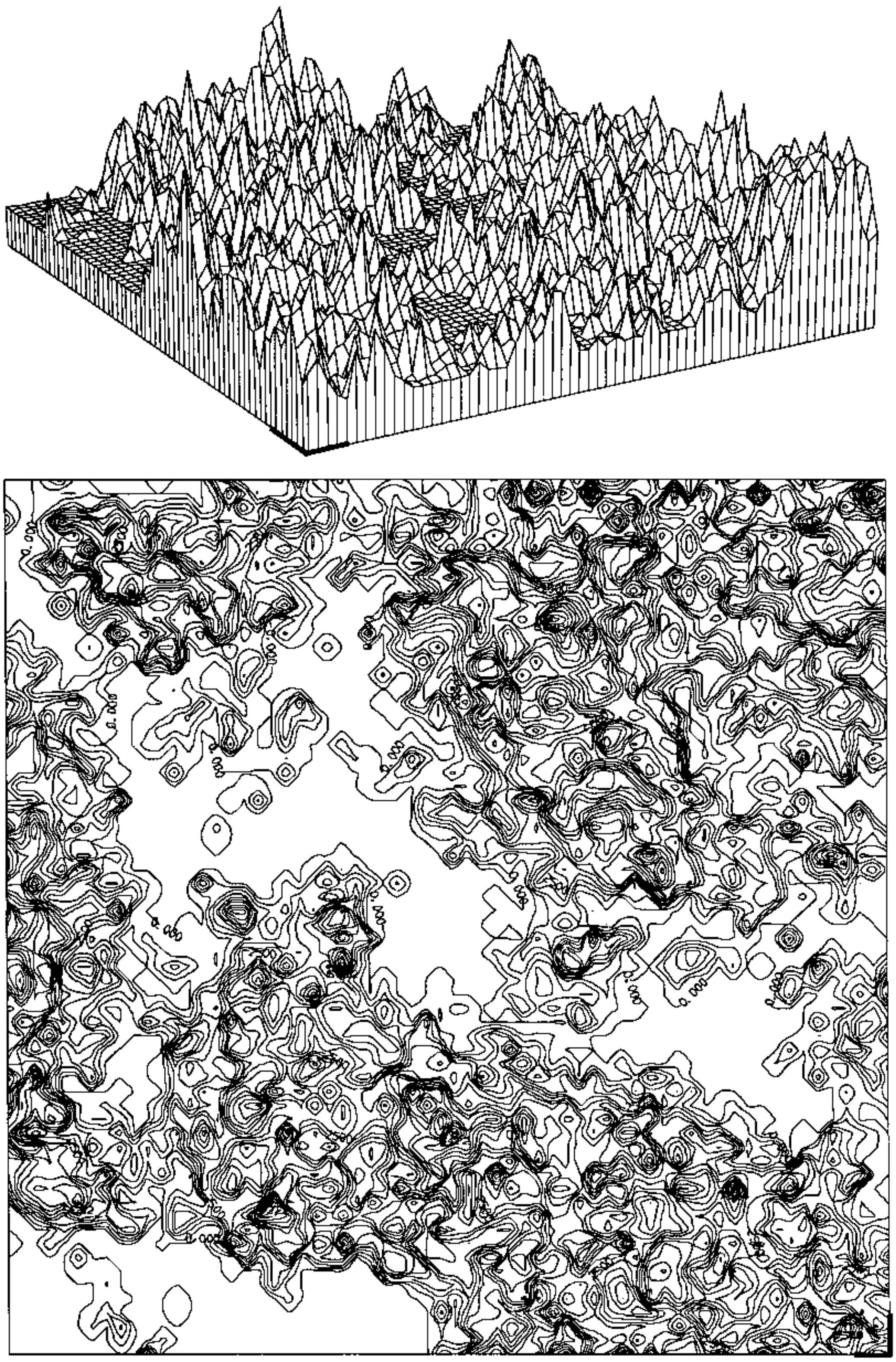


Fig. 2.13: Topographical map and perspective view of random fractal ($H = 0.2$).

```
ALGORITHM MidPointFM2D (X, maxlevel, sigma, H, addition, seed)
Title       Midpoint displacement and successive random additions in 2 dimensions

Arguments   X[][]            doubly indexed real array of size $(N + 1)^2$
            maxlevel         maximal number of recursions, $N = 2^{maxlevel}$
            sigma            initial standard deviation
            H                parameter $H$ determines fractal dimension $D = 3 - H$
            addition         boolean parameter (turns random additions on/off)
            seed             seed value for random number generator
Variables   i, N, stage      integers
            delta            real holding standard deviation
            x, y, y0, D, d   integer array indexing variables
Functions   f3(delta,x0,x1,x2) = (x0+x1+x2)/3 + delta * Gauss()
            f4(delta,x0,x1,x2,x3) = (x0+x1+x2+x3)/4 + delta * Gauss()

BEGIN
    InitGauss (seed)
    N = power (2, maxlevel)
                                    /* set the initial random corners */
    delta := sigma
    X[0][0] := delta * Gauss ()
    X[0][N] := delta * Gauss ()
    X[N][0] := delta * Gauss ()
    X[N][N] := delta * Gauss ()
    D := N
    d := N / 2
    FOR stage := 1 TO maxlevel DO
                                    /* going from grid type I to type II */
        delta := delta * power (0.5, 0.5*H)
                                    /* interpolate and offset points */
        FOR x :=d TO N-d STEP D DO
            FOR y := d TO N-d STEP D DO
                X[x][y] := f4 (delta, X[x+d][y+d], X[x+d][y-d], X[x-d][y+d],
                    X[x-d][y-d])
            END FOR
        END FOR
                                    /* displace other points also if needed */
        IF (addition) THEN
            FOR x := 0 TO N STEP D DO
                FOR y := 0 TO N STEP D DO
                    X[x][y] := X[x][y] + delta * Gauss ()
                END FOR
            END FOR
        END IF
        (to be continued on the next page)
```

We also remark that one does not necessarily have to work with rectangular grids. Some authors also consider triangular subdivisions (see [40,49,50,76] and Appendix A). For a detailed discussion of displacement techniques, their history and possible extensions, we refer to Appendix A.

The algorithm *InterpolatedFM1D()* of Section 2.3 implementing variable scaling factors of resolutions can be modified to compute approximations of

```
ALGORITHM MidPointFM2D (X, maxlevel, sigma, H, addition, seed)
Title        Midpoint displacement and successive random additions in 2 dimensions
             (continued from previous page)

                        /* going from grid type II to type I */
          delta := delta * power (0.5, 0.5*H)
                        /* interpolate and offset boundary grid points */
          FOR x := d TO N-d STEP D DO
               X[x][0] := f3 (delta, X[x+d][0], X[x-d][0], X[x][d])
               X[x][N] := f3 (delta, X[x+d][N], X[x-d][N], X[x][N-d])
               X[0][x] := f3 (delta, X[0][x+d], X[0][x-d], X[d][x])
               X[N][x] := f3 (delta, X[N][x+d], X[N][x-d], X[N-d][x])
          END FOR
                        /* interpolate and offset interior grid points */
          FOR x := d TO N-d STEP D DO
               FOR y := D TO N-d STEP D DO
                    X[x][y] := f4 (delta, X[x][y+d], X[x][y-d], X[x+d][y], X[x-d][y])
               END FOR
          END FOR
          FOR x := D TO N-d STEP D DO
               FOR y := d TO N-d STEP D DO
                    X[x][y] := f4 (delta, X[x][y+d], X[x][y-d], X[x+d][y], X[x-d][y])
               END FOR
          END FOR
                        /* displace other points also if needed */
          IF (addition) THEN
               FOR x := 0 TO N STEP D DO
                    FOR y := 0 TO N STEP D DO
                         X[x][y] := X[x][y] + delta * Gauss ()
                    END FOR
               END FOR
               FOR x := d TO N-d STEP D DO
                    FOR y := d TO N-d STEP D DO
                         X[x][y] := X[x][y] + delta * Gauss ()
                    END FOR
               END FOR
          END IF
          D := D / 2
          d := d / 2
     END FOR
END
```

fBm in two or three (or even higher) dimensions. If d denotes this dimension we have to fill an array of size N^d elements subject to (2.15). These points are evenly spaced grid points in a unit cube, thus the final resolution (grid size) is $1/(N-1)$. At a particular stage of this algorithm we have an approximation at resolution $1/(M-1)$ given in the form of an array of size M^d with $M < N$. The next stage is approached in two steps: 1. If $0 < r < 1$ denotes the factor at which the resolution changes, then a new approximation consisting of L^d points where $L \approx M/r$ must be computed. First the values of these numbers are taken as multilinear or higher order interpolation of the M^d old values.

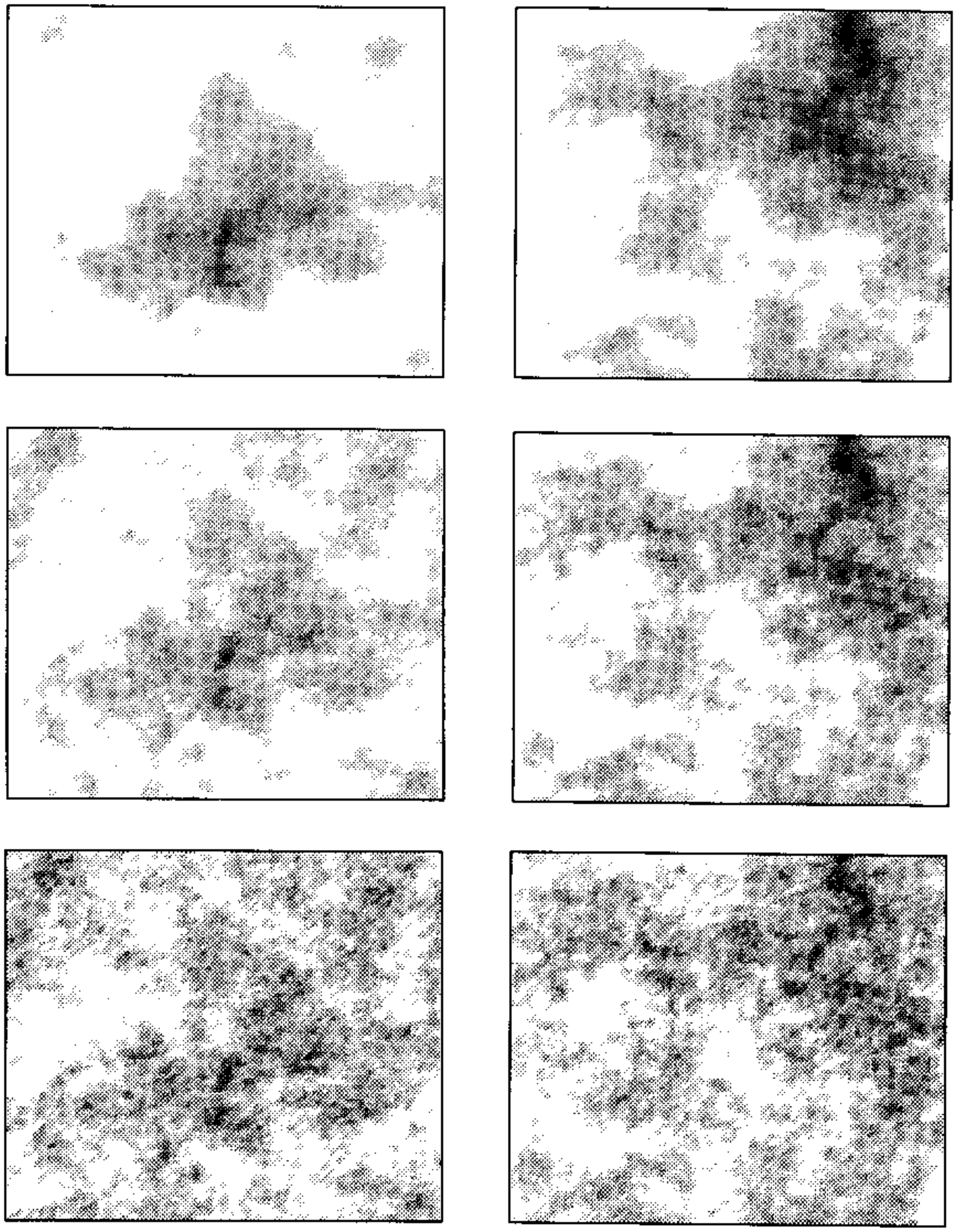
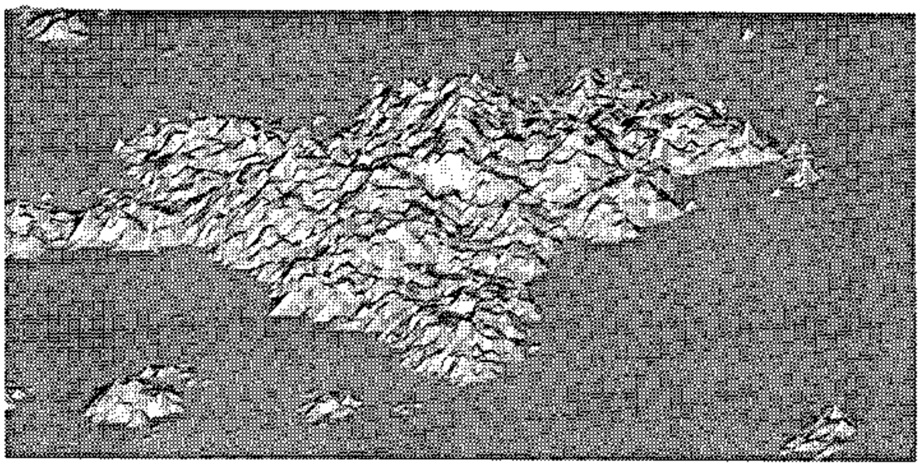


Fig. 2.14: Fractal clouds with varying dimensions. In two sequences (left and right) we show the effect of the parameter H. The data for the clouds was produced in the same way as in the Figures 2.11 to 2.13 (the parameter maxlevel was at 7 though). The rendering displays a top view of the "landscape" where the height now defines the shade of gray to be used. Points with heights below zero are not drawn, i.e. they appear as white (see Section 2.7.1). The parameter H varies from 0.8 (top) to 0.5 (middle) to 0.2 (bottom).



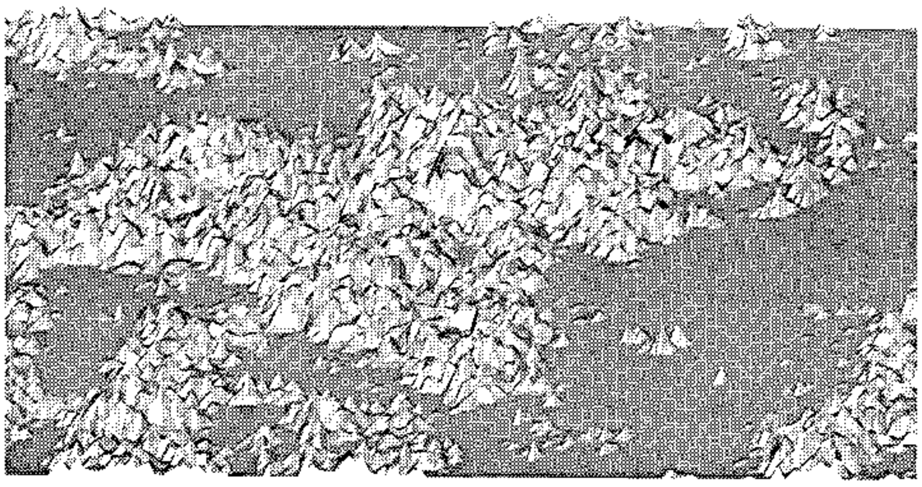


Fig. 2.15: Clouds as mountains. The two first clouds of Figure 2.14 are rendered as mountains here.

2. A random element of variance proportional to $1/(L-1)^{2H}$ is added to all points. These two steps are repeated until we have obtained N^d points.

We briefly describe the multilinear interpolation (compare [86]). Let us first take $d = 2$ and assume that a grid point (x, y) of the new grid falls into a square of old grid points $(x_0, y_0), (x_0, y_1), (x_1, y_0), (x_1, y_1)$ (see Figure 2.16). We can express (x, y) relative to these points by

$$\begin{aligned} x &= (1-h_x)x_0 + h_x x_1 \\ y &= (1-h_y)y_0 + h_y y_1 \end{aligned}$$

where $0 \le h_x, h_y \le 1$. If $X(x_i, y_j)$ denotes the approximation of fBm at the

ALGORITHM InterpolatedFM (X, N, d, r, sigma, H, seed)
Title Fractal motion via interpolation and variable scaling

Arguments	X[]	real array of size N^d
	N	number of elements in X along one dimension
	d	dimension of the space (1, 2, 3 or more)
	r	scaling factor for resolutions ($0 < r < 1$)
	sigma	initial standard deviation
	H	parameter $0 < H < 1$ determines fractal dimension
	seed	seed value for random number generator
Variables	delta	real variable holding standard deviations Δ
	Y[]	real array of size N for the interpolated values of X
	mT, mt	integers, number of elements N^d in arrays X and Y along one dimension
	t, T	real sampling rates for X and Y
	i, index	integer
	h	real
Subroutine	Interpolate()	multilinear interpolation routine (see text)

```
BEGIN
                          /* initialize the array with 2^d points */
    InitGauss (seed)
    mT := 2
    FOR i := 0 TO power(2,d)-1 DO
        X[i] := sigma * Gauss ()
    END FOR
                          /* loop while less than N^d points in array */
    WHILE (mT < N) DO
                          /* set up new resolution of mt points */
        mt := mT / r
        IF (mt = mT) THEN mt := mT + 1
        IF (mt > N) THEN mt := N
        t := 1.0 / (mt-1)
        T := 1.0 / (mT-1)
                          /* interpolate new points from old points */
        Interpolate (X, mT, Y, mt, d)
                          /* compute the standard deviation for offsets */
        delta := sqrt (0.5) * sigma * sqrt (1.0 - power (t/T, 2-2*H)) * power (t, H)
                          /* do displacement at all positions */
        FOR i := 0 TO power(mt,d)-1 DO
            X[i] := Y[i] + delta * Gauss ()
        END FOR
        mT := mt
    END WHILE
END
```

old points we now interpolate at the point (x, y) by setting

$$\begin{aligned} X(x,y) &= (1-h_x)(1-h_y)X(x_0,y_0) + h_x(1-h_y)X(x_1,y_0) \\ &+ (1-h_x)h_yX(x_0,y_1) + h_xh_yX(x_1,y_1). \end{aligned}$$

In three dimensions ($d = 3$) we have (x, y, z) with x, y as above and $z = (1-h_z)z_0 + h_z z_1$ and the above interpolation formula extends to this case in

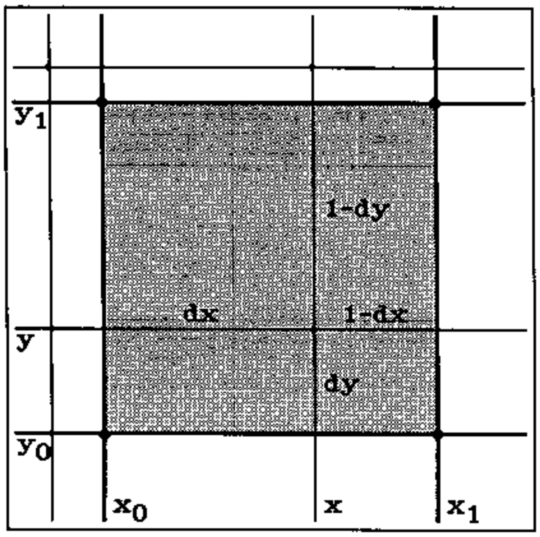


Fig. 2.16: Multilinear interpolation (see text).

the obvious way. In the pseudo code *InterpolatedFM()* we do not include these very technical details. The points are stored in a long linear array X of floating point numbers and the routine *Interpolate* should be the implementation of the multilinear interpolation. Its inputs are the array X, the number of points along one of the dimensions, the dimension, and the requested number of points in one dimensions. The output is a new array Y with the specified size.

2.5.3 The Fourier filtering method

We now proceed to the extension of the Fourier filtering method to higher dimensions. In two dimensions the spectral density S generally will depend on two frequency variables u and v corresponding to the x and y directions. But since all directions in the xy-plane are equivalent with respect to statistical properties, S depends only on $\sqrt{u^2 + v^2}$. If we cut the surface along a straight line in the xy-plane, we expect for the spectral density S of this fBm in only one dimension a power law $1/f^\beta$ as before. This requirement implies (see (2.16)) for the two-dimensional spectral density to behave like

$$S(u, v) = \frac{1}{(u^2 + v^2)^{H+1}}.$$

The two-dimensional discrete inverse Fourier transform is

$$X(x, y) = \sum_{k=0}^{N-1} \sum_{l=0}^{N-1} a_{kl} e^{2\pi i(kx+ly)}$$

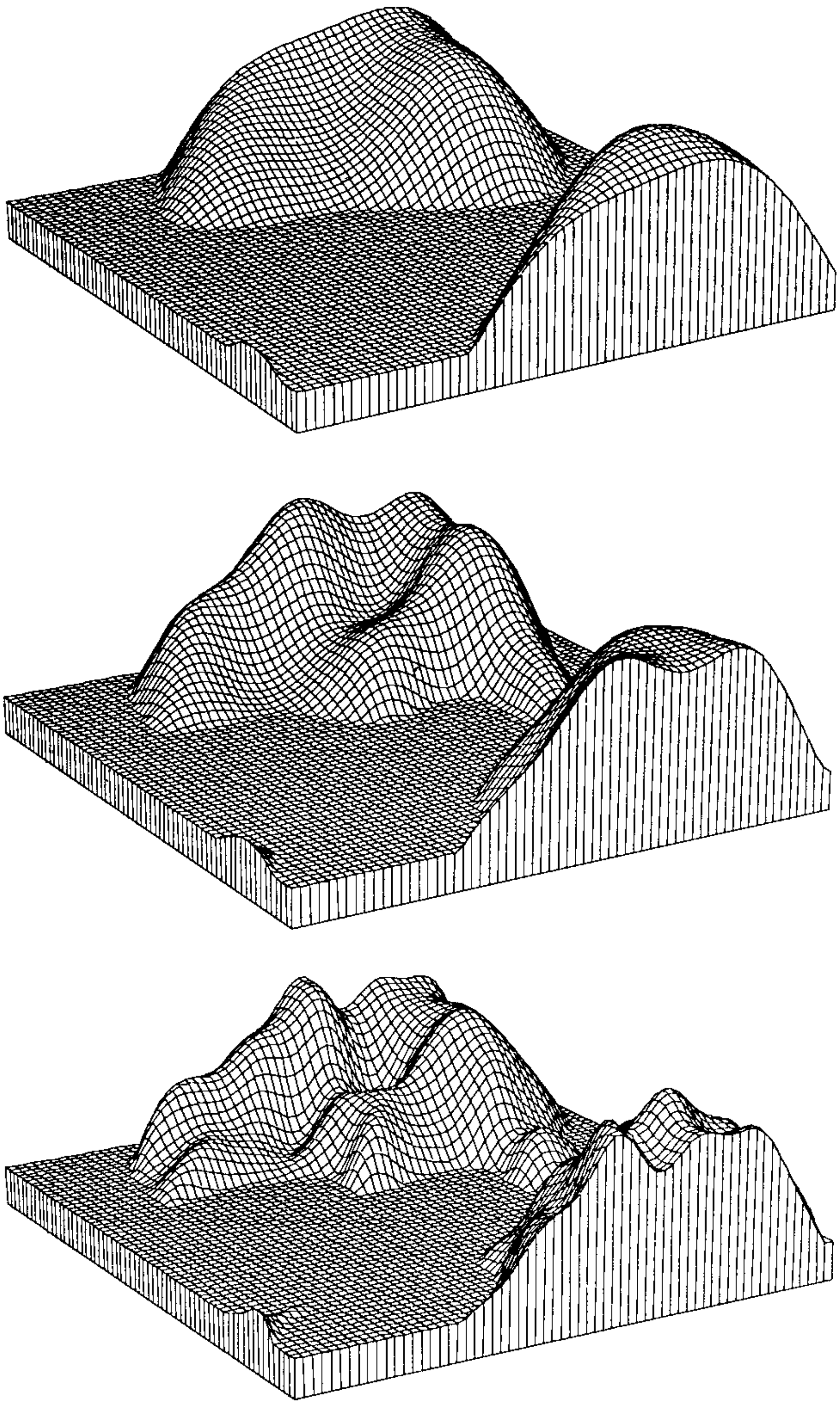


Fig. 2.17: Spectral synthesis of a mountain.

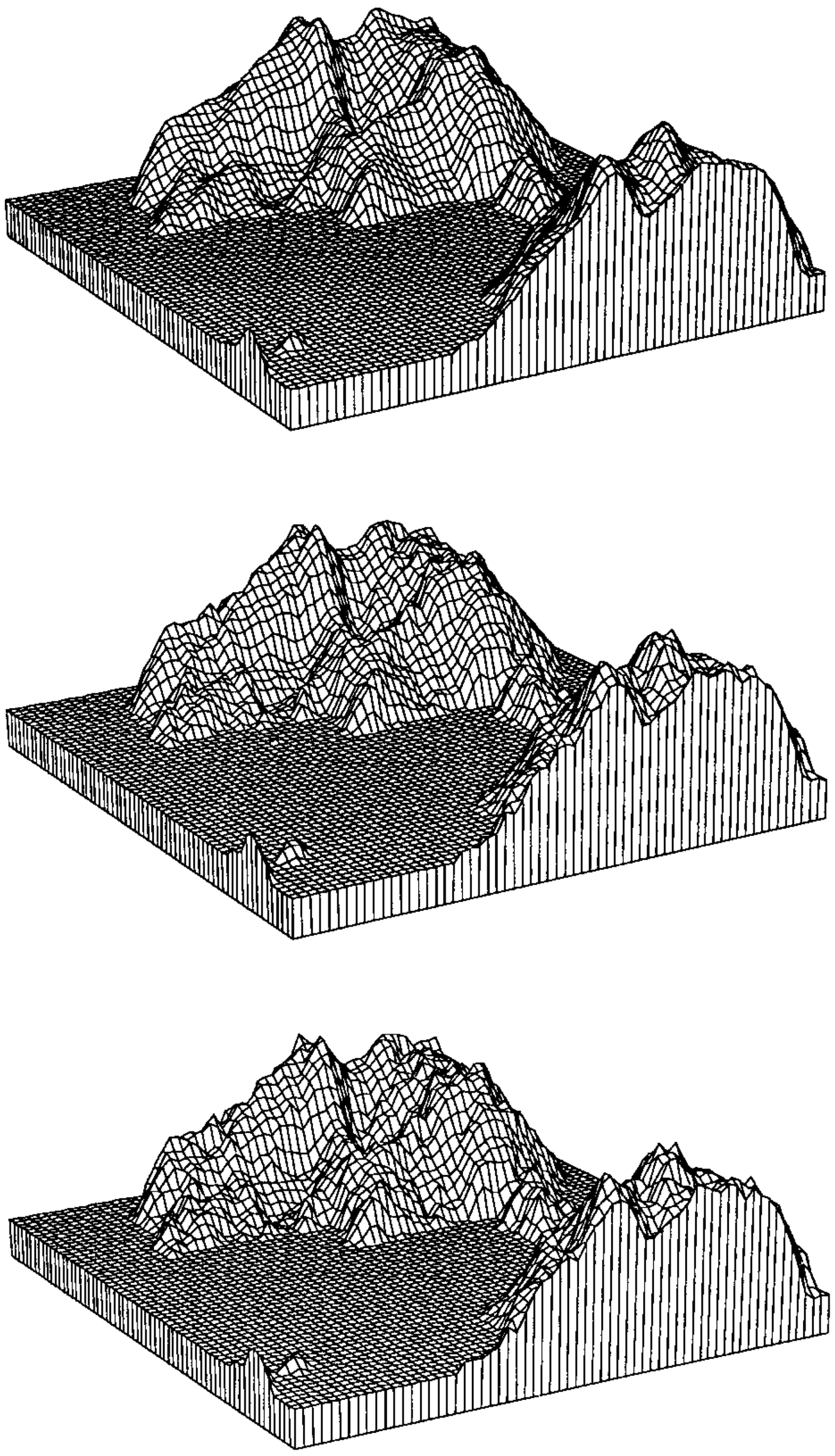


Fig. 2.18: Spectral synthesis of a mountain (continued).

ALGORITHM SpectralSynthesisFM2D (X, N, H, seed)
Title Fractal motion in 2 dimensions

Arguments	X[][]	doubly indexed array of complex variables of size N^2
	N	size of array X along one dimension
	H	$0 < H < 1$ determines fractal dimension $D = 3 - H$
	seed	seed value for random number generator
Globals	Arand	rand() returns values between 0 and Arand
Variables	i, j, i0, j0	integers
	rad, phase	polar coordinates of Fourier coefficient
	A[][]	doubly indexed array of complex variables of size N^2
Subroutines	InvFFT2D	fast inverse Fourier transform in 2 dimensions
	rand()	returns system random numbers

```
BEGIN
    InitGauss (seed)
    FOR i = 0 TO N/2 DO
        FOR j = 0 TO N/2 DO
            phase := 2 * 3.141592 * rand () / Arand
            IF (i≠0 OR j≠0) THEN
                rad := power(i*i+j*j,-(H+1)/2) * Gauss()
            ELSE
                rad := 0
            END IF
            A[i][j] := (rad * cos(phase), rad * sin(phase))
            IF (i=0) THEN
                i0 := 0
            ELSE
                i0 := N - i
            END IF
            IF (j=0) THEN
                j0 := 0
            ELSE
                j0 := N - j
            END IF
            A[i0][j0] := (rad * cos(phase), - rad * sin(phase))
        END FOR
    END FOR
    A[N/2][0].imag := 0
    A[0][N/2].imag := 0
    A[N/2][N/2].imag := 0
    FOR i = 1 TO N/2-1 DO
        FOR j = 1 TO N/2-1 DO
            phase := 2 * 3.141592 * rand() / Arand
            rad := power(i*i+j*j,-(H+1)/2) * Gauss()
            A[i][N-j] := (rad * cos(phase), rad * sin(phase))
            A[N-i][j] := (rad * cos(phase), - rad * sin(phase))
        END FOR
    END FOR
    InvFFT2D(A,X,N)
END
```

for $x, y = 0, \frac{1}{N}, \frac{2}{N}, \ldots, \frac{N-1}{N}$ (see [46]), and thus, we specify for the coefficients a_{kl}

$$E(|a_{kl}|^2) \propto \frac{1}{(k^2 + l^2)^{H+1}}.$$

Since the constructed function X is a real function we must also satisfy a conjugate symmetry condition, namely

$$\begin{array}{lll} a_{N-i,N-j} & = \overline{a_{i,j}} & \text{for} \quad i, j > 0, \\ a_{0,N-j} & = \overline{a_{0,j}} & \text{for} \quad j > 0, \\ a_{N-i,0} & = \overline{a_{i,0}} & \text{for} \quad i > 0, \\ a_{0,0} & = \overline{a_{0,0}}. & \end{array}$$

The fractal dimension D_f of the surface will be $D_f = 3 - H$. The algorithm then consists simply of choosing the Fourier coefficients accordingly and then performing a two-dimensional inverse transform.

The code for the inverse Fourier transformation is not included. It can easily be constructed from the one-dimensional version *InvFFT* (see [46]). As in the one-dimensional case one can reduce the number of coefficients since the random function is only real. The method also generalizes to three dimensions (see Chapter 1).

In the sequence of Figure 2.17 and 2.18 we show how the spectral synthesis method "adds detail" by improving the spectral representation of the random fractal, i.e. by allowing more and more Fourier coefficients to be employed. The resolution in the algorithm *SpectralSynthesisFM2D()* was $N = 64$ but in the top image of Figure 2.17 only the first $2^2 = 4$ coefficients were used. In the other pictures we allowed $4^2 = 16$ (middle) and $8^2 = 64$ (bottom) non-zero coefficients. In Figure 2.18 we increase the number of Fourier coefficients to 16^2 (top), 32^2 (middle) and 64^2 (bottom).

2.6 Generalized stochastic subdivision and spectral synthesis of ocean waves

A glance at the SIGGRAPH proceedings of the last few years will reveal that the computer graphics community is very interested in the rendering aspect of fractals and other procedurally defined objects. In particular ray tracing methods have been adapted to handle the infinite detail at small scales that is characteristic for fractals. The images of random fractals presented by Voss demonstrate the perfection with which it is now possible to generate mountain scenes, craters on a moon etc. The question remains in what way the methods used for the production of random fractals can be modified in order to model even more natural phenomena. In this last section we will briefly review two such approaches published in the recent papers [60] and [74]. Both papers pursue the idea of spectral synthesis.

J.P. Lewis generalizes the displacement methods in his paper [60] to provide a better control of the synthesized random functions. He calls it *generalized stochastic subdivision.* His generated noises are stationary with a prescribed autocorrelation function

$$R(s) = E(X(t)X(t+s)).$$

This information is equivalent to the spectral density function since the two functions are a Fourier transform pair as stated by the Wiener-Khintchine relation. In the usual displacement techniques a new point is first obtained via interpolation from its immediate neighbors and then displaced by a random element. The generalization also has to take points into account that are not immediate neighbors. In the one-dimensional case, if X is known at times $t+t_k$, then sums of the form

$$\hat{X}(t) = \sum_k a_k X(t+t_k)$$

yield a linear estimator for the random function X at time t. In the usual midpoint technique, valid for spectral density functions $S(f) \propto \frac{1}{f^\beta}$, this sum just ranges over two points, and the weights are both equal to $\frac{1}{2}$. In the generalized stochastic subdivision the weights a_k have to be chosen in harmony with the prescribed autocorrelation function $R(s)$ or equivalently with the spectral density function $S(f)$ because in the general case the expected value of X at a midpoint no longer is just the average of its two neighbors. The values of the coefficients are deduced using an orthogonality principle known in estimation theory (see the references in [60]). Eventually this amounts to solving a linear system of equations at each stage of the subdivision method. Of course, the estimated value $\hat{X}(t)$ is finally displaced by a random element with a properly chosen variance. Following Lewis, some of the expected advantages of this method over the direct synthesis via the spectral density function and inverse Fourier transformation are (1) that the resolution of synthesized functions may be adjusted to the resolution required by its local perspective projection in the rendering process, (2) that portions of the noise may be reproduced at different locations and at different resolutions, and (3) that the random function can be required to pass through certain points, e.g. to ensure boundary continuity with adjacent objects. Some of the considered spectra are the Markovian-like spectrum $1/(a^2+f^2)^d$, where the parameter d is similar to H in the random fractals (set $2d = 2H+1$), and the Lorentzian spectrum, $1/(a^2+b^2(f-f_0)^2)$. In two dimensions spectra with directional trends and others yielding oscillatory textures are easily accommodated in the setup of generalized stochastic subdivision.

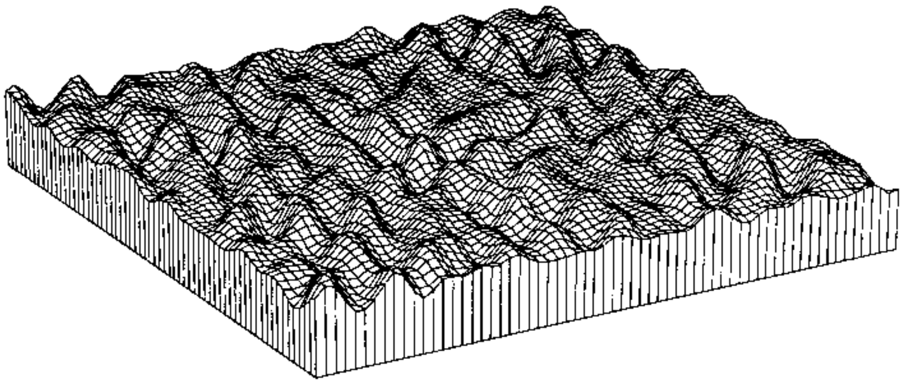


Fig. 2.19: Nonfractal waves. The spectral synthesis method was applied using a spectral density function similar to (2.18). These waves are not fractals since the spectral density falls off as $1/f^5$ for high frequencies.

The objective of the other paper by G. Mastin et al. [74] is to model a "fully developed sea in nature" using the spectral synthesis method with a spectral density function based on empirical results. Earlier researchers had recorded measurements of waves leading to a spectral model first in the downwind direction and then also in other directions. The resulting two-dimensional spectrum of these models, called the modified Pierson-Moskowics spectrum, can be formulated as

$$S(f,\phi) = \frac{a}{f^5} e^{-(\frac{b}{f})^4} D(f,\phi). \tag{2.18}$$

Here (f,ϕ) are the polar coordinates of the frequency variables; a and b are certain positive constants;

$$D(f,\phi) = \frac{1}{N(f)}(\cos(\frac{\phi}{2}))^{2p(f)}$$

is a directional spreading factor that weighs the spectrum at angles ϕ from the downwind direction; $p(f)$ is a piecewise power function of the type

$$p(f) = \begin{cases} a_1(f/f_m)^{b_1} & \text{if } f < f_m \\ a_2(f/f_m)^{-b_2} & \text{if } f \geq f_m \end{cases}$$

where f_m, a_1, a_2, b_1 and b_2 are positive constants; and $N(f)$ is chosen such that

$$\int_{-\pi}^{\pi} D(f,\phi)d\phi = 1.$$

There are numerous parameters and constants that enter into this formulation, e.g. the peak frequency f_m, the gravitational constant g and the wind speed at a given height above sea level.

For the implementation the given spectral density determines the expected magnitude of the complex Fourier coefficients of the sea model. In the paper this is achieved by filtering a white noise image appropriately, i.e. by multiplying the squared magnitudes of the Fourier transform of the white noise with the spectral density function. A following inverse transform then yields the sea model as a height function over the xy-plane. Plate 16 shows the result in a raytraced image with an added maehlstrom.

2.7 Computer graphics for smooth and fractal surfaces

The output of the algorithms for the synthesis of fractal surfaces (*MidPointFM2D()*, *InterpolatedFM()* and *SpectralSynthesisFM2D()*) is given in the form of an array which holds surface elevation data for a rectangular or square region. This data can be graphically represented in various ways. We can distinguish two cases depending on the relation between the screen resolution and the computed resolution of the surface (number of points). If the number of computed elevations is large enough, i.e. approximately equal to the number of pixels on the screen, we can use these points as primitives for the rendering. In the other case we have to interpolate the surface between data points e.g. by using polygons or bilinear patches. Ray tracings have added another layer of realism to fractals. We forgo these methods and refer the reader to [14,18,58,76].

2.7.1 Top view with color mapped elevations

This is the simplest way to render a picture of the data. Geometrically the surface is treated as flat, and the elevation of the surface is color coded so that e.g. the low regions correspond to various shades of blue (depending on depth) and the high elevations are shown in shades of green, brown and white. In this way one can produce the equivalent of a geographical map of the landscape. If there are enough elevation data points we draw one pixel of color per point. Otherwise we can use the interpolation/multilinear of Section 2.5 to obtain as many new points as needed. E.g. for a screen resolution of 768 by 1024 an array of 192 by 256 elevations is sufficient. By multilinear interpolation the resolution can be quadrupled in the horizontal and vertical direction thus matching the given screen resolution. This procedure may be considered a special case of Gouraud shading. Color Plate 14 has been produced this way, the data were computed with *MidPointFM2D()*.

By changing the colors corresponding to various heights it is very easy to produce convincing clouds. Color Plate 15 shows the same data as Plate 14 but uses a color map as follows: We specify colors in RGB space (red, green, blue) with intensities varying from 0 to 255. Let h_{min} and h_{max} denote the minimum and maximum heights of the data and $\overline{h} = \frac{1}{2}(h_{min} + h_{max})$. Then the color corresponding to height h with $hmin < h < hmax$ is defined as follows : If $h \leq \overline{h}$, then the color values for red and green are 0, where the blue value is 255. If $h > \overline{h}$, then the value of the blue component remains at 255, while the red and green portions are given by

$$255 \cdot \frac{h - \overline{h}}{h_{max} - \overline{h}}.$$

This defines a color map which is blue in the lower half and is a color ramp from blue to white in the upper half. This method to generate pictures of clouds of course does not model the natural phenomenon "clouds" physically accurately. For example, the three-dimensionality is not taken into account.

In the above procedure the geometry of the scene is rather trivial, the family of displayed polygons forms a flat square or rectangle. The elevation data is transformed into color. The obvious extension of this is to use the elevation data as the third coordinate for the vertices of the polygons. This turns the flat surface into fractal terrain embedded in three-dimensional space. One then has to define a projection of the polygons onto the screen. The painter's algorithm sorts the polygons according to decreasing distance, and then draws them from back to front, thereby eliminating the hidden parts of the surface. This and the application of a lighting model is a very standard procedure (see e.g. [39]). Figure 2.15 is a coarse example. In addition to the three-dimensionality of the surface we again can use the elevation data to define colors relative to height. The lighting model then adds intensities.

2.7.2 Extended floating horizon method

All methods for the approximation of two-dimensional fractional Brownian motion easily generate as many points as wished. Using the Fourier synthesis method, however, it may take a longer time to perform the compute intensive 2D Fourier transformations. If enough points are produced it is possible in the rendering to avoid the interpolation that is introduced by using polygons. The points themselves serve as primitives for the rendering. This approach obviously takes the idea of fractals more seriously, since the characteristic of fractals, that

Color plates and captions

In the following we list the captions for the color plates of the following pages. Captions for the front and back cover images are at the end of this list. At the end of this section we include credits for the authors of the images.

Plates 1 to 7 show the stages of creating the fractal planet by adding random faults encircling the sphere.

Plate 1 The surface height variations (as indicated by color) after 10 faults shown both on the sphere and on a flat projection map.

Plate 2 The surface map after 20 faults where the colors indicate different land altitudes and ocean depths.

Plate 3 The surface map after 100 faults.

Plate 4 The surface map after 500 faults. Here the land has condensed into a single mass similar to the earth before continental drift.

Plate 5 The surface after more than 10000 faults with added polar caps. Additional faults would cause changes of detail but the overall appearance would remain the same.

Plate 6 This Brownian fractal planet mapped back onto the sphere (surface $D = 2.5$) with an earth-like water level and polar caps.

Plate 7 The same planet without water.

The differing visual effect of changing the fractal dimension D for the same landscape is shown in Plates 11 to 13. As D increases, the perceived surface roughness also increases. Also shown is fractal landscaping by scaling the surface height variations (relative to water level) by a power law in Plates 8 to 10.

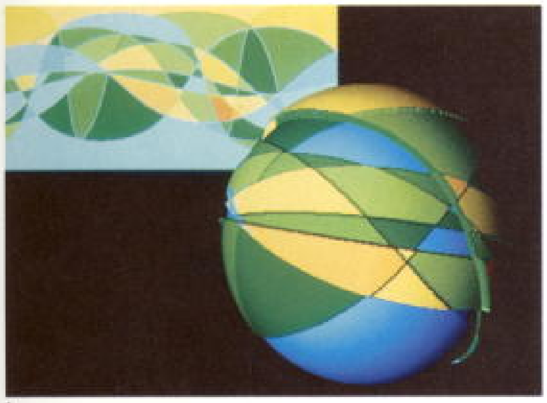
Plate 8 $D = 2.15$, the cube root of the original height is taken.

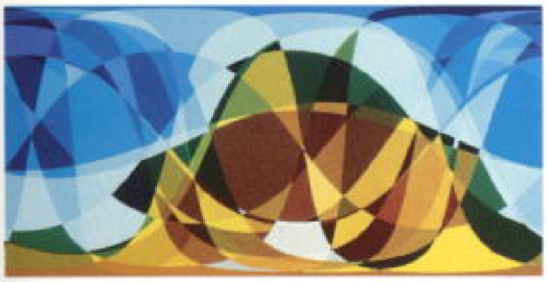
Plate 9 $D = 2.15$, original height.

Plate 10 $D = 2.15$, cubed height.

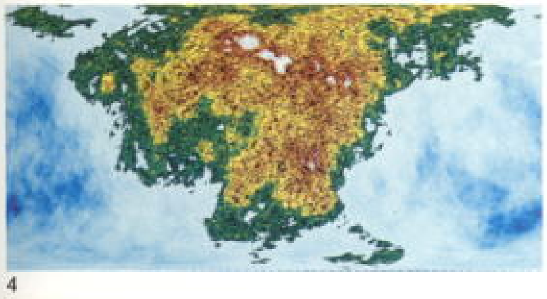
Plate 11 $D = 2.15$

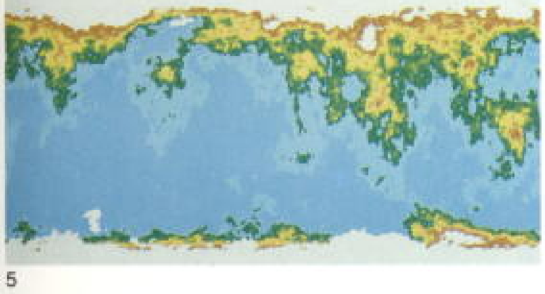
Plate 12 $D = 2.5$

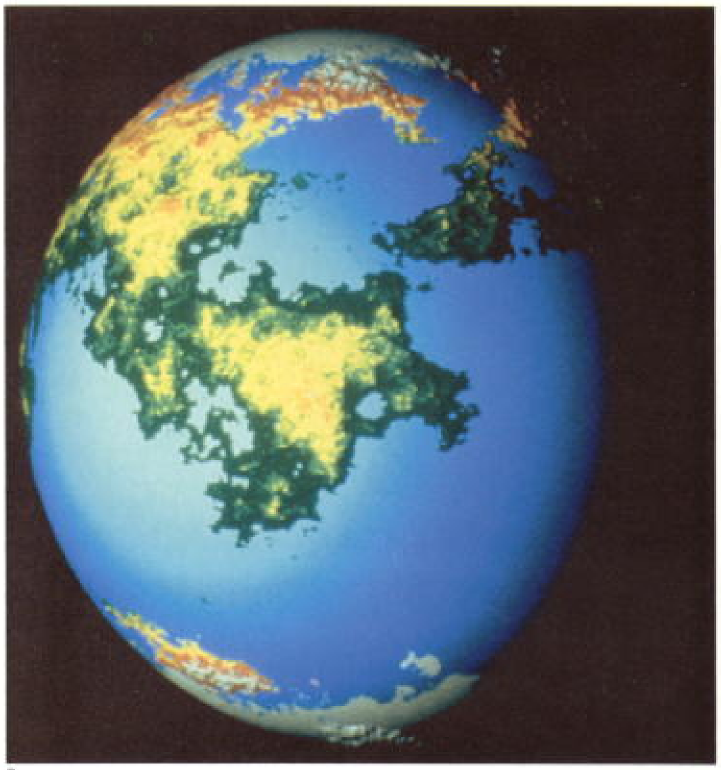

1


2

3


4


5


6

7

8

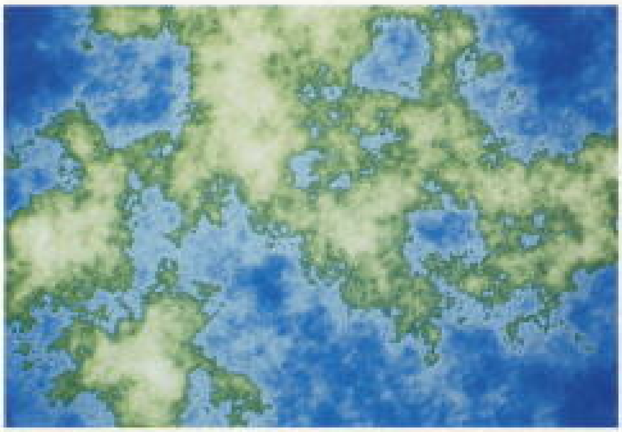
9

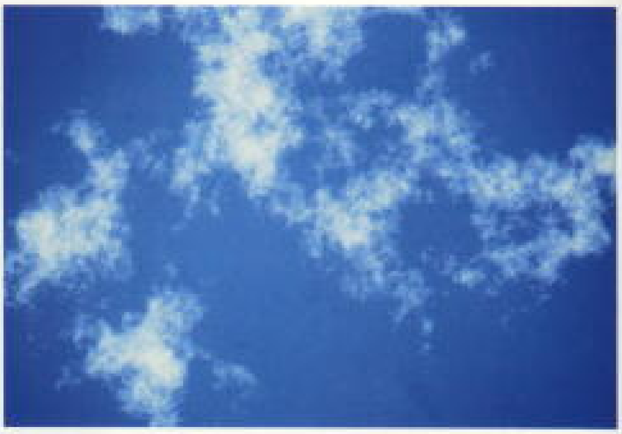
10

11

12

13


14

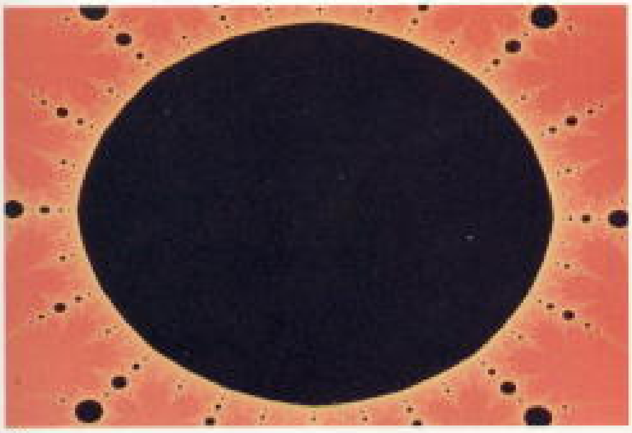

15

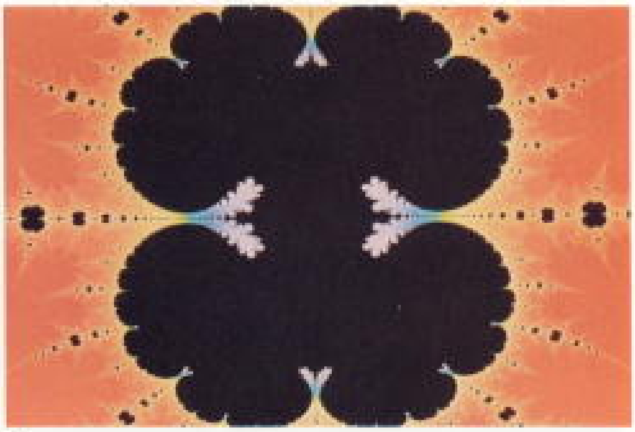
16a

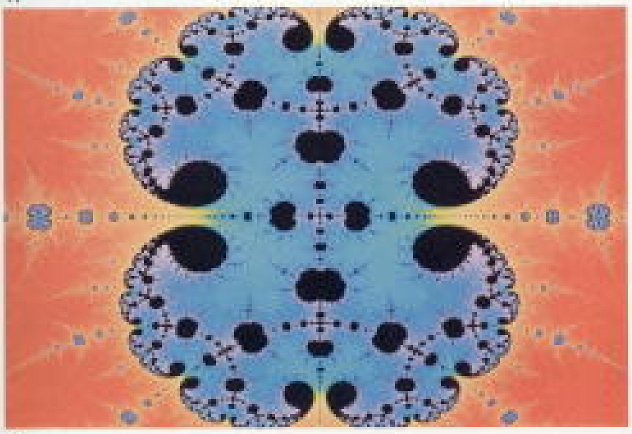
16b

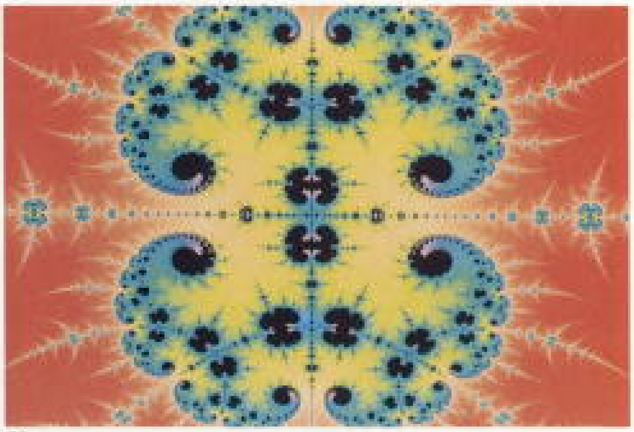
16c

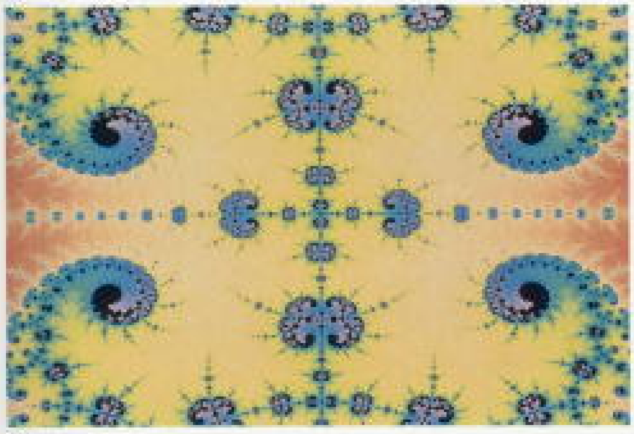
16d

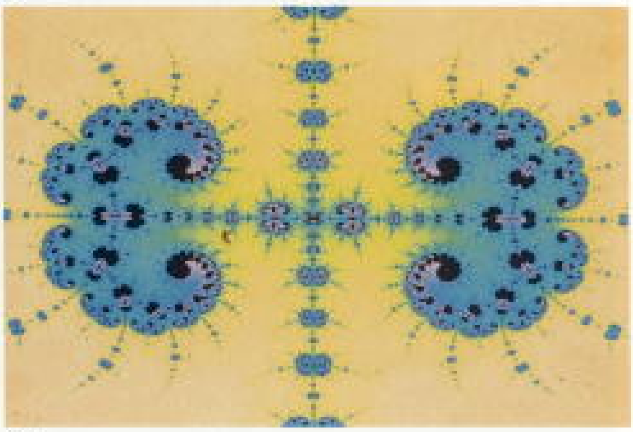

17

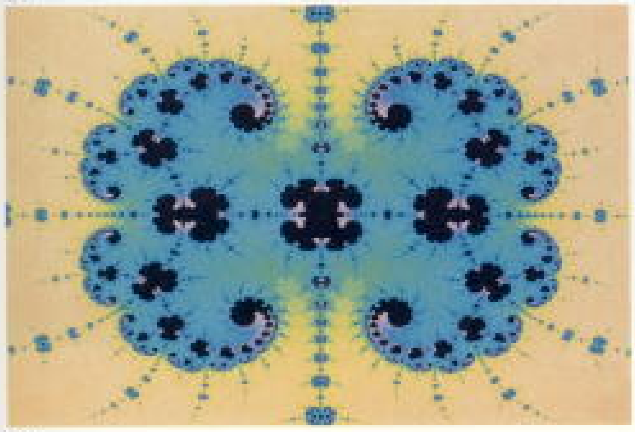

18

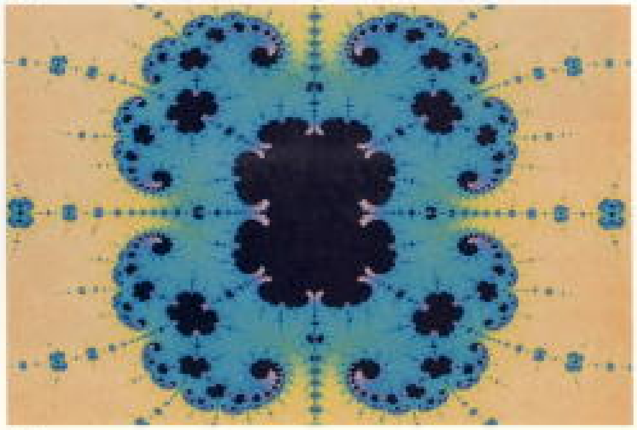

19

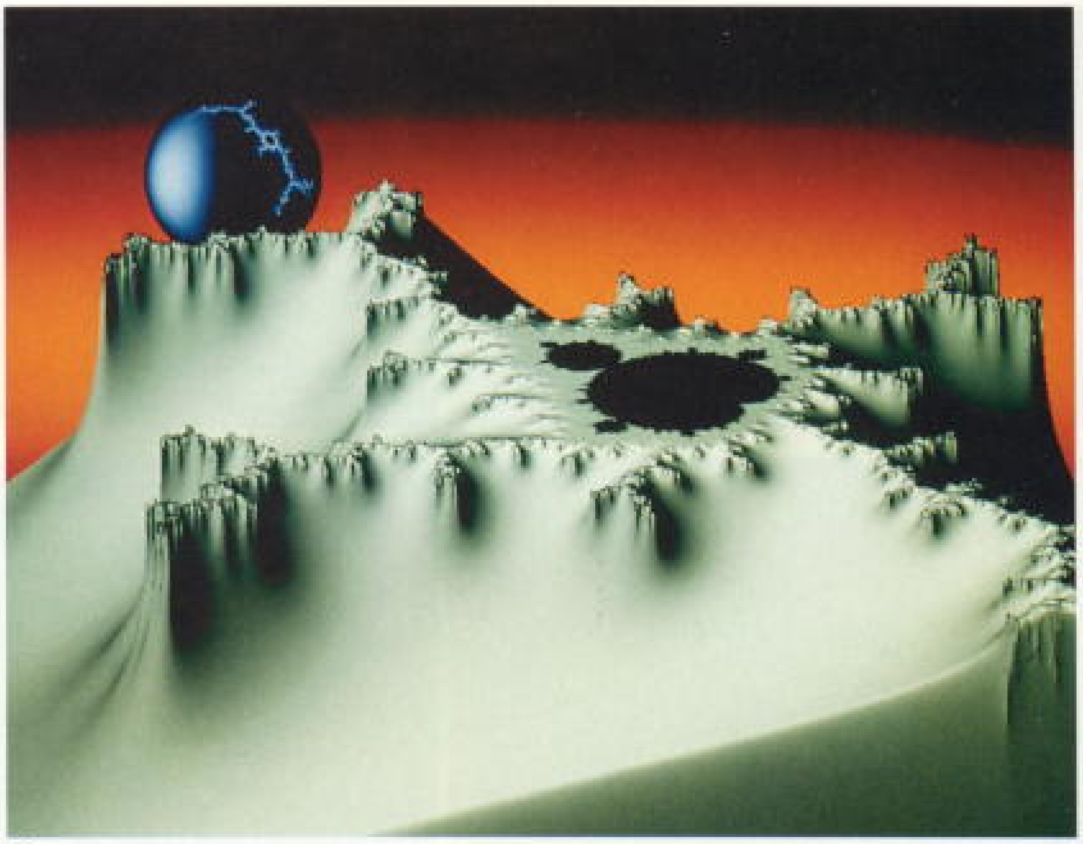

20


21


22


23


24

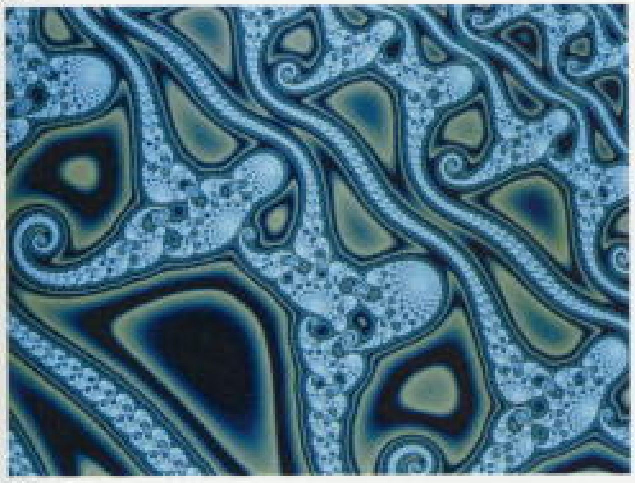


25

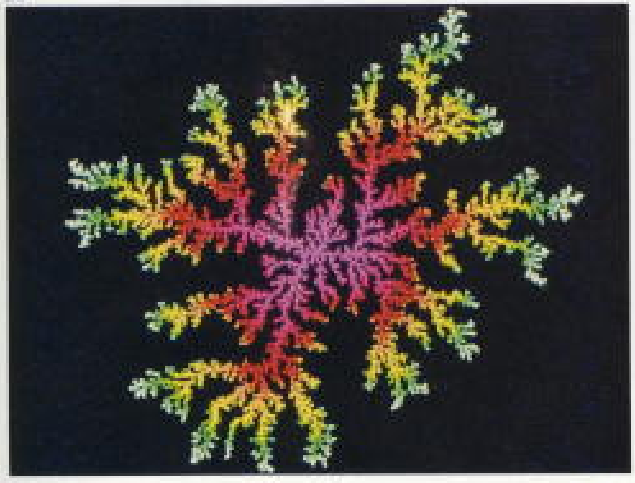
26

27

28


29

30


31

32

33

34

35

36

Plate 13 $D = 2.8$

Plate 14 Top view of fractal landscape with color coded heights. Blue shades indicate elevation below 0 and green, brown, white corresponds to increasing heights above 0. Data was generated using *MidPointFM2D()*.

Plate 15 Same fractal data as in Plate 14 with different color map applied to surface elevation yielding simulated clouds.

Plate 16 The Fourier synthesis method is applied to generate ocean waves, mountains and the sky in image (a), called *Drain*. Stochastic subdivision is used in the sunset picture (b), see Section 2.6 for methods used in (a) and (b). The *Lilac Twig* and the *Flower Field* in (c) and (d) are results of the L-systems explained in Appendix C.

The following 8 plates are illustrations of complex dynamics discussed in Chapter 3.

Plate 17 $\pi \cos z$. The basin of attraction of the attracting fixed point at $-\pi$.

Plate 18 $2.97 \cos z$. The basin about to explode.

Plate 19 $2.96 \cos z$. After the explosion. Note the small black regions representing components of the complement of the Julia set.

Plate 20 $2.945 \cos z$. Note how the black regions have moved as the parameter is varied.

Plate 21 $2.935 \cos z$. A magnification shows that there is structure within the black regions that is reminiscent of the original picture.

Plate 22 $2.92 \cos z$. Two components of the stable sets about to collide at the point $-\pi$.

Plate 23 $2.919 \cos z$. Formation of smaller black regions within a component.

Plate 24 $2.918 \cos z$. A saddle-node periodic point about to bifurcate.

Plate 25 A 3D rendering of the Continuous Potential Method (CPM/M). A version of the cover of "The Beauty of Fractals" [83]. The "moon" is a Riemann sphere, which carries the structure of a Julia set.

Plate 26 A 3D rendering of the Continuous Potential Method (CPM/M), (cf. Figure 4.21(4)). The "sky" was obtained by the Level Set Method (LSM/M) close to the boundary of the Mandelbrot set and is displayed using pixel zoom.

Plate 27 A 3D rendering of the Continuous Potential Method (CPM/J) showing a filled-in Julia set in red.

Plate 28 A 2D rendering of the Continuous Potential Method (CPM/M) (cf. Figure 4.22).

Plate 29 A 2D rendering of the Distance Estimator Method (DEM/M). Points of same color have same distance to Mandelbrot set.

Plate 30 A 3D-rendering of the Distance Estimator Method (DEM/M). Points of same height and color have same distance to Mandelbrot set.

Plate 31 Diffusion Limited Aggregation (DLA) about a single point.

Plate 32 Four views of the three-dimensional fern of Figure 5.14 . The IFS code consists of four three-dimensional affine transformations.

Plate 33 Landscape with chimneys and smoke. This figure and Figure 34 computed from the same IFS database; which consists of 57 affine transformations.

Plate 34 Zoom on smoke shown in Plate 33

Plate 35 A 3-dimensional cross section of $M_+ \cup M_-$ where $\text{Im}b = 0$, see Section 4.3.4.

Plate 36 "Moon rise in Mandelbrot sky".

Captions for the cover images :

Plate 37 (Front cover) "Lake Mandelbrot". The Continuous Potential Method is applied to a section of the Mandelbrot set and rendered as a height field. Coloring and an added random fractal sky enhance "realism".

Plate 38 (Back cover top left) "Black Forest in Winter", several composed images using Iterated Function Systems (IFS) .

Plate 39 (Back cover top right) 3D rendering of Distance Estimator Method (DEM/M) for a blow up at the boundary of the Mandelbrot set. Points of same height have same distance to Mandelbrot set.

Plate 40 (Back cover center left) The floating island of *Gulliver's Travels* whose mathematically obsessed occupants lost all contact with the world around them. Shown here is the center of the zenith view of a fractal landscape in a lake opening onto fractal clouds (in homage to Magritte).

Plate 41 (Back cover center right) A foggy fractally cratered landscape. Light scattering with varying intensity through the same $T(x, y, z)$ as Figure 1.6(c) produces a fractal cloud with its shadows on the fractal landscape of Figure 1.5.

Plate 42 (Back cover bottom) Fractal landscaping.

Credits for the color images

Besides the authors of the book several others have contributed to the pictures displayed on these pages and on the front and back cover of the book, which we gratefully acknowledge. Here is a list of the contributors and their images :

James Hanan, University of Regina
Arnaud Jacquin, Georgia Institute of Technology
Hartmut Jürgens, Universität Bremen
John P. Lewis, New York Institute of Technology
Francois Malassenet, Georgia Institute of Technology
John F. Mareda, Sandia National Laboratories
Gary A. Mastin, Sandia National Laboratories
Ken Musgrave, Yale University
Przemyslaw Prusinkiewicz, University of Regina
Laurie Reuter, George Washington University
Peter H. Richter, Universität Bremen
Marie-Theres Roeckerath-Ries, Universität Bremen
Alan D. Sloan, Georgia Institute of Technology
Peter A. Watterberg, Savannah River Laboratory

Plates 1 – 13	R.F. Voss
Plates 14 – 15	D. Saupe
Plate 16a	G.A. Mastin, P.A. Watterberg and J.F. Mareda
Plate 16b	J.P. Lewis
Plate 16c	J. Hanan and P. Prusinkiewicz
Plate 16d	P. Prusinkiewicz
Plates 17 – 24	R.L. Devaney
Plates 25 – 30	H. Jürgens, H.-O. Peitgen and D. Saupe
Plates 31	R.F. Voss
Plates 32 – 34	M.F. Barnsley, L. Reuter and A.D. Sloan
Plate 35	P. Richter and M.-T. Roeckerath-Ries
Plate 36	R.F. Voss
Front cover	H. Jürgens, H.-O. Peitgen and D. Saupe
Back cover top left	M.F. Barnsley, A. Jacquin, F. Malassenet, A.D. Sloan and L. Reuter
Back cover top right	H. Jürgens, H.-O. Peitgen and D. Saupe
Back cover center left	R.F. Voss
Back cover center right	R.F. Voss
Back cover bottom	B.B. Mandelbrot and F.K. Musgrave

there is detail at all scales, is carried out up to the screen resolution. The floating horizon method [92] is usually employed to render some coarse elevation data in a perspective view as in Figures 2.11 to 2.13 and 2.17 to 2.19. There each point is connected to its visible neighbors by a short line. In our case, where we have many more points, these lines generally can be omitted if each point is projected to one pixel on the screen and drawn. The hidden surface problem is solved most easily when the view direction is parallel to one coordinate plane. The farthest row of the data is drawn first, then the second last row, etc. Occasionally it may occur that two consecutive projected rows leave some pixels undefined in between them. For these pixels which are very easy to detect we must set some interpolated intermediate color and intensity. Most of the fascinating pictures of fractal terrain in Chapter 1 and of the potential surfaces for the Mandelbrot and Julia sets in Chapter 4 have been produced by such an algorithm (see the color plates). In the following we present a description of our extended version of the floating horizon method. It is sufficiently detailed as a guide for an implementation. Related algorithms are used in [38] and [41]. The method which is outlined below and refinements such as shadows and antialiasing will be the topic of our forthcoming paper [57].

2.7.3 The data and the projection

Let us assume for simplicity that the output of a fractal generator or of some other surface computation is presented as a square array of N by N elevation points. If

$$h : [0,1] \times [0,1] \to [0,1]$$

denotes a corresponding height function we may think of the data as a sample of h, namely

$$z_{i,j} = h(x_i, y_j)$$

where

$$x_i = \frac{i}{N-1}, y_j = \frac{j}{N-1}$$

and $i, j = 0, ..., N-1$. The surface to be rendered therefore is assumed to lie in the unit cube $[0,1]^3$. The projection of the data from the unit cube to the screen coordinates is most easily done by parallel projection such that one column of data elements is projected to one column on the screen. If the surface is to be viewed at an angle ϕ then the projection is given by

$$(x, y, z) \to (x, y \sin\phi + z\cos\phi).$$

Here $\phi = 0$ corresponds to a side view and $\phi = \frac{\pi}{2}$ yields a top view.

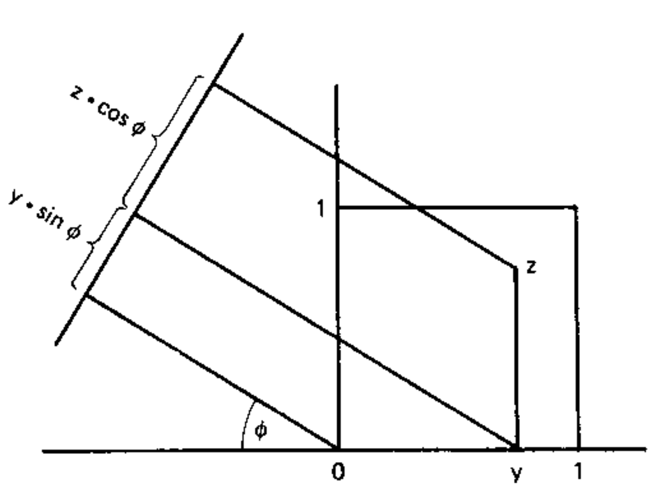


Fig. 2.20: Projection of data from unit cube to screen coordinates

2.7.4 A simple illumination model

For a local lighting model surface normals are essential. An approximate surface normal at the point

$$\vec{p} = (x_i, y_j, z_{i,j})$$

is given by

$$\vec{n} = (\frac{dx}{r}, \frac{dy}{r}, \frac{dz}{r})$$

where

$$\begin{aligned} dx &= z_{i,j} - z_{i+1,j} \\ dy &= z_{i,j-1} - z_{i,j} \\ dz &= \frac{1}{N-1} \end{aligned}$$

and

$$r = \sqrt{dx^2 + dy^2 + dz^2}.$$

This will work very well for smooth surfaces. For fractal surfaces averages over several such approximations for nearby points are more suitable.

Let $\vec{l}$ denote the unit length vector based at $\vec{p}$ and pointing towards a light source. Let further $\vec{r}$ be the reflected light vector, i.e.

$$\vec{r} = 2 \cos \theta \cdot \vec{n} - \vec{l}$$

where θ is the angle enclosed between $\vec{n}$ and $\vec{l}$. If α denotes the angle between the view vector

$$\vec{v} = (0, -\cos \phi, \sin \phi)$$

and the reflected vector $\vec{r}$, then a simple illumination model, which takes into account only ambient, diffuse and specularly reflected light, may be written as

$$I(\vec{p}) = \begin{cases} I_a, & \text{if} \quad \cos\theta < 0 \\ I_a + f_d \cos\theta + f_s \cdot (\cos\alpha)^b, & \text{otherwise} \end{cases} \tag{2.19}$$

Here I_a denotes the contribution of the ambient light, f_d and f_s are factors to be chosen as weights for the diffuse and specular components, and b is an exponent which controls the shininess of the surface. Satisfactory results can be obtained e.g. with

$$\begin{aligned} I_a &= 0.0 \, , \\ f_d &= 0.6 \, , \\ f_s &= 0.4 \, , \\ b &= 2.0 \, . \end{aligned}$$

For completeness let us remark that if the angle between the surface normal $\vec{n}$ and the view vector $\vec{v}$ is greater than $\frac{\pi}{2}$ in magnitude, then we are looking at the back side of the surface, and we can set the intensity to 0. In our algorithm *DisplayHeightField()* this case may only occur for the points in the first row. This simple model is a very crude one. For example, it ignores colors of the light and the surface. Better models are described in more detail in standard textbooks such as [39] or [92].

2.7.5 The rendering

Slices of the surface, which are parallel to the yz-plane, are projected onto a line on the screen, which is parallel to the screen y-axis. Therfore we can render these lines one by one and independently of each other. Define

$$P(y, z) = int\,[(N-1)(y \sin\phi + z \cos\phi)]$$

as the integer part of the y-value of the projection of a point (x, y, z) in pixel units. Then the algorithm for the rendering of all N columns of data looks as follows (see *DisplayHeightField()*). Here *Intens(i, k)* denotes the intensity calculated for the pixel *(i, k)* of the screen, while *I(x, y, z)* denotes the corresponding intensity for a point (x, y, z) on the surface.

In the first loop all intensities in the i-th column are initialized to 0. Then the data points in the i-th column are projected one by one, starting at the front and moving to the back. The variable *horizon* keeps track of the current maximum of all projected y-values, and this explains the names of both the variable and

```
ALGORITHM DisplayHeightField (Z, N, phi)
Title          Computer graphics for surfaces given by array of elevation

Arguments      Z[][]          doubly indexed square array of heights
               N              number of elements in each row/column of Z
               phi            view angle
Globals        Intens[][]     output array for intensities of size at least N by $\sqrt{2}$N
Variables      i, j, k        integers
               p, p0, p1      projected heights, screen y-coordinates
               x[]            N real x-values
               y[]            N real y-values
               horizon        integer (the floating horizon)
               h              real parameter for interpolation
Functions      P(y,z) = int [(N-1) * (y * sin(phi) + z * cos(phi)]
               I(x,y,z) = intensity function, see Eqn. (2.19)

BEGIN
    FOR i = 0 TO N-1 DO
        x[i] := i / (N - 1)
        y[i] := i / (N - 1)
    END FOR
    FOR i = 0 TO N-1 DO
        FOR k = 0, 1, 2, ... DO
            Intens[i][k] := 0
        END FOR
        p0 := P(y[0],z[i][0])
        Intens[0][p0] := I(x[i],y[0],z[i][0])
        horizon := p0
        FOR j = 1 TO N-1 DO
            p1 := P(y[j],z[i][j])
            IF p1 > horizon THEN
                Intens[i][p1] := I (x[i],y[j],z[i][j])
                p := p1 - 1
                WHILE p > horizon DO
                    h := (p - p0) / (p1 - p0)
                    Intens[i][p] := (1-h) * I(x[i],y[j-1],z[i][j-1]) + h * Intens[i][p1]
                    p := p - 1
                END WHILE
                horizon := p1
            END IF
            p0 := p1
        END FOR
    END FOR
END
```

the rendering method. If a point is projected below this horizon, it is not visible, and the normal and intensity calculations can be saved. Otherwise, the correct intensity is entered at the projected screen coordinates and the value of the horizon is updated. A special case arises when there are one or more pixels left out between the old and the new horizon. At these locations an interpolated intensity has to be applied. In the above scheme linear interpolation is employed, which corresponds to Gouraud shading. Alternatively one could interpolate surface normals and recompute intensities for the intermediate points, a procedure known as Phong shading.

It depends very much on the data and the projection angle ϕ chosen, which portion of the screen coordinate system actually should be displayed. In the worst case ($\phi = \frac{\pi}{4}$ and data occurring at $(x, 0, 0)$ as well as at $(x, 1, 1)$) only a viewport containing at least N by $\sqrt{2}N$ pixels will contain the whole surface. To get aesthetically pleasing results one obviously has to experiment with the projection and introduce proper scalings in the x and y screen coordinates.

2.7.6 Data manipulation

An alternative way to match the viewport to the diplayed object is to change the data instead. For example, a linear transformation

$$z_{i,j} = a \cdot h(x_i, y_j) + b$$

will shift the picture up or down and may scale the heights by the factor a. This may be desirable for other reasons as well. Often the height variations in the data are too strong or not pronounced enough for a nice picture. A linear scaling may help in this case. But also nonlinear scalings are of interest as color Plates 8 and 10 ($D = 2.15$ cubed height) demonstrates. In the case of the electrostatic potentials of the Mandelbrot set in Chapter 4 only a nonlinear scaling of the potential was able to change the initially rather dull looking images into something dramatic.

Another useful manipulation may be to pull down the heights of points near the front and/or the back of the surface in order to eliminate the impression of a surface that has been artificially cut off in the front and the back.

2.7.7 Color, anti-aliasing and shadows

The surface may have different colors at different places. E.g. color may be linked to height and possibly also to surface normals as seen in the color plates. It is easy to add color to our little rendering system just by multiplying the color of each projected point with the intensity as calculated in the algorithm. However, it must be noted, that this is straight forward only in the case that a full color (24 bit) graphics system is eventually used for display. Some restrictions apply for smaller systems with color lookup tables.

The algorithm *DisplayHeightField()* can easily be extended to include anti-aliasing by means of supersampling. The principle is to first interpolate (multilinear or with splines) the surface to obtain a representation in a higher resolution. Then the algorithm is applied to the "inflated" surface and finally pixel averaging results in the anti-aliased image. This process can take advantage of the piecewise linear structure in the interpolated surface, thus, the computational burden of anti-aliasing is not as severe as it seems. In addition, distance fading as in [38] or simulated fog as in the color plates may be employed to further enhance picture quality.

The same algorithm that eliminates the hidden surfaces in our rendering may be used to calculate shadows from several light sources. This is the case because a shadow reflects nothing but the visibility from a given light source. Due to the simple structure of the algorithm only infinitely far light sources are admissible. In addition the vectors pointing to the light sources must be parallel to the xz-plane or the yz-plane. A complication arises : The rendering of the different data columns is no longer independent. Points in one column may cast a shadow on points of other columns. Thus, one complete row of data has to be rendered simultaneously. The cover picture was rendered with two light sources, one from behind the view point and the other from the right. The shadows from the second light source are clearly visible.

2.7.8 Data storage considerations

The storage requirements for the data may be quite stringent. For example, a common file may contain 1024 by 1024 floating point numbers, which typically amount to 4 mega bytes of storage. To reduce the size of these large files, several techniques may be of value. The first obvious one is to store the programs that generate the data in place of the data itself. This is a good idea for fractal surfaces generated by one of the cheaper methods such as midpoint displacement. However, for potentials of the Mandelbrot set and related images it is not advisable since the generation of a single set of data may take too long to be done twice or more often for several renderings.

To reduce the size of the data files we first note that we do not need the full range of precision and also of the exponents offered by floating point numbers. Thus, one elevation point may be conveniently stored as an unsigned 16 bit integer. This will reduce the above mentioned typical 4 mega bytes to only 2 mega bytes per file without decreasing the picture quality. But this still seems to be too much if one intends to keep many data files on hand.

The next step for the data reduction exploits the continuity of the height function or even its differentiability, if applicable as in the 3D renderings of Chapter 4. Instead of storing the absolute values of the height function we may, for each row j of data, store only the first elevation $z_{0,j}$ and the differences

$$\Delta^1_{i,j} = z_{i,j} - z_{i-1,j} \quad \text{for} \quad i = 1, ..., N-1 \; .$$

These first differences will be much smaller than the values $z_{i,j}$ thus, they may be stored in fewer bits per number. Still the original data may be reconstructed via

$$z_{i,j} = z_{i-1,j} + \Delta^1_{i,j} \quad \text{for} \quad i = 1, ..., N-1 \; .$$

If the height function h is also differentiable then the second differences

$$\Delta^2_{i,j} = \Delta^1_{i,j} - \Delta^1_{i-1,j} \quad \text{for} \quad i = 2, ..., N-1$$

or, more explicitly

$$\Delta^2_{i,j} = z_{i,j} - 2\,z_{i-1,j} + z_{i-2,j} \quad \text{for} \quad i = 2, ..., N-1$$

can be expected to be even smaller in magnitude than the first differences. Thus, we may be able to store the numbers

$$z_{0,j}, \Delta^1_{1,j}, \Delta^2_{2,j}, \Delta^2_{3,j}, ..., \Delta^2_{N-1,j}$$

even more efficiently. The reconstruction of the j-th row of the original data then is achieved with one or two additions per point via

$$z_{1,j} = z_{0,j} + \Delta^1_{1,j}$$

and

$$\begin{aligned} \Delta^1_{i,j} &= \Delta^1_{i-1,j} + \Delta^2_{i,j} \\ z_{i,j} &= z_{i-1,j} + \Delta^1_{i,j} \end{aligned}$$

for $i = 2, ..., N-1$. For many of the smooth surfaces of Chapter 4 this technique resulted in a requirement of only roughly 8 bits per data element thus reducing the initial 4 mega bytes to only about one mega byte.

There is even more structure in the first and second differences which can be used to further reduce the size of the data files. For this purpose data compression techniques such as simple run length encoding, or more advanced methods such as Huffman encoding [89] or Lempel-Ziv-Welch encoding [106] as implemented in the UNIX *compress* utility may result in an additional saving. Finally, one may modify these system compression techniques to better fit our case of interest.

2.8 Random variables and random functions

Random variables and random functions are the two major building blocks used in random fractals. Here we review some of the basic definitions (see e.g. [8,107]). [44] is another elementary text for probability theory.

Sample space. An abstract representation for a trial and its outcomes such as one throw of a die or ideally also one call to a (pseudo) random number generator is the *sample space*. Each distinguishable and indecomposable outcome, or simple event, is regarded as a point in sample space S. Thus, for a typical random number generator the sample space would consist of all integers between 0 and a very large number such as $2^{31} - 1$. Every collection of simple events i.e. a set of points in S is called an *event*.

Probability. For every event E in the sample space S, i.e. $E \subset S$ we assign a non-negative number, called the *probability* of E denoted by $P(E)$ so that the following axioms are satisfied:

(a) for every event E, $P(E) \geq 0$

(b) for the certain event, $P(S) = 1$

(c) for mutually exclusive events E_1, E_2, $P(E_1 \cup E_2) = P(E_1) + P(E_2)$

Random variables. A *random variable* X is a function on a sample space S with two properties

(a) the values it assumes are real numbers,

(b) for every real number x the probability that the value of the function is less than or equal to x can be calculated, and is denoted by $P(X \leq x)$.

In the example of dice or random number generators the points in the sample space are themselves numerical outcomes, and the outcome is itself a random variable.

Probability density function. Suppose that X is a random variable and there is a function $f(x)$ such that

(a) $f(x) \geq 0$.

(b) $f(x)$ has at most a finite number of discontinuities in every finite interval on the real line.

(c) $\int_{-\infty}^{\infty} f(x)\,dx = 1$.

(d) For every interval $[a, b]$

$$P(a \le X \le b) = \int_a^b f(x)\,dx.$$

Then X is said to be a *continuous random variable* with *probability density function* $f(x)$.

Distribution function. A useful picture of the spread of probability in a distribution is provided by the *distribution function* $F(x)$, which is defined as

$$F(x) = P(X \le x).$$

From its definition, $0 \le F(x) \le 1$, $F(-\infty) = 0$, $F(\infty) = 1$, $P(a \le X \le b) = F(b) - F(a) \ge 0$. For a continuous random variable with probability density $f(x)$ we have

$$F(x) = \int_{-\infty}^{x} f(t)\,dt.$$

Uniform distribution. A continuous random variable X is said to have a *uniform distribution* over an interval $[a, b]$, if its probability density function $f(x)$ satisfies

$$f(x) = \begin{cases} \frac{1}{b-a}, & \text{if } a \le x \le b \\ 0, & \text{otherwise} \end{cases}$$

If $a \le x \le b$, then

$$P(X \le x) = \frac{x-a}{b-a}.$$

Normal distribution. A continuous random variable X is said to have a *normal distribution* with parameters μ and $\sigma > 0$ if its probability density function $f(x)$ is

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}.$$

Such a random function is said to be distributed $N(\mu, \sigma)$. The graph of f has the familiar Gaussian bell shape centered around μ. The routine *Gauss()* in Section 2.2 is constructed to approximate a random variable with a distribution $N(0, 1)$. Its distribution function $F(x)$ is tabulated in most handbooks on statistics.

Expectation. If X is a continuous random variable with distribution function $f(x)$ then the *mean* or the *expectation* of X, written as $E(X)$ is defined as

$$E(X) = \int_{-\infty}^{\infty} x f(x)\,dx$$

provided that

$$\int_{-\infty}^{\infty} |x| f(x)\,dx$$

converges. Otherwise, $E(X)$ is said not to exist. The name has its justification in the weak law of large numbers, which states, roughly speaking, that if $X_1, X_2, \ldots, X_n$ is a random sample from a distribution for which $E(X)$ exists, then the average

$$\overline{X} = \sum_{i=1}^{n} \frac{X_i}{n}$$

converges in probability to the mean $E(X)$. For the uniformly distributed random variable defined two paragraphs above we have of course $E(X) = \frac{a+b}{2}$, whereas for a random variable with a normal distribution $N(\mu, \sigma)$ we obtain $E(X) = \mu$. The expectation E is a linear function in the sense that for two random variables X and Y and real numbers a and b we have $E(aX + bY) = aE(X) + bE(Y)$. Moreover, if $g(x)$ is a real function, then $Y = g(X)$ is another random variable and its expectation is

$$E(g(X)) = \int_{-\infty}^{\infty} g(x) f(x)\, dx.$$

Variance. Generally a random variable is not adequately described by just stating its expected value. We should also have some idea of how the values are dispersed around the mean. One such measure is the *variance*, denoted by var X and defined as

$$\operatorname{var} X = E[(X - E(X))^2].$$

The positive square root of the variance is known as the *standard deviation* of X, and is denoted by σ. We also have

$$\operatorname{var} X = E(X^2) - E(X)^2.$$

The variance of a random variable which is uniformly distributed over $[a, b]$ is easily calculated as

$$\operatorname{var} X = \frac{1}{12}(b - a)^2$$

and for random variables with normal distribution $N(\mu, \sigma)$ the variance is σ^2. If two random variables X, Y are independent, i.e.

$$P(X \leq x \text{ and } Y \leq y) = P(X \leq x) + P(Y \leq y)$$

for all $x, y \in \mathbf{R}$, then we have for all $a, b \in \mathbf{R}$

$$\operatorname{var}(aX + bY) = a^2 \operatorname{var} X + b^2 \operatorname{var} Y.$$

Covariance and correlation. A measure which associates two random variables is given by the *covariance*. The covariance of X, Y, written $C(X, Y)$, is defined as

$$C(X,Y) = E[(X - E(X))(Y - E(Y))].$$

If the two random variables are independent, then their covariance is 0, of course. If units of X or Y are changed then the covariance also changes. This defect is removed by considering the *correlation coefficient,* which is the covariance divided by both standard deviations.

Random functions. Let T be a subset of the real line. By a *random function* of an argument $t \in T$ we mean a function $X(t)$ whose values are random variables. Thus, a random function X is a family of random variables $X(s)$, $X(t)$, etc. Formally, a random function is considered to be specified if for each element $t \in T$ we are given the distribution function

$$F_t(x) = P(X(t) \leq x)$$

of the random variable $X(t)$, and if also for each pair of elements $t_1, t_2 \in T$ we are given the distribution function

$$F_{t_1,t_2}(x_1, x_2) = P(X(t_1) \leq x_1 \text{ and } X(t_2) \leq x_2)$$

of the two-dimensional random variable $Z(t_1, t_2) = (X(t_1), X(t_2))$, and so on for all finite dimensional random variables

$$Z(t_1, t_2, \ldots, t_n) = (X(t_1), X(t_2), \ldots, X(t_n))$$

Stationarity. The random function will be called *stationary,* if all the finite dimensional distribution functions defined above remain the same when all points $t_1, t_2, \ldots, t_n$ are shifted along the time axis, i.e. if

$$F_{t_1+s,t_2+s,\ldots,t_n+s}(x_1, x_2, \ldots, x_n) = F_{t_1,t_2,\ldots,t_n}(x_1, x_2, \ldots, x_n)$$

for any $t_1, t_2, \ldots, t_n$ and s. The physical meaning of the concept of stationarity is that $X(t)$ describes the time variation of a numerical characteristic of an event such that none of the observed macroscopic factors influencing the occurrence of the event change in time.

Stationary increments. Fractional Brownian motion is not a stationary process, but its increments

$$X(t + s) - X(t)$$

are. In general, we say that the random function $X(t)$ has *stationary increments,* if the expected value of the increment is 0

$$E[X(t + s) - X(t)] = 0$$

and if the *mean square increments*

$$E[\,|X(t + s) - X(t)|^2\,]$$

do not depend on time t.