

Chapter 13

Fractals and Kinetic Growth Models

©2004 by Harvey Gould, Jan Tobochnik, and Wolfgang Christian
17 December 2004

We introduce the concept of fractal dimension and discuss several processes that generate fractal objects.

13.1 The Fractal Dimension

One of the more interesting geometrical properties of objects is their shape. As an example, we show in Figure 13.1 a spanning cluster generated at the percolation threshold. Although the visual description of such a cluster is subjective, such a cluster can be described as ramified, airy, tenuous, and stringy, and would not be described as compact or space-filling.

In the 1970's a new *fractal* geometry was developed by Mandelbrot and others to describe the characteristics of ramified objects. One quantitative measure of the structure of these objects is their *fractal dimension* D . To define D , we first review some simple ideas of dimension in ordinary Euclidean geometry. Consider a circular or spherical object of mass M and radius R . If the radius of the object is increased from R to $2R$, the mass of the object is increased by a factor of 2^2 if the object is circular, or by 2^3 if the object is spherical. We can express this relation between mass and the radius or a length as

$$M(R) \sim R^D, \quad (\text{mass dimension}) \quad (13.1)$$

where D is the dimension of the object. Equation (13.1) implies that if the linear dimensions of an object are increased by a factor of b while preserving its shape, then the mass of the object is increased by b^D . This mass-length scaling relation is closely related to our intuitive understanding of dimension.

If the dimension of the object, D , and the dimension of the Euclidean space in which the object is embedded, d , are identical, then the mass density $\rho = M/R^d$ scales as

$$\rho(R) \propto M(R)/R^d \sim R^0. \quad (13.2)$$

An example of a two-dimensional object is shown in Figure 13.2. An object whose mass-length relation satisfies (13.1) with $D = d$ is said to be *compact*.

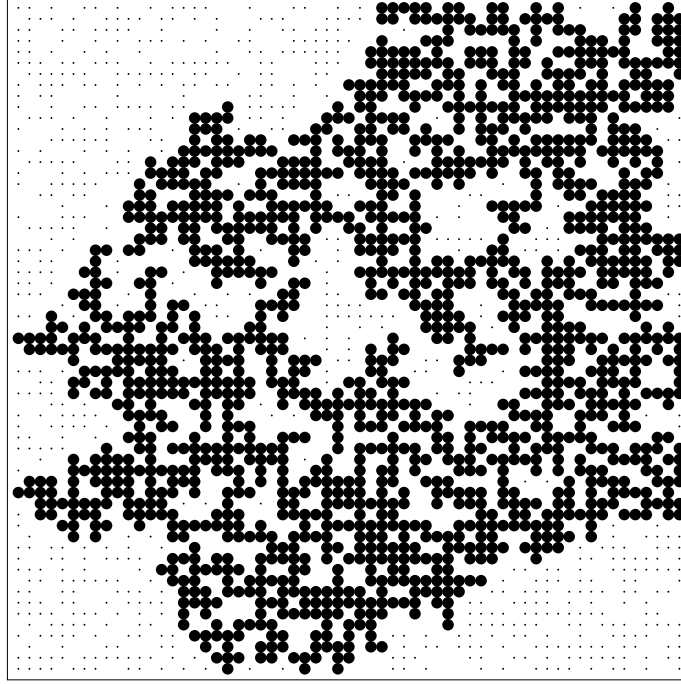


Figure 13.1: Example of a percolation cluster generated at $p = 0.5927$ on a $L = 61$ square lattice. Occupied sites that are not part of the spanning cluster are shown as points; unoccupied sites are not shown. [xx need a better figure xx]

Equation (13.1) can be used to define the fractal dimension. We denote objects as fractals if they satisfy (13.1) with a value of D different from the spatial dimension d . If an object satisfies (13.1) with $D < d$, its density is not the same for all R , but scales as

$$\rho(R) \propto M/R^d \sim R^{D-d}. \quad (13.3)$$

Because $D < d$, a fractal object becomes less dense at larger length scales. The scale dependence of the density is a quantitative measure of the ramified or stringy nature of fractal objects. That is, one characteristic of fractal objects is that they have holes of all sizes.

The percolation cluster shown in Figure 13.1 is an example of a *random* or statistical fractal because the mass-length relation (13.1) is satisfied only on the average, that is, only if the quantity $M(R)$ is averaged over many different origins in a given cluster and over many clusters.

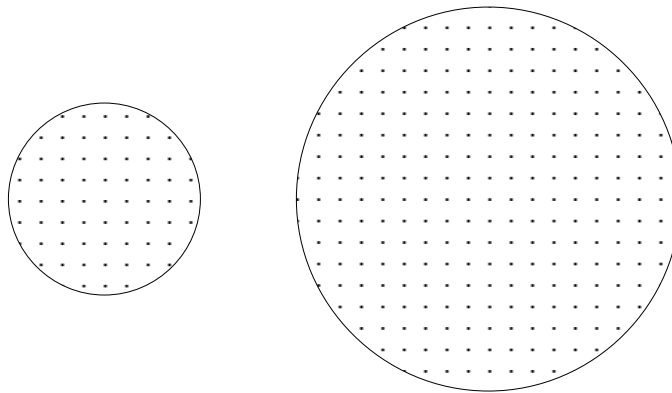


Figure 13.2: The number of dots per unit area in each circle is uniform. How does the total number of dots (mass) vary with the radius of the circle?

In physical systems, the relation (13.1) does not extend over all length scales, but is bounded by both upper and lower cut-off lengths. For example, a lower cut-off length is provided by a microscopic distance such as a lattice spacing or the mean distance between the constituents of the object. In computer simulations a maximum length usually is provided by the finite system size. The presence of these cut-offs complicates the determination of the fractal dimension.

Another important characteristic of fractal objects is that they look the same over a range of length scales. This property of self-similarity or scale invariance means that if we take part of a fractal object and magnify it by the same magnification factor in all directions, the magnified picture is indistinguishable from the original.

In Problem 13.1 we compute the fractal dimension of percolation clusters using straightforward Monte Carlo methods. Remember that data extending over several decades is required to obtain convincing evidence for a power law relationship between M and R and to determine accurate estimates for the fractal dimension. Hence, conclusions based on the limited simulations posed in the problems need to be interpreted with caution. A renormalization group method for estimating the fractal dimension is considered in Problem 13.2.

Problem 13.1. The fractal dimension of percolation clusters

- a. Generate a site percolation configuration on a square lattice with $L = 61$ at $p = p_c \approx 0.5927$. Why might it be necessary to generate a number of configurations before a spanning cluster is obtained? Obtain a feel for the ramified nature of the spanning cluster by printing a configuration and marking the positions of the sites in the spanning cluster as in Figure 13.1. Does the spanning cluster have many dangling ends?
- b. Choose a point on the spanning cluster and count the number of points in the spanning cluster $M(b)$ within a square of area b^2 centered about that point. Then double b and count the number of points within the larger box. Repeat this procedure until you can estimate the b -dependence of the number of points. Can you repeat this procedure indefinitely? Use the b -dependence of

$M(b)$ to estimate D according to the definition, $M(b) \sim b^D$ (see (13.1)). Choose another point in the cluster and repeat this procedure. Are your results similar? A better estimate for D can be found by averaging $M(b)$ over several origins in each spanning cluster and averaging over many spanning clusters.

- c. If you have not done Problem 12.10a, compute D by determining the mean size (mass) M of the spanning cluster at $p = p_c$ as a function of the linear dimension L of the lattice. Consider $L = 11, 21, 41$, and 61 and estimate D from a log-log plot of M versus L .

***Problem 13.2.** Renormalization group calculation of the fractal dimension

Compute $\langle M^2 \rangle$, the average of the square of the number of occupied sites in the spanning cluster at $p = p_c$, and the quantity $\langle M'^2 \rangle$, the average of the square of the number of occupied sites in the spanning cluster on the renormalized lattice of linear dimension $L' = L/b$. Because $\langle M^2 \rangle \sim R^{2D}$ and $\langle M'^2 \rangle \sim (R/b)^{2D}$, we can obtain D from the relation $b^{2D} = \langle M^2 \rangle / \langle M'^2 \rangle$. Choose the length rescaling factor to be $b = 2$ and adopt the same blocking procedure as was used in Section 12.6. An average over ten spanning clusters for $L = 16$ and $p = 0.5927$ is sufficient for qualitative results.

In Problems 13.1 and 13.2 we were interested only in the properties of the spanning clusters. For this reason, our algorithm for generating percolation configurations by randomly occupying each site is inefficient because it generates many clusters. There is a more efficient way of generating single percolation clusters due independently to Hammersley, Leath, and Alexandrowicz. This algorithm, commonly known as the Leath algorithm, is equivalent to the following steps (see Figure 13.3):

1. Occupy a single seed on the lattice. The nearest neighbors (four on the square lattice) of the seed represent the *perimeter* sites.
2. For each perimeter site, generate a random number r in the unit interval. If $r \leq p$, the site is occupied and added to the cluster; otherwise the site is not occupied. In order that sites be unoccupied with probability $1 - p$, these sites are not tested again.
3. For each site that is occupied, determine if there are any new perimeter sites, that is, untested neighbors. Add the new perimeter sites to the perimeter list.
4. Continue steps 2 and 3 until there are no untested perimeter sites to test for occupancy.

Class `Cluster` implements this algorithm and computes the number of occupied sites within a radius r of the seed particle. The seed site is placed at the center of a square lattice. Two one-dimensional arrays, `pxs` and `pys`, store the x and y positions of the perimeter sites. The status of a site is stored in the array `s` with $s(x, y) = 1$ an occupied site, $s(x, y) = 2$ a perimeter site, and $s(x, y) = -1$ a site that has already been tested and not occupied, and $s(x, y) = 0$ an untested and unvisited site. To avoid checking for the boundaries of the lattice, we add an extra row and column at the boundary and set these sites equal to -1 .

Listing 13.1: Class for generating and analyzing a cluster.

```
/* Creates percolation cluster with probability p and computes mass distribution */
```

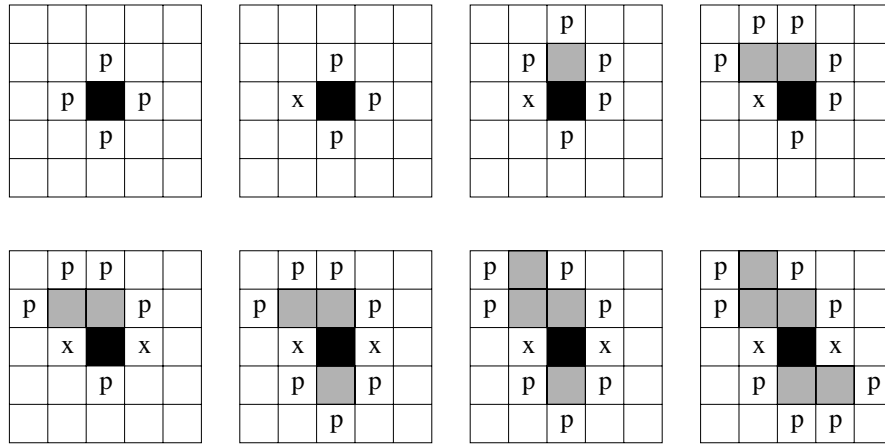


Figure 13.3: An example of the growth of a percolation cluster. Sites are occupied with probability $p = 0.5927$. Occupied sites are represented by a shaded square, perimeter sites are labeled by ‘p,’ and tested unoccupied sites are labeled by ‘x.’ Because the seed site is occupied but not tested, we have represented it differently than the other occupied sites. The perimeter sites are chosen at random.

```

package org.opensourcephysics.sip.ch13.cluster ;
import org.opensourcephysics.display.*;
import java.awt.*;

public class Cluster implements Drawable{

    public int s [][];
    public int xs [], ys [], pxs [], pys [];
    public int L;
    public double p; //site occupation probability
    int occupiedNumber;
    int perimeterNumber;
    int nx[] = {1,-1,0,0};
    int ny[] = {0,0,1,-1};
    double mass[];

    public void initialize () {
        s = new int[L+2][L+2];
        xs = new int[L*L]; //location of occupied sites
        ys = new int[L*L];
        pxs = new int[L*L]; //location of perimeter sites
        pys = new int[L*L];
        for(int i = 0; i < L+2; i++) {
            s[0][i] = -1; // don't occupy edge sites
            s[L+1][i] = -1;
        }
    }
}

```

```

        s[i][0] = -1;
        s[i][L+1] = -1;
    }
    xs[0] = 1 + (L/2);
    ys[0] = xs[0];
    s[xs[0]][ys[0]] = 1;    // occupy center site
    occupiedNumber = 1;
    for(int n = 0; n < 4; n++) { // perimeter sites
        pxs[n] = xs[0] + nx[n];
        pys[n] = ys[0] + ny[n];
        s[pxs[n]][pys[n]] = 2;
    }
    perimeterNumber = 4;
}

public void step() {
    if(perimeterNumber > 0) {
        int perimeter = (int)(Math.random()*perimeterNumber);
        int x = pxs[perimeter];
        int y = pys[perimeter];
        perimeterNumber--;
        pxs[perimeter] = pxs[perimeterNumber];
        pys[perimeter] = pys[perimeterNumber];
        if(Math.random() < p) { //occupy site
            s[x][y] = 1;
            xs[occupiedNumber] = x;
            ys[occupiedNumber] = y;
            occupiedNumber++;
            for(int n = 0; n < 4; n++) { // find new perimeter sites
                int px = x + nx[n];
                int py = y + ny[n];
                if(s[px][py] == 0) {
                    pxs[perimeterNumber] = px;
                    pys[perimeterNumber] = py;
                    s[px][py] = 2;
                    perimeterNumber++;
                }
            }
        }
        else
            s[x][y] = -1;
    }
}

public void massDistribution() {
    mass = new double[L];
    double xcm = 0;
    double ycm = 0;
    for(int n = 0; n < occupiedNumber; n++) {
        xcm += xs[n];

```

```

    ycm += ys[n];
}
xcm /= occupiedNumber;
ycm /= occupiedNumber;
for(int n = 0; n < occupiedNumber; n++) {
    double dx = xs[n] - xcm;
    double dy = ys[n] - ycm;
    int r = (int)Math.sqrt(dx*dx + dy*dy);
    if((r > 1) && (r < L)) mass[r]++;
}
}

public void draw (DrawingPanel myWorld, Graphics g) {
    if(s == null) return;
    int sizeX = Math.abs(myWorld.xToPix(0.8) - myWorld.xToPix(0));
    int sizeY = Math.abs(myWorld.yToPix(0.8) - myWorld.yToPix(0));
    for(int i = 1; i < L+1; i++)
        for(int j = 1; j < L+1; j++) {
            int xpix = myWorld.xToPix(i) - sizeX;
            int ypix = myWorld.yToPix(j) - sizeY;
            switch (s[i][j]) {
                case 0:
                    g.setColor(Color.black);
                    g.fillRect (xpix + sizeX/2,ypix + sizeY/2,1,1);
                    break;
                case 1:
                    g.setColor(Color.blue);
                    g.fillOval (xpix,ypix,sizeX,sizeY);
                    break;
                case -1:
                    g.setColor(Color.yellow);
                    g.fillOval (xpix,ypix,sizeX,sizeY);
                    break;
                case 2:
                    g.setColor(Color.green);
                    g.fillOval (xpix,ypix,sizeX,sizeY);
                    break;
            }
        }
    }
}

```

We use the growth algorithm in Problem 13.3 to generate a spanning cluster at the percolation threshold. The fractal dimension is determined by counting the number of sites M in the cluster within a distance r of the center of mass of the cluster. The center of mass is defined by

$$\mathbf{r}_{\text{cm}} = \frac{1}{N} \sum_i \mathbf{r}_i, \quad (13.4)$$

where N is the total number of particles in the cluster. A typical plot of $\ln M$ versus $\ln r$ is shown in Figure 13.4. Because the cluster cannot grow past the edge of the lattice, we do not include data for $r \approx L$.

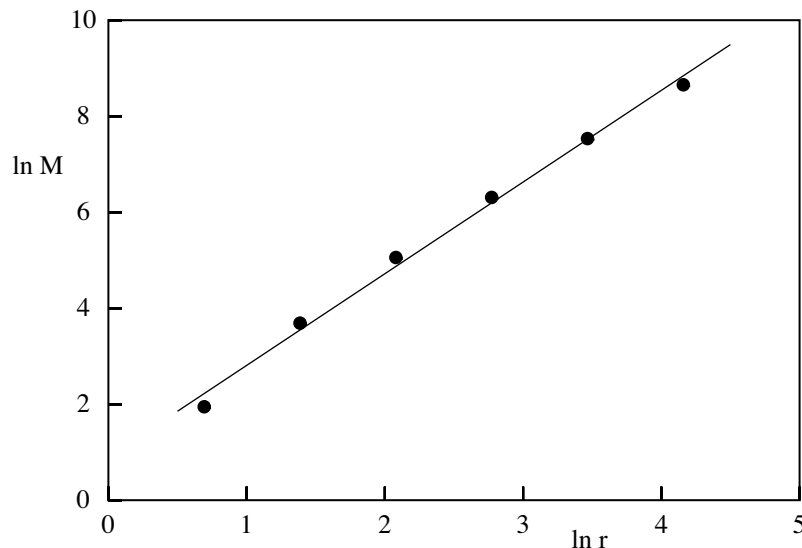


Figure 13.4: Plot of $\ln M$ versus $\ln r$ for a single spanning percolation cluster generated at $p = 0.5927$ on a $L = 129$ square lattice. The straight line is a linear least squares fit to the data. The slope of this line is 1.91 and is an estimate of the fractal dimension D . The exact value of D for a percolation cluster is $D = 91/48 \approx 1.896$.

Problem 13.3. Single cluster growth and the fractal dimension

- Explain how the Leath algorithm generates single clusters in a way that is equivalent to the multiple clusters that are generated by visiting all sites. More precisely, the Leath algorithm generates percolation clusters with a distribution of cluster sizes, sn_s . The additional factor of s is due to the fact that each site of the cluster has an equal chance of being the seed of the cluster, and hence the same cluster can be generated in s ways. See Project 13.18 for a discussion of the scaling form of n_s .
- Use class `Cluster` to grow percolation clusters using the Leath algorithm, and write a target class that shows the cluster as it grows and plots the mass distribution when the animation is stopped. Consider a spanning cluster to be one that connects the top and bottom rows of the lattice. Can you grow a spanning cluster for $p = 0.4$ or does the growth usually stop after a few sites are occupied? Choose $L \geq 31$.
- Choose $p = 0.5927$ and $L \geq 31$ and generate several pictures of spanning clusters. Do all your trials generate a spanning cluster? Explain. Determine the number of occupied sites $M(r)$ within a distance r of the center of mass of the cluster. Determine M for several values of r .

and average $M(r)$ over at least ten spanning clusters. Estimate D from the log-log plot of M versus r (see Figure 13.4). If time permits, generate percolation clusters on larger lattices.

- d. Grow as large a spanning cluster as you can and look at it on different length scales. One way to do so is to divide the screen into four windows, each of which magnifies a part of the cluster shown in the previous window. Does the part of the cluster shown in each window look approximately self-similar?
- e. Generate clusters at $p = 0.65$, a value of p greater than p_c , for $L = 61$. Make a log-log plot of $M(r)$ versus r . Is the slope approximately equal to the value of D found in part (b)? Does the slope increase or decrease for larger r ? Repeat for $p = 0.80$. Is a spanning cluster generated at $p > p_c$ a fractal?
- f. The fractal dimension of percolation clusters is not an independent exponent, but satisfies the scaling law

$$D = d - \beta/\nu, \quad (13.5)$$

where β and ν are defined in Table 12.1. The relation (13.5) can be understood by a finite-size scaling argument which we now summarize. The number of sites in the spanning cluster on a lattice of linear dimension L is given by

$$M(L) \sim P_\infty(L)L^d, \quad (13.6)$$

where P_∞ is the probability that an occupied site belongs to the spanning cluster and L^d is the total number of sites in the lattice. In the limit of an infinite lattice and p near p_c , we know that $P_\infty(p) \sim (p - p_c)^\beta$ and $\xi(p) \sim (p - p_c)^{-\nu}$ independent of L . Hence for $L \sim \xi$, we have that $P_\infty(L) \sim L^{-\beta/\nu}$ (see (12.13)), and we can write

$$M(L) \sim L^{-\beta/\nu} L^d \sim L^D. \quad (13.7)$$

The relation (13.5) follows. Use the exact values of β and ν from Table 12.1 to find the exact value of D for $d = 2$. Is your estimate for D consistent with this value?

- g.* Estimate the fractal dimension for percolation clusters on a simple cubic lattice. Take $p_c = 0.3117$.

13.2 Regular Fractals

As we have seen, one characteristic of random fractal objects is that they look the same on a range of length scales. To gain a better understanding of the meaning of self-similarity, consider the following example of a *regular* fractal, a mathematical object that is self-similar on *all* length scales. Begin with a line one unit long (see Figure 13.5a). Remove the middle third of the line and replace it by two lines of length 1/3 each so that the curve has a triangular bump in it and the total length of the curve is 4/3 (see Figure 13.5b). In the next stage, each of the segments of length 1/3 is divided into lines of length 1/9 and the procedure is repeated as shown in Figure 13.5c. What is the length of the curve shown in Figure 13.5c?

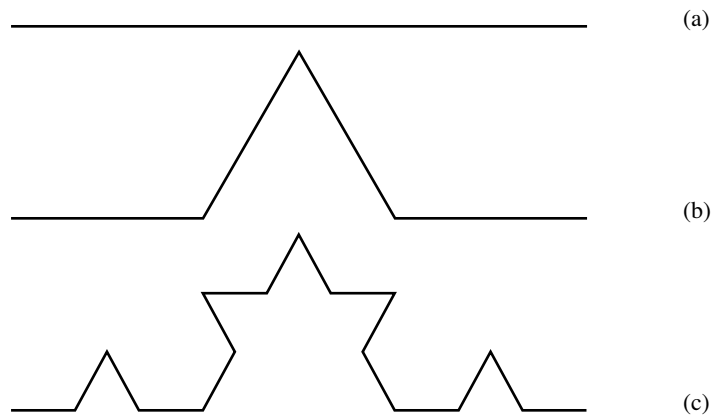


Figure 13.5: The first three stages (a)–(c) of the generation of a self-similar Koch curve. At each stage the displacement of the middle third of each segment is in the direction that increases the area under the curve. The curves were generated using `Class Koch`. The Koch curve is an example of a continuous curve for which there is no tangent defined at any of its points. The Koch curve is self-similar on each length scale.

The three stages shown in Figure 13.5 can be extended an infinite number of times. The resulting curve is infinitely long and contains an infinite number of infinitesimally small segments. Such a curve is known as the triadic Koch curve. A Java class that uses a recursive procedure (see Section 6.3) to draw this curve is given in the following. Note that `method iterate` calls itself. Use `Class KochApp` to generate the curves shown in Figure 13.5.

Listing 13.2: Class for drawing the Koch curve.

```
/* Draws Koch curve */

package org.opensourcephysics.sip.ch13;
import org.opensourcephysics.controls.*;
import org.opensourcephysics.display.*;
import org.opensourcephysics.display2d.*;
import javax.swing.*;
import java.awt.geom.*;
import java.awt.*;
import java.util.*;

public class KochApp extends AbstractCalculation implements Drawable {
    DrawingPanel drawingPanel = new DrawingPanel();
    DrawingFrame frame = new DrawingFrame(drawingPanel);
    int n = 0;

    public KochApp() {
        frame.show();
        drawingPanel.addDrawable(this);
    }
}
```

```

        drawingPanel.setPreferredMinMax(-100, 600, -100, 600);
        drawingPanel.setSquareAspect(true);
        drawingPanel.repaint();
    }

    public void calculate() {
        n = control.getInt("Number of iterations");
        drawingPanel.repaint();
    }

    public void iterate(double x1, double y1, double x2, double y2, int n,
                     DrawingPanel myWorld, Graphics g) {
        if (n > 0) {
            double dx = (x2-x1)/3;
            double dy = (y2-y1)/3;
            double xOneThird = x1 + dx; // new endpoint at 1/3 of line segment
            double yOneThird = y1 + dy;
            double xTwoThird = x1 + 2*dx; // new endpoint at 2/3 of line segment
            double yTwoThird = y1 + 2*dy;
            // rotate line segment (dx, dy) by 60 degrees and add to (xOneThird, yOneThird)
            double xMidPoint = (0.5*dx - 0.866*dy + xOneThird);
            double yMidPoint = (0.5*dy + 0.866*dx + yOneThird);
            // each line segment generates 4 new ones
            iterate(x1, y1, xOneThird, yOneThird, n-1, myWorld, g);
            iterate(xOneThird, yOneThird, xMidPoint, yMidPoint, n-1, myWorld, g);
            iterate(xMidPoint, yMidPoint, xTwoThird, yTwoThird, n-1, myWorld, g);
            iterate(xTwoThird, yTwoThird, x2, y2, n-1, myWorld, g);
        }
        else {
            int ix1 = myWorld.xToPix(x1);
            int iy1 = myWorld.yToPix(y1);
            int ix2 = myWorld.xToPix(x2);
            int iy2 = myWorld.yToPix(y2);
            g.drawLine(ix1, iy1, ix2, iy2);
        }
    }

    public void draw (DrawingPanel myWorld, Graphics g) {
        iterate (0,0,500,0, n, myWorld, g);
    }

    public void resetCalculation(){
        super.resetCalculation();
        control.setValue("Number of iterations" , 3);
    }

    public static void main(String args[]) {
        KochApp app = new KochApp();
        Control control = new CalculationControl(app);
        app.setControl(control);
    }

```

}
}

How can we determine the fractal dimension of the Koch and similar mathematical objects? In Section 13.5 we will see that there are several generalizations of the Euclidean dimension that lead naturally to a definition of the fractal dimension. Here we consider a definition based on counting boxes. Consider a one-dimensional curve of unit length that has been divided into N equal segments of length ℓ so that $N = 1/\ell$ (see Figure 13.6). As ℓ is decreased, N increases linearly — the expected result for a one-dimensional curve. Similarly if we divide a two-dimensional square of unit area into N equal subsquares of length ℓ , we have $N = 1/\ell^2$, the expected result for a two-dimensional object (see Figure 13.6). In general, we can write that $N = 1/\ell^D$, where D is the fractal dimension of the object. If we take the logarithm of both sides of this relation, we can express the fractal dimension as

$$D = \frac{\log N}{\log(1/\ell)}. \quad (\text{box dimension}) \quad (13.8)$$

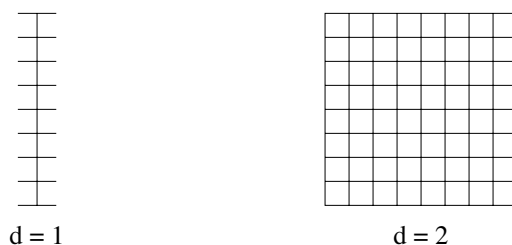


Figure 13.6: Examples of one-dimensional and two-dimensional objects.

Now let us apply these ideas to the Koch curve. We found that each time the length ℓ of our measuring unit is reduced by a factor of 3, the number of segments is increased by a factor of 4. Hence, we have $N = 4$ and $\ell = 1/3$, and the fractal dimension of the triadic Koch curve is given by

$$D = \frac{\log 4}{\log 3} \approx 1.2619. \quad (\text{triadic Koch curve}) \quad (13.9)$$

From (13.9) we see that the Koch curve has a dimension between that of a line and a plane. Is this statement consistent with your visual interpretation of the degree to which the triadic Koch curve fills space?

Problem 13.4. The recursive generation of regular fractals

- The concept of recursive programming as illustrated in `KochApp` is probably one of the most difficult programming concepts you will encounter. Explain the nature of recursion and the way it is implemented in `KochApp`.
- Regular fractals can be generated from a pattern that is used in a self-replicating manner. Write a program to generate the quadric Koch curve shown in Figure 13.7a. What is its fractal dimension?

- c. What is the fractal dimension of the Sierpiński gasket shown in Figure 13.7b? Write a program that generates the next several iterations.
- d. What is the fractal dimension of the Sierpiński carpet shown in Figure 13.7c? How does the fractal dimension of the Sierpiński carpet compare to the fractal dimension of a percolation cluster? Are the two fractals visually similar?

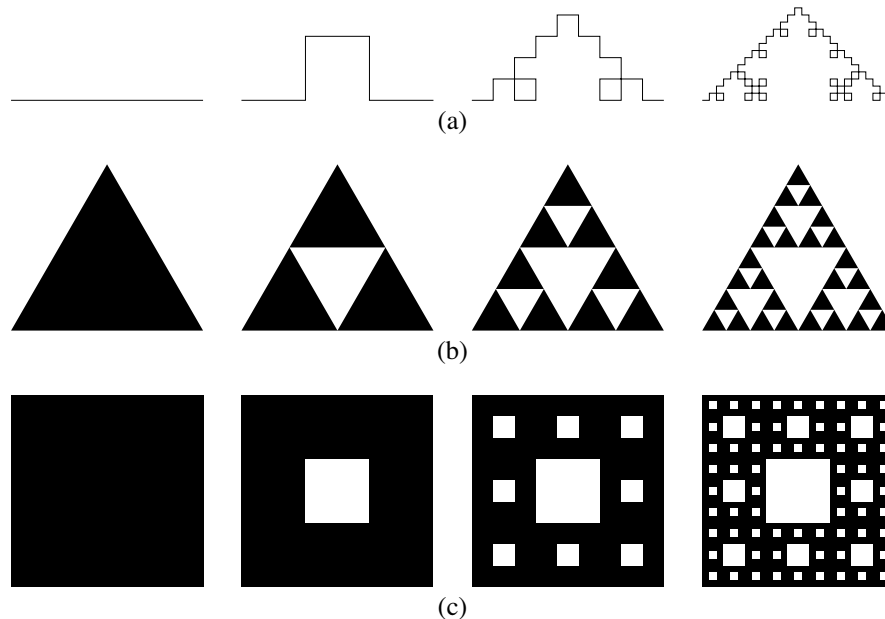


Figure 13.7: (a) The first few iterations of the quadric Koch curve; (b) The first few iterations of the Sierpiński gasket; (c) The first few iterations of the Sierpiński carpet.

13.3 Fractal Growth Processes

Many systems occurring in nature exhibit fractal geometry. Fractals have been used to describe the irregular shapes of such varied objects as coastlines, clouds, coral reefs, and the human lung. Why are fractal structures so common? How do fractal structures form? In this section we discuss several simple growth models that generate structures which show a remarkable similarity to forms observed in nature. The first two models are already familiar to us and exemplify the flexibility and general utility of kinetic growth models.

zzzz **Epidemic model.** In the context of the spread of disease, we usually want to know the conditions for an epidemic. A simple lattice model of the spread of a disease can be formulated as follows. Suppose that an occupied site corresponds to an infected person. Initially there is a single infected site and the four nearest neighbor perimeter sites (on the square lattice) are

susceptible. At the next time step, we visit the four susceptible sites and occupy (infect) each site with probability p . If a susceptible site is not occupied, we say that the site is immune and we do not test it again. We then find the new susceptible sites and continue until either the disease is controlled or reaches the boundary of the lattice. Convince yourself that this growth model of a disease generates a cluster of infected sites that is identical to a percolation cluster at probability p . The only difference is that we have introduced a discrete time step into the model. Some of the properties of this model are explored in Problem 13.5.

Problem 13.5. A simple epidemic model

- Explain why the simple epidemic model discussed in the text generates the same clusters as the Leath algorithm if the probability that a susceptible site becomes infected is p . What is the minimum value of p necessary for an epidemic to occur? Recall that in one time step, all susceptible sites are visited *simultaneously* and infected with probability p . Determine how N , the number of infected sites, depends on the time t (the number of time steps) for various values of p . A straightforward way to proceed is to extend `Class Cluster` with a new `method step` so that all perimeter sites are visited and occupied with probability p before new perimeter sites are found. In Chapter 14 we will learn that this model is an example of a cellular automaton.
- The susceptible (or growth) sites S are the only sites from which the disease can spread. Verify that for p near p_c^+ , S increases as $f(p)N^{\delta_s}$ with $f(p) \propto (p - p_c)^y$. Estimate the numerical values of the exponents δ_s and y . How does S depend on the time? Does δ_s have a different value at p_c for clusters that grow indefinitely? Choose $L \geq 61$ and average over at least 10 realizations.
- A similar growth exponent can be defined for $p < p_c$. In this case the number of susceptible sites does not increase without bound, and S usually first increases and then goes to zero. Determine the maximum number $S_{\max}$ of susceptible sites and show that $S_{\max} \propto (p_c - p)^{-x}$ near p_c . Estimate the numerical value of the exponent x .

Eden model. An even simpler example of a growth model was proposed by Eden in 1958 to simulate the growth of cell colonies. Although we will find that the resultant mass distribution is not a fractal, the description of the Eden growth algorithm illustrates the general nature of the fractal growth models we discuss.

The algorithm can be summarized as follows. Place a seed site at the origin, for example, the center of the lattice. The unoccupied nearest neighbors of the occupied sites are denoted as *growth* or perimeter sites. In the simplest version of the model, a growth site is chosen at random and occupied. The newly occupied site is removed from the list of growth sites and the new growth sites are added to the list. This growth process is repeated many times until a large cluster of occupied sites are formed (see Figure 13.8). The basic difference between this model and the previous one is that all tested sites are occupied. In other words, no sites are ever “immune.” Some of the properties of Eden clusters are investigated in Problem 13.6.

Problem 13.6. Eden model

- Use `Class Cluster` so that clusters are generated on a square lattice according to the Eden model. A straightforward procedure is to occupy perimeter sites with probability $p = 1$. The simulation should be stopped when the cluster reaches the edge of the lattice. What would

								p							
							p	46	p	p		p	p		
					p	p	p	44	p	34	p	43	52	p	
				p	47	13	35	37	p	33	25	39	p		
			p	50	24	8	16	27	19	12	18	p			
			p	10	2	1	3	5	6	20	p				
				p	26	4	29	14	9	51	p				
				p	49	15	7	28	17	21	30	42	p		
				p	41	23	11	45	p	32	31	p			
				p	48	p	22	p	p	40	36	p			
					p	p	38	p		p	p				
							p								

Figure 13.8: An example of a cluster grown on the square lattice according to the Eden model. The numbers on the sites denote the order in which these sites were occupied and the growth sites are denoted by the letter p.

happen if we were to occupy perimeter sites indefinitely? Follow the procedure of Problem 13.3 and determine the number of occupied sites $M(r)$ within a distance r of the seed site. Assume that $M(r) \sim r^D$ for sufficiently large r , and estimate D from the slope of a log-log plot of M versus r . A typical log-log plot is shown in Figure 13.9 for $L = 61$. Can you conclude from your data that Eden clusters are compact?

- b. Modify your program so that only the perimeter or growth sites are shown. Where are the majority of the perimeter sites relative to the center of the cluster? Grow as big a cluster as your time and patience permits.

Invasion percolation. A dynamical process known as *invasion percolation* can be used to model the shape of the oil-water interface which occurs when water is forced into a porous medium containing oil. The idea is to use the water to recover as much oil as possible. In this process a water cluster grows into the oil through the path of least resistance. Consider a lattice of size $L \times 2L$, with the water (the invader) initially occupying the left edge (see Figure 13.10). The resistance to the invader is given by uniformly distributed random numbers between 0 and 1 which are assigned to each site in the lattice and held fixed throughout the invasion. Sites that are nearest neighbors of the invader sites are the perimeter sites. At each time step, the perimeter site with

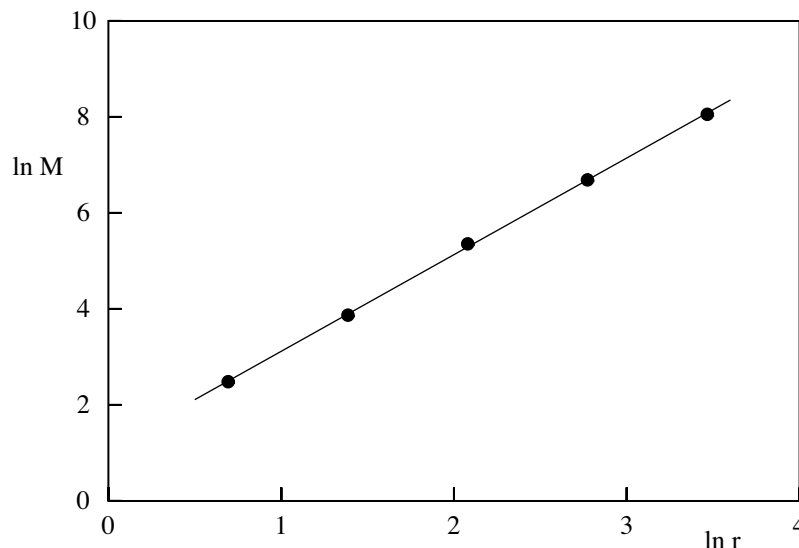


Figure 13.9: Plot of $\ln M$ versus $\ln r$ for a single Eden cluster generated on a $L = 61$ square lattice. A least squares fit to the data from $r = 2$ to $r = 32$ yields a slope of approximately 2.01.

the lowest random number is occupied by the invader and the oil (the defender) is displaced. The invading cluster grows until a path forms which connects the left and right edges of the lattice. Note that after this path forms, there is no need for the water to occupy any additional sites. To minimize boundary effects, periodic boundary conditions are used for the top and bottom edges and all quantities are measured only over the central $L \times L$ region of the lattice.

Class invasion implements the invasion percolation algorithm. The two-dimensional array element `site(i,j)` initially stores a random number for the site at (i,j) . If the site at (i,j) is occupied, then `site(i,j)` is increased by 1. If the site at (i,j) is a perimeter site, then `site(i,j)` is increased by 2, and is inserted into its proper ordered position in the perimeter lists `perx` and `pery`. The perimeter lists are ordered with the site with the largest random number at the beginning of the list.

Two searching routines are provided in **Class Invasion** for determining the position of a new perimeter site in the perimeter lists. In a *linear search* we go through the list in order until the random number associated with the new perimeter site is between two random numbers in the list. In a *binary search* we divide the list in two, and determine the half in which the new random number belongs. Then we divide this half into half again and so on until the correct position is found. A comparison of the linear and binary search methods is investigated in Problem 13.7d. The binary search is the default method used in **Class Invasion**.

The main quantities of interest are the fraction of sites occupied by the invader, and the probability $P(r)dr$ that a site with a random number between r and $r + dr$ is occupied. The properties of the invasion percolation model are explored in Problem 13.7.

Listing 13.3: Class for simulating invasion percolation.

```

package org.opensourcephysics.sip.ch13.invasion;
import org.opensourcephysics.display.*;
import org.opensourcephysics.frames.*;
import java.awt.*;

public class Invasion {

    public int Lx,Ly;
    public double s[][];
    public int perimeterListX[],perimeterListY[];
    public int numberOfPerimeterSites;
    public boolean ok = true;
    public LatticeFrame lattice;

    public Invasion(LatticeFrame latticeFrame) {
        lattice = latticeFrame;
        lattice .setIndexToColor(0,Color.blue);
        lattice .setIndexToColor(1,Color.black);
    }

    public void initialize () {
        Lx = 2*Ly;
        s = new double[Lx][Ly];
        perimeterListX = new int[Lx*Ly];
        perimeterListY = new int[Lx*Ly];
        for(int y = 0; y < Ly; y++){
            s [0][y] = 1;    // occupy first column
            lattice .setValue(0,y,1);
        }
        for(int y = 0; y < Ly; y++)
            for(int x = 1; x < Lx; x++) {
                s[x][y] = Math.random();
                lattice .setValue(x,y,0);
            }
        numberOfPerimeterSites = 0;
        for(int y = 0; y < Ly; y++) { //second column are perimeter sites
            s [1][y] += 2;    // perimeter sites have s > 2;
            numberOfPerimeterSites++;
            insert (1,y);    // insert site in perimeter list in order.
        }
        ok = true;
    }

    public void insert(int x, int y) {
        int insertionLocation = binarySearch(x,y);
        for(int i = numberOfPerimeterSites-1; i > insertionLocation; i--) {
            perimeterListX[i] = perimeterListX[i-1];
            perimeterListY[i] = perimeterListY[i-1];
        }
        perimeterListX[insertionLocation] = x;
        perimeterListY[insertionLocation] = y;
        numberOfPerimeterSites++;
    }
}

```

```

    }
    perimeterListX[insertionLocation] = x;
    perimeterListY[insertionLocation] = y;
}

public int binarySearch(int x, int y) {
    int firstLocation = 0;
    int lastLocation = numberOfPerimeterSites-2;
    if(lastLocation < 0) lastLocation = 0;
    int middleLocation = (firstLocation + lastLocation)/2;
    // determine which half of list new number is in
    while(lastLocation - firstLocation > 1) {
        int middleX = perimeterListX[middleLocation];
        int middleY = perimeterListY[middleLocation];
        if(s[x][y] > s[middleX][middleY])
            lastLocation = middleLocation;
        else
            firstLocation = middleLocation;
        middleLocation = (firstLocation + lastLocation)/2;
    }
    return lastLocation;
}

public int linear_search(int x, int y) {
    if(numberOfPerimeterSites == 1)
        return 0;
    else
        for(int i = 0; i < numberOfPerimeterSites-1; i++)
            if(s[x][y] > s[perimeterListX[i]][perimeterListY[i]])
                return i;
    return numberOfPerimeterSites-1;
}

public void step() {
    if(ok) {
        int nx[] = {1,-1,0,0};
        int ny[] = {0,0,1,-1};
        int x = perimeterListX[numberOfPerimeterSites-1];
        int y = perimeterListY[numberOfPerimeterSites-1];
        if(x > Lx -3) ok = false;
        numberOfPerimeterSites--;
        s[x][y] -= 1;
        lattice.setValue(x,y,1);
        for(int i = 0; i < 4;i++) { // find new perimeter sites
            int perimeterX = x + nx[i];
            int perimeterY = (y + ny[i]) % Ly;
            if(perimeterY == -1) perimeterY = Ly-1;
            if(s[perimeterX][perimeterY] < 1) { // new perimeter site
                s[perimeterX][perimeterY] += 2;
                numberOfPerimeterSites++;
            }
        }
    }
}

```

```

        insert(perimeterX,perimeterY);
    }
}
}

public void computeDistribution(PlotFrame data) {
    int numberOfBins = 20;
    int numberOccupied = 0;
    double occupied[] = new double[numberOfBins];
    double number[] = new double[numberOfBins];
    double binSize = 1.0/numberOfBins;
    int minX = Lx/3;
    int maxX = 2*minX;
    for(int x = minX; x <= maxX; x++)
        for(int y = 0; y < Ly; y++) {
            int bin = (int)(numberOfBins*(s[x][y] % 1));
            number[bin]++;
            if((s[x][y] >= 1) && (s[x][y] < 2)){
                numberOccupied++;
                occupied[bin]++;
            }
        }
    data.setMessage("Number occupied = " + numberOccupied);
    for(int bin = 0; bin < numberOfBins; bin++)
        data.append(0,(bin+0.5)*binSize, occupied[bin]/number[bin]);
}
}

```

Problem 13.7. Invasion percolation

- a. Use `Class Invasion` to generate an invasion percolation cluster on a 20×40 lattice and describe the qualitative nature of the cluster.
- b. Calculate $M(L)$, the number of sites occupied by the invader in the central $L \times L$ region of the $L \times 2L$ lattice at the time that the invader first reaches the right edge, is averaged over at least twenty trials. Assume that $M(L) \sim L^D$ and estimate D from a plot of $\ln M$ versus $\ln L$. Compare your estimate for D with the fractal dimension of ordinary percolation. (The published results for $M(L)$ by Wilkinson and Willemsen are for 2000 realizations each for L in the range 20 to 100.)
- c. Determine the probability $P(r)dr$ that a site with a random number between r and $r + dr$ is occupied. It is sufficient to choose $dr = 0.05$. Plot $P(r)$ versus r for $L = 20$ and also for larger values of L up to $L = 50$. Can you define a critical value of r near which $P(r)$ changes rapidly? How does this critical value of r compare to the value of p_c for ordinary site percolation on the square lattice? On the basis of your numerical estimate for the exponent D found in part (b) and the qualitative behavior of $P(r)$, make an hypothesis about the relation between the nature of the geometrical properties of the invasion percolation cluster and the spanning percolation cluster at $p = p_c$.

- d. Explain the nature of the two searching algorithms given in **Class Invasion**. Which method yields the fastest results on a 30×60 lattice? Verify that the CPU time for a linear and binary search is proportional to n and $\log n$ respectively, where n is the number of items in the list to be searched. Hence, for sufficiently large n , a binary search usually is preferred.
- e.* Modify your program so that the invasion percolation clusters are grown from a seed at the origin. Grow a cluster until it either occupies a given fraction of the lattice or it reaches a boundary of the lattice. Estimate the fractal dimension as you did for the spanning percolation clusters in Problem 13.3 and compare your two estimates. On the basis of this estimate and your results from part (b) and (c), can you conclude that the spanning cluster is a fractal? Note that this process of occupying the minimum number of sites to obtain a spanning cluster is an example of a self-organized critical phenomenon (see Chapter 14).

Diffusion in disordered media. In Chapter 7 we considered random walks on perfect lattices and on simple continuum systems. We found that the mean-square displacement of a random walker, $\langle R^2(t) \rangle$, is proportional to the time t for sufficiently large t . (For a simple random walk this relation holds for all t .) Now let us suppose that the random walker is restricted to a disordered lattice, for example, the occupied sites of a percolation cluster. What is the asymptotic t -dependence of $\langle R^2(t) \rangle$ in this case? This simple model of a random walk on a percolation cluster is known as the “ant in the labyrinth” problem.

There are at least two reasons for our interest in random walks on disordered lattices. Just as a random walk on a lattice is a simple model of diffusion, a random walk on a disordered lattice is a simple example of the general problem of diffusion and transport in disordered media. Because most materials of interest are noncrystalline and disordered, there are many physical phenomena that can be related to the motion of an ant in the labyrinth. Another reason for the interest in diffusion in disordered media is that the diffusion coefficient is proportional to the electrical conductivity of the medium. This relation between the conductivity and the diffusion coefficient is known as the Einstein relation (cf. Reif). We can understand this relation as follows. Consider for example, a system of electrons. Classically, we can follow the individual motion of the electrons and determine their mean square displacement. In the absence of external forces we can measure the self-diffusion coefficient D . In the presence of a “small” electric field, we can measure the electron’s mean velocity in the direction of the field and deduce the electron’s *mobility* μ , the ratio of the mean velocity to the applied force. Einstein’s contribution was to show that μ is proportional to D , that is, the linear response of the system is related to an equilibrium quantity. Because the mean velocity of the electrons is proportional to the electron current and the applied force is proportional to the voltage, the mobility and the electrical conductivity are proportional. Hence, we conclude that the conductivity is proportional to the self-diffusion coefficient.

In the usual formulation of the ant in the labyrinth problem we place a walker (the ant) at random on one of the occupied sites of a percolation cluster generated with probability p . At each time step, the ant tosses a coin with four possible outcomes (on a square lattice). If the outcome corresponds to a step to an occupied site, the ant moves; otherwise it remains in its present position. Either way, the time t is increased by one unit. The main quantity of interest is $R^2(t)$, the square of the distance between the ant’s position at $t = 0$ and its position at time t . We can generate many walks with different initial positions on the same cluster as well as over many percolation clusters to obtain the ant’s mean square displacement $\langle R^2(t) \rangle$. How does $\langle R^2(t) \rangle$ depend on p and

t ? How do the laws of diffusion change on a fractal lattice (for example, the percolation cluster at $p = p_c$)? We consider these questions in Problem 13.8.

Problem 13.8. The ant in the labyrinth

- a. For $p = 1$, the ants walk on a perfect lattice, and hence, $\langle R^2(t) \rangle \propto t$. Suppose that an ant does a random walk on a two-dimensional percolation cluster with $p > p_c$. Assume that $\langle R^2(t) \rangle \sim 4D_s(p)t$ for $p > p_c$. We have denoted the diffusion coefficient by D_s to remind ourselves that we are considering random walks on spanning clusters only and are not considering walks on the finite clusters that also exist for $p > p_c$. Generate a percolation cluster at $p = 0.7$ using the growth algorithm considered in Problem 13.3. Choose the initial position of the ant to be the seed site and modify your program to observe the motion of the ant on the screen. Where does the ant spend much of its time? If the ant diffuses, what can you say qualitatively about the ratio $D_s(p)/D(p = 1)$?
- b. Compute $\langle R^2(t) \rangle$ for $p = 0.4$ and confirm that for $p < p_c$, the clusters are finite, $\langle R^2(t) \rangle$ is bounded, and diffusion is impossible.
- c. As in part (a) compute the mean square displacement for $p = 1.0, 0.8, 0.7, 0.65$, and 0.62 with $L = 61$. If time permits, average over several clusters. Make a log-log plot of $\langle R^2(t) \rangle$ versus t . What is the qualitative t -dependence of $\langle R^2(t) \rangle$ for relatively short times? Decide whether $\langle R^2(t) \rangle$ is proportional to t for longer times. (Remember that the maximum value of $\langle R^2 \rangle$ is bounded by the finite size of the lattice.) If $\langle R^2(t) \rangle \sim t$, estimate $D_s(p)$. Plot the ratio $D_s(p)/D(p = 1)$ as a function of p and discuss its qualitative behavior.
- d. Because there is no diffusion for $p < p_c$, we might expect that D_s vanishes as $p \rightarrow p_c$, that is, $D_s(p) \sim (p - p_c)^{\mu_s}$ for $p \geq p_c$. Extend your calculations of part (c) to larger L and more values of p near p_c and estimate the dynamical exponent μ_s .
- e. At $p = p_c$, we might expect a different type of t -dependence of $\langle R^2(t) \rangle$ to be observed, for example, $\langle R^2(t) \rangle \sim t^{2/z}$ for large t . Do you expect the exponent z to be greater or less than two? Do a Monte Carlo simulation of $\langle R^2(t) \rangle$ at $p = p_c$ and estimate z . Choose $L \geq 61$ and average over several spanning clusters.
- f. The chicken wire measurements by Watson and Leath (see Section 12.1) found that the dc electrical conductivity σ vanishes near the percolation threshold as $\sigma \sim (p - p_c)^\mu$, with $\mu \approx 1.38 \pm 0.12$. More precise estimates give $\mu \approx 1.30$. The difficulty of doing a direct Monte Carlo calculation of σ was considered in Project 12.17. From the Einstein relation we know that the electrical conductivity and the self-diffusion coefficient behave in the same way. However, we measured the self-diffusion coefficient D_s by always placing the ant on a spanning cluster rather than on *any* cluster. In contrast, the conductivity is measured for the entire system including all finite clusters. Hence, the self-diffusion coefficient D that enters into the Einstein relation should be determined by placing the ant at random anywhere on the lattice, including sites that belong to the spanning cluster and sites that belong to the many finite clusters. Because only those ants that start on the spanning cluster can contribute to D , D is related to D_s by $D = P_\infty D_s$, where P_∞ is the probability that the ant would land on a spanning cluster. Because P_∞ scales as $P_\infty \sim (p - p_c)^\beta$, we have that $(p - p_c)^\mu \sim (p - p_c)^\beta (p - p_c)^{\mu_s}$ or $\mu_s = \mu - \beta$. Use your result for μ_s found in part (d) and the exact result $\beta = 5/36$ (see Table 12.1) to estimate μ and compare your result to the critical exponent for the dc electrical conductivity given above.

- g.* We have found that $D_s(p) \sim (p - p_c)^{\mu_s}$ for $p > p_c$ and $\langle R^2(t) \rangle \sim t^{2/z}$ for $p = p_c$. We now give a simple scaling argument to find a relation between z and μ_s . For $p > p_c$, we know that $\langle R^2(t) \rangle \sim (p - p_c)^{\mu_s} t$ in the limit of $t \gg 1$ such that the root mean square displacement is much larger than the connectedness length ξ . We also know that $\langle R^2(t) \rangle \sim t^{2/z}$ for shorter time scales that satisfy the condition $\langle R^2(t) \rangle \ll \xi^2$. We expect that the crossover between the two dependencies on t occurs when $\langle R^2 \rangle \sim \xi^2$ or when $t \sim \xi^z$. Hence we have $\xi^2 \sim (p - p_c)^{\mu_s} \xi^z$, or because $\xi \sim (p - p_c)^{-\nu}$, we have $(p - p_c)^{\nu(z-2)} \sim (p - p_c)^{\mu_s}$. If we equate powers of $(p - p_c)$, we have

$$z = 2 + \frac{\mu_s}{\nu} = 2 + \frac{\mu - \beta}{\nu}. \quad (13.10)$$

Is it easier to determine μ_s or z accurately from a Monte Carlo simulation on a finite lattice? That is, if our real interest is estimating the best value of the critical exponent μ for the conductivity, should we determine the conductivity directly or should we measure the self-diffusion coefficient at $p = p_c$ or at $p > p_c$? What is your best estimate of the conductivity exponent μ ?

- h.* A better method for treating random walks on a random lattice is to use an exact enumeration approach. The essence of the exact enumeration method is that $W_{t+1}(i)$, the probability that the ant is at site i at time $t + 1$, is determined solely by the probabilities of the ant being at the neighbors of site i at time t . Store the positions of the occupied sites in an array and introduce two arrays corresponding to $W_{t+1}(i)$ and $W_t(i)$ for all sites i in the cluster. Use the probabilities $W_t(i)$ to obtain $W_{t+1}(i)$ (see Figure 13.11). Spatial averages such as the mean square displacement can be calculated from the probability distribution function at different times. Details of the method and the results are discussed in Majid et al. These workers considered walks of 5000 steps on clusters with $\sim 10^3$ sites and averaged their results over 1000 different clusters.

Diffusion-limited aggregation (DLA). Many objects in nature grow by the random addition of subunits. Examples include snow flakes, lightning, crack formation along a geological fault, and the growth of bacterial colonies. Although it might seem unlikely that such phenomena have much in common, the behavior observed in many models that have been developed in recent years gives us clues that these and many other natural phenomena can be understood in terms of a few unifying principles. One model that has provided much insight is known as *diffusion limited aggregation* or DLA. The model provides an example of how random motion can give rise to beautiful self-similar clusters.

The first step is to occupy a site with a seed particle. Next, a particle is released from the perimeter of a large circle whose center coincides with the seed. The particle undergoes a random walk, that is, diffuses, until it reaches a perimeter site of the seed and sticks. Then another random walker is released and allowed to walk until it reaches a perimeter site of one of the two particles in the cluster and sticks. The process is repeated many times (typically on the order of several thousand to several million) until a large cluster is formed. A typical DLA cluster is shown in Figure 13.12. Some of the properties of DLA clusters are explored in Problem 13.9.

Problem 13.9. Diffusion limited aggregation

- a. Write a program to generate diffusion limited aggregation clusters on a square lattice. Let each walker begin at a random site on a circle of radius $r = R_{\max} + 2$, where $R_{\max}$ is the maximum distance of any cluster particle from the origin. To save computer time, assume that a walker that reaches a distance $2R_{\max}$ from the seed site is removed and a new walker is placed at random on the circle of radius $r = R_{\max} + 2$. Choose a lattice of linear dimension $L \geq 31$. Color code the cluster sites according to their time of arrival, for example, choose the first 100 sites to be blue, the next 100 sites to be yellow, etc. Which parts of the cluster grow faster? If the clusters appear to be fractals, make a visual estimate of the fractal dimension. (Experts can make a visual estimate of D to within a few percent!)
- b. At $t = 0$ the four perimeter (growth) sites on the square lattice each have a probability $p_i = 1/4$ of growing, that is, of becoming part of the cluster. At $t = 1$, the cluster has mass two and six perimeter sites. Identify the perimeter sites and convince yourself that their growth probabilities are not uniform. Do a Monte Carlo simulation and verify that two perimeter sites have $p_i = 2/9$ and the other four have $p_i = 5/36$. We discuss a more direct way of determining the growth probabilities in Problem 13.10.
- c. It is likely that your program generates DLA clusters inefficiently, because most of the CPU time is spent while the random walker is wandering far from the perimeter sites of the cluster. There are several ways of overcoming this problem. One way is to let the walker take bigger steps the further the walker is from the cluster. For example, if the random walker is at a distance $R > R_{\max}$, a step of length greater than or equal to $R - R_{\max} - 1$ may be permitted if this distance is greater than one lattice unit. If the walker is very close to the cluster, the step length is one lattice unit. Another possibility is to start the walk over if the walker moves too far away. Other possible modifications are discussed by Meakin (see references). Modify your program (or see **Class DLA** listed below) and estimate the fractal dimension of diffusion limited clusters generated on a square lattice.
- d.* Modify your program so that DLA clusters are generated on a triangular lattice. Do the clusters have the same visual appearance as on the square lattice? Estimate the fractal dimension and compare your estimate to your result for the square lattice.
- e.* In Chapter 12 we found that the exponents describing the percolation transition are independent of the symmetry of the lattice, for example, the exponents for the square and triangular lattices are the same. We might expect that the fractal dimension of DLA clusters would also show such universal behavior. However, the presence of a lattice introduces a small anisotropy that becomes apparent only when very large clusters with on the order of 10^6 sites are grown. We again are reminded of the difficulty of extrapolating from finite L to infinite L . The best estimates of D for the square and triangular lattices are $D \approx 1.5$ and $D \approx 1.7$ respectively. We consider the growth of diffusion-limited aggregation clusters in a continuum in Project 13.16.

The following class provides a reasonably efficient simulation of DLA. Walkers begin just outside a circle of radius `startRadius` enclosing the existing cluster and centered at the seed site $(0, 0)$. If the walker moves away from the cluster, the step size for the random walker increases. If the walker wanders too far away (further than `maxRadius`), then the walk is started over.

Listing 13.4: Class for simulating diffusion limited aggregation.

```

package org.opensourcephysics.sip.ch13.dla;
import org.opensourcephysics.display.*;
import java.awt.*;

public class DLA implements Drawable {
    public int s [][]; // lattice on which cluster lives
    public int xOccupied[],yOccupied[]; // location of occupied sites
    public int L; // linear lattice dimension
    public int halfL; // L/2
    public int ringSize; // ring size in which walkers can move
    public int numberOfParticles; // number of particles in cluster
    public int startRadius; // radius of cluster where walkers are started
    public int maxRadius; // maximum radius walker can go to before a new walk is started

    public void initialize () {
        s = new int[L][L];
        halfL = L/2;
        ringSize = L/10;
        xOccupied = new int[L*halfL];
        yOccupied = new int[L*halfL];
        s[halfL][halfL] = 1;
        xOccupied[0] = halfL;
        yOccupied[0] = halfL;
        numberOfParticles = 1;
        startRadius = 3;
        maxRadius = startRadius + ringSize;
    }

    public void step() {
        int x = 0, y = 0;
        if(startRadius < halfL) {
            // find random initial position of new walker
            do{
                double theta = 2*Math.PI*Math.random();
                x = halfL + (int)(startRadius*Math.cos(theta));
                y = halfL + (int)(startRadius*Math.sin(theta));
            }
            while(walk(x,y)); // random walk, returns true if new walk is needed
        }
    }

    /** Walk until next to perimeter site
     * @param x,y initial walker location
     *
     */
    public boolean walk(int x, int y) {
        do {
            double rSquared = (x-halfL)*(x-halfL) + (y-halfL)*(y-halfL);
            int r = 1 + (int)Math.sqrt(rSquared);

```

```

    if(r > maxRadius) return true; // start walk over
    if((r < halfL) &&
        (s[x+1][y] + s[x-1][y] + s[x][y+1] + s[x][y-1] > 0)) {
        s[x][y] = 1;
        xOccupied[numberOfParticles] = x;
        yOccupied[numberOfParticles] = y;
        numberOfParticles++;
        if(r >= startRadius) startRadius = r + 2;
        maxRadius = startRadius + ringSize;
        return false; // walk is finished
    }
    else { // take a step
        double random = Math.random();
        if(random < 0.25) {
            x++;
        }
        else if(random < 0.5) {
            x--;
        }
        else if(random < 0.75){
            y++;
        }
        else{
            y--;
        }
    } // end else if
}
while(true); // end do loop
}

public void draw (DrawingPanel myWorld, Graphics g) {
    if(s == null) return;
    int sizeX = Math.abs(myWorld.xToPix(0.8) - myWorld.xToPix(0));
    int sizeY = Math.abs(myWorld.yToPix(0.8) - myWorld.yToPix(0));
    for(int i = 0; i < numberOfParticles; i++){
        int xpix = myWorld.xToPix(xOccupied[i]) - sizeX;
        int ypix = myWorld.yToPix(yOccupied[i]) - sizeY;
        g.fillRect (xpix + sizeX/2,ypix + sizeY/2,sizeX,sizeY);
    }
}
}

```

***Laplacian growth model.** As we discussed in Section 10.8, we can formulate the solution of Laplace's equation in terms of a random walk. We now do the converse and formulate the DLA algorithm in terms of a solution to Laplace's equation. Consider the probability $P(\mathbf{r})$ that a random walker reaches a site $\mathbf{r}$ between the external boundary and the growing cluster without

having visited the cluster or the external boundary. This probability satisfies the relation

$$p(\mathbf{r}) = \frac{1}{4} \sum_{\mathbf{a}} p(\mathbf{r} + \mathbf{a}), \quad (13.11)$$

where the sum in (13.11) is over the four nearest neighbor sites (on a square lattice). If we set $p = 1$ on the boundary and $p = 0$ on the cluster, then (13.11) also applies to sites that are neighbors of the external boundary and the cluster. A comparison of the form of (13.11) with the form of (10.12) shows that the former is a discrete version of Laplace's equation, $\nabla^2 p = 0$. Hence $p(\mathbf{r})$ has the same behavior as the electrical potential between two electrodes connected to the outer boundary and the cluster respectively, and the growth probability at a perimeter site of the cluster is proportional to the value of the potential at that site.

***Problem 13.10.** Laplacian growth models

- a. Solve the discrete Laplace equation (13.11) by hand for the growth probabilities of a DLA cluster of mass 1, 2, and 3. Set $p = 1$ on the boundary and $p = 0$ on the cluster.
- b. You are probably familiar with the complicated and random nature of electrical discharge patterns that occur in atmospheric lightning. Although this phenomenon, known as *dielectric breakdown*, is complicated, we will see that a simple model leads to discharge patterns that are similar to those observed experimentally. Because lightning occurs in an inhomogeneous medium with differences in the density, humidity and conductivity of air, we want to develop a model of an electrical discharge in an inhomogeneous insulator. We know that when an electrical discharge occurs, the electrical potential ϕ satisfies Laplace's equation $\nabla^2 \phi = 0$. One version of the model (see Family et al.) is specified by the following steps:
 - (a) Consider a large boundary circle of radius R and place a charge source at the origin. Choose the potential $\phi = 0$ at the origin (occupied site) and $\phi = 1$ for sites on the circumference of the circle. The radius R should be larger than the radius of the growing pattern.
 - (b) Use the relaxation method (see Chapter 10) to compute the values of the potential ϕ_i for (empty) sites within the circle.
 - (c) Assign a random number r to each empty site within the boundary circle. The random number r_i at site i represents a breakdown coefficient and the random inhomogeneous nature of the insulator.
 - (d) The perimeter sites are the nearest neighbor sites of the discharge pattern (occupied sites). Form the product $r_i \phi_i^a$ for each perimeter site i , where a is an adjustable parameter. (Because the potential for the discharge pattern is zero, ϕ_i for perimeter site i can be interpreted as the magnitude of the potential gradient at site i .)
 - (e) The perimeter site with the maximum value of the product $r \phi^a$ breaks down, that is, set ϕ for this site equal to zero. (We can say that the bond between the discharge pattern and the perimeter site breaks down.)
 - (f) Use the relaxation method to recalculate the values of the potential at the remaining unoccupied sites and repeat steps (4)–(6).

Choose $a = \frac{1}{4}$ and analyze the structure of the discharge pattern. Does the pattern appear qualitatively similar to lightning? Does the pattern appear to have a fractal geometry? Estimate the fractal dimension by counting $M(b)$, the average number of sites belonging to the discharge pattern that are within a $b \times b$ box. Consider other values of a , for example, $a = \frac{1}{6}$ and $a = \frac{1}{3}$, and show that the patterns have a fractal structure with a tunable fractal dimension. Published results (Family et al.) are for patterns generated with 800 occupied sites.

- c. The usual version of the dielectric breakdown model associates a growth probability $p_i = \phi_i^a / \sum_j \phi_j^a$ with each perimeter site i , where the sum is over all perimeter sites. One of the perimeter sites is occupied with probability p_i . That is, choose a perimeter site at random and generate a random number r between 0 and 1. If $r \leq p_i$, the perimeter site i is occupied. As before, the exponent a is a free parameter. Convince yourself that $a = 1$ corresponds to diffusion-limited aggregation. (The boundary condition used in the latter corresponds to a zero potential at the perimeter sites.) To what type of cluster does $a = 0$ correspond? Choose $a = 1/2, 1$, and 2 and explore the dependence of the visual appearance of the clusters on a . If time permits, estimate the fractal dimension of the clusters.
- d. Consider a deterministic growth model for which *all* perimeter sites are tested for occupancy at each growth step. We adopt the same geometry and boundary conditions as in part (b) and use the relaxation method to solve Laplace's equation for ϕ_i . Then we find the perimeter site with the largest value of ϕ and set this value equal to $\phi_{\max}$. Only those perimeter sites for which the ratio $\phi_i/\phi_{\max}$ is larger than a parameter p become part of the cluster and ϕ_i is set equal to unity for these sites. After each growth step, the new perimeter sites are determined and the relaxation method is used to recalculate the values of ϕ_i at each unoccupied site. Choose $p = 0.35$ and determine the nature of the regular fractal pattern. What is the fractal dimension? Consider other values of p and determine the corresponding fractal dimension. These patterns have been termed *Laplace fractal carpets* (see Family et al.).

***Cluster-cluster aggregation.** Fractal structures commonly occur in aggregates that have been formed by the clustering of particles that are diffusing in a fluid. For example, colloids consist of particles that stick together in a liquid solvent, and aerosols are the analog in a gas. In DLA, all the particles that stick to a cluster are the same size (the growth occurs by cluster-monomer contact), and the cluster that is formed is motionless. In the following, we consider a cluster-cluster aggregation (CCA) model in which the clusters diffuse as they aggregate.

In a typical simulation we begin with a dilute collection of N particles. Each of these particles is a cluster with unit mass. The particles do random walks until one of them becomes a nearest neighbor of another particle. At that point they stick together to form a cluster of two particles. This new cluster now moves as a single random walker with a reduced diffusion coefficient. As this process continues, the clusters become larger and fewer in number. For simplicity, we assume a square lattice with periodic boundary conditions. The CCA algorithm can be summarized as follows:

1. Place N particles at random positions on the lattice. Do not allow a site to be occupied by more than one particle. Identify the i th particle with the i th cluster.
2. Check if any two clusters have particles that are nearest neighbors. If so, join these two clusters to form a single cluster.

3. Choose a cluster at random. Decide whether to move the cluster as discussed below. If so, move it randomly in one of the four possible directions. In the following, we discuss the strategy for deciding when to move a cluster.
4. Repeat steps 2 and 3 until the desired time or until there is only a single cluster.

What rule should we use to decide whether to move a cluster? One possibility is to select a cluster at random and simply move it. This possibility corresponds to all clusters having the same diffusion coefficient, regardless of the mass s of the cluster. A more realistic rule is to assume that the diffusion coefficient D_s is inversely related to the mass, for example as s^{-x} with $x \neq 0$. A common assumption in the literature is to assume $x = 1$. If we assume instead that D_s is inversely proportional to the linear dimension of the cluster, an assumption consistent with the Stokes-Einstein relation, it is reasonable to take $x = 1/d$. However because the clusters are fractals, we really should take $x = 1/D$, where D is the fractal dimension of the cluster. In Problem 13.11 we explore some of the possible forms of D_s .

To implement the cluster-cluster aggregation algorithm, we need to store the position of each particle and the cluster to which each particle belongs. In `Class CCA` the position of a particle is given by its x and y coordinates and stored in the arrays `x` and `y` respectively. The array element `site[x][y]` equals zero if there is no particle at (x, y) ; otherwise the element equals the label of the cluster to which the particle at (x, y) belongs.

The labels of the clusters are found as follows. The array element `firstParticle(k)` gives the particle label of the first particle in the k th cluster. To determine all the particles in a given cluster, we use a data structure called a *linked list*. This list is implemented using an array such that the value of an element of the array is the index for the next element in the linked list. The linked list is an example of a *circular linked list*, because the value of the last element in the linked list is the index for the first element. The array `nextParticle` contains a series of circular linked lists, one for each cluster, such that `nextParticle[i]` equals the particle label of another particle in the same cluster as the i th particle. If `nextParticle[i] = i`, then the i th particle is a cluster with only one particle. To see how these arrays work, consider three particles 5, 9, and 16 which constitute cluster 4. We have `firstParticle[4] = 5`, `nextParticle[5] = 9`, `nextParticle[9] = 16`, and `nextParticle[16] = 5`.

As the clusters undergo a random walk, we need to check if any pair of particles in different clusters have become nearest neighbors. If such a situation occurs, their respective clusters have to be merged. The check for nearest neighbors is done in `method checkNeighbors`. If for example, `site[x][y]` and `site[x+1][y]` are both nonzero and are not equal, then the two clusters associated with these sites need to be combined. To do so, we combine the two circular lists for each cluster into one circular list as is done in `method merge`. Hence if `p1` and `p2` are the first particles of their respective clusters, then

```
p1Next = nextParticle[p1]
p2Next = nextParticle[p2]
```

gives the second particle in each cluster. The following code merges the two clusters.

```
nextParticle[p1] = p2Next
nextParticle[p2] = p1Next
```

To complete the merger, all the entries in `site[x][y]` corresponding to the second cluster are relabeled with the label for the first cluster. To provide some efficiency we set the smaller cluster to be relabeled.

Listing 13.5: Class for simulating cluster–cluster aggregation.

```
package org.opensourcephysics.sip.ch13.cca;
import org.opensourcephysics.display.*;
import org.opensourcephysics.numerics.*;
import java.awt.*;

/**
 * CCA provides simualtes cluster–cluster aggregation
 * @author Jan Tobochnik
 */

public class CCA implements Drawable {
    public int[] site; // lattice on which clusters move
    public int[] x,y; // location of particles
    public int[] firstParticle ,nextParticle , lastParticle , mass;
    public int L; // linear lattice dimension
    public int numberOfParticles; // number of particles in system
    public int numberOfClusters; // number of clusters in system
    private int nnx[] = {1,0,-1,0}; // used to find neighbors of site
    private int nny[] = {0,1,0,-1};
    public int[] box;

    /**
     * initialize site lattice with single particle clusters
     * randomly placed on the lattice
     */
    public void initialize () {
        site = new int[L][L];
        for(int i = 0; i < L; i++) {
            for(int j = 0; j < L; j++) {
                site[i][j] = -1; // site not occupied
            }
        }
        x = new int[numberOfParticles];
        y = new int[numberOfParticles];
        firstParticle = new int[numberOfParticles];
        nextParticle = new int[numberOfParticles];
        lastParticle = new int[numberOfParticles];
        mass = new int[numberOfParticles+1];
        numberOfClusters = 0;
        for (int i = 0; i < numberOfParticles; i++) {
            do{
                x[i] = (int)(Math.random()*L);
                y[i] = (int)(Math.random()*L);
            }
            while(site[x[i]][y[i]] != -1);
        }
    }
}
```

```

        site [x[i]][y[i]] = numberOfClusters;
        firstParticle [numberOfClusters] = i;
        mass[numberOfClusters] = 1;
        nextParticle[i] = -1; // no more particles in cluster
        lastParticle [numberOfClusters] = i;
        numberOfClusters++;
        checkNeighbors(x[i],y[i]);
    }
}

/**
 * Checks to see if there are two neighbors not in same cluster
 * @param x_i,y_i
 */
public void checkNeighbors(int x_i, int y_i) {
    for(int j = 0; j < 4; j++) {
        int px = PBC.position(x_i + nnx[j],L);
        int py = PBC.position(y_i + nny[j],L);
        if((site [px][py] != -1) && (site[px][py] != site [x_i][y_i]))
            merge(site [px][py], site [x_i][y_i]);
    }
}

/**
 * Merges two clusters which are next to each other
 * @param largerCluster,smallerCluster
 */
public void merge(int c1, int c2){
    int largerClusterLabel,smallerClusterLabel;
    if(mass[c1] > mass[c2]) {
        largerClusterLabel = c1;
        smallerClusterLabel = c2;
    }
    else{
        largerClusterLabel = c2;
        smallerClusterLabel = c1;
    }
    // do the merging by first changing links in linked list
    nextParticle[ lastParticle [largerClusterLabel]] = firstParticle [smallerClusterLabel];
    lastParticle [largerClusterLabel] = lastParticle [smallerClusterLabel];
    mass[largerClusterLabel] += mass[smallerClusterLabel];
    int particle = firstParticle [smallerClusterLabel];
    do{
        site [x[particle]][y[particle]] = largerClusterLabel; // relabel sites of smaller cluster
        particle = nextParticle[particle];
    }
    while(particle != -1);
    numberOfClusters--;
    if(smallerClusterLabel != numberOfClusters) {

```

```

        particle = firstParticle [numberOfClusters];
    do{
        site [x[particle]][y[particle]] = smallerClusterLabel; // relabel sites of last cluster
        particle = nextParticle[particle];
    }
    while(particle != -1);
    firstParticle [smallerClusterLabel] = firstParticle [numberOfClusters];
    lastParticle [smallerClusterLabel] = lastParticle [numberOfClusters];
    mass[smallerClusterLabel] = mass[numberOfClusters];
}

}

/**
 * Move a cluster chosen at random in a random direction
 *
 */
public void step() {
    int cluster = (int)(Math.random()*numberOfClusters);
    int direction = (int)(Math.random()*4);
    int dx = nnx[direction];
    int dy = nny[direction];
    int particle = firstParticle [cluster];
    do {
        site [x[particle]][y[particle]] = -1;
        x[particle] = PBC.position(x[particle] + dx,L);
        y[particle] = PBC.position(y[particle] + dy,L);
        particle = nextParticle[particle];
    }
    while(particle != -1);
    particle = firstParticle [cluster];
    do {
        site [x[particle]][y[particle]] = cluster; // label new sites occupied by cluster
        particle = nextParticle[particle];
    }
    while(particle != -1);
    particle = firstParticle [cluster];
    do {
        checkNeighbors(x[particle],y[particle]); // check for merger
        particle = nextParticle[particle];
    }
    while(particle != -1);
}

public int occupiedSiteInCell(int cell, int i, int j) {
    for (int ic = 0; ic < cell; ic++) {
        for (int jc = 0; jc < cell; jc++) {
            if (site [i+ic][j+jc] > -1) return 1;
        }
    }
    return 0;
}

```

```

    }

    public void boxCount() {
        box = new int[L];
        int cell = 1;
        while (cell < L) {
            for (int i = 0; i < 1+L-cell; i += cell) {
                for (int j = 0; j < 1+L-cell; j += cell) {
                    box[cell] += occupiedSiteInCell(cell, i, j);
                }
            }
            cell *= 2;
        }
    }

    /** Draw clusters
     *
     */
    public void draw (DrawingPanel myWorld, Graphics g) {
        if (site == null) return;
        int sizeX = Math.abs(myWorld.xToPix(1.0) - myWorld.xToPix(0));
        int sizeY = Math.abs(myWorld.yToPix(1.0) - myWorld.yToPix(0));
        for (int i = 0; i < numberOfParticles; i++) {
            int xpix = myWorld.xToPix(x[i]) - sizeX;
            int ypix = myWorld.yToPix(y[i]) - sizeY;
            g.fillRect (xpix + sizeX/2, ypix + sizeY/2, sizeX, sizeY);
        }
    }
}

```

***Problem 13.11.** Cluster-cluster aggregation

- Class CCA assumes that the diffusion coefficient is independent of the cluster mass. Write a target class to use with Class CCA. Run your application with $L = 50$ and $N = 500$ and describe the qualitative appearance of the clusters as they form. Do they appear to be fractals? Compare their appearance to DLA clusters.
- Choose $L = 50$ and $N = 500$ and compute the fractal dimension of the final cluster. Use the center of mass, $\mathbf{r}_{\text{cm}}$, as the origin of the cluster, where $\mathbf{r}_{\text{cm}} = (1/N)(\sum_i x_i, \sum_i y_i)$ and (x_i, y_i) is the position of the i th particle. Average your results over at least ten final clusters. Do the same for other values of L and N . Are the clusters formed by cluster-cluster aggregation more or less space filling than DLA clusters?
- Assume that the diffusion coefficient of a cluster of s particles varies as $D_s \propto s^{-1/d}$, where d is the spatial dimension. Let D_{max} be the diffusion coefficient of the largest cluster. Choose a random number r between 0 and 1 and move the cluster if $r < D_s/D_{\text{max}}$. Repeat the above

simulations and discuss any changes in your results. What effect does this dependence of D on s have on the motion of the clusters? The time-dependence of the cluster size distribution is investigated in Project 13.19.

Surface growth. The fractal objects we have discussed so far are self-similar, that is, if we look at a small piece of the object and magnify it isotropically to the size of the original, then the original and the magnified object look similar (on the average). In the following, we introduce some simple models that generate a class of fractals that are self-similar only for scale changes in certain directions.

One of the problems in surface science is understanding the formation of rough surfaces. Suppose that we have a flat surface at time $t = 0$. Let us ask how the surface grows as a result of vapor deposition and sedimentation. For example, consider a surface which initially is a line of L occupied sites. Growth is confined to the vertical direction (see Figure 13.13).

As before, we simply choose a perimeter site at random and occupy it. The average height of the cluster is given by

$$\bar{h} = \frac{1}{N_s} \sum_{i=1}^{N_s} h_i, \quad (13.12)$$

where h_i is the distance of the i th surface site from the substrate, and the sum is over all surface sites N_s . (The precise definition of a surface site for the Eden model is discussed in Problem 13.12.)

Each time a particle is deposited, the time t is increased by unity. Our main interest is how the “width” of the surface changes with t . We define the width of the surface by

$$w^2 = \frac{1}{N_s} \sum_{i=1}^{N_s} (h_i - \bar{h})^2. \quad (13.13)$$

In general, the surface width w , which is a measure of the surface roughness, depends on L and t . Initially w grows with time. We expect that

$$w(L, t) \sim t^\beta. \quad (13.14)$$

The exponent β describes the growth of the correlations with time along the vertical direction. Figure 13.13 illustrates the evolution of the surface generated according to the Eden model. After a characteristic time, the length over which the fluctuations are correlated becomes comparable to L , and the width reaches a steady state value that depends only on L . We write

$$w(L, t \gg 1) \sim L^\alpha, \quad (13.15)$$

where α is known as the roughness exponent.

From (13.15) we see that in the steady state, the width of the surface in the direction perpendicular to the substrate grows as L^α . This steady-state behavior of the width is characteristic of a *self-affine fractal*. Such a fractal is invariant (on the average) under anisotropic scale changes, that is, different scaling relations exist along different directions. For example, if we rescale the surface by a factor b in the horizontal direction, then the surface must be rescaled by a factor of

b^α in the direction perpendicular to the surface to preserve the similarity along the original and rescaled surfaces.

Note that on short length scales, that is, lengths shorter than the width of the interface, the surface is rough and its roughness can be characterized by the exponent α . (Imagine an ant walking on the surface.) However on length scales much larger than the width of the surface, the surface appears to be flat and, in our example, it is a one-dimensional object. The properties of the surface as given by several growth models are explored in Problem 13.12.

Problem 13.12. Growing surfaces

- a. *Eden model.* In the Eden model a perimeter site is chosen at random and occupied. In this model there can be “overhangs” as shown in Figure 13.13, and the height h_x corresponds to the maximum distance of any perimeter site in column x from the surface. Use periodic boundary conditions in the horizontal directions to determine the perimeter sites. Note that the growth rule is the same as the usual Eden model, but the growth is started from the top of a strip of length L . Choose a square lattice with $L = 100$. Describe the visual appearance of the surface as the surface grows. Is the surface well-defined visually? Where are most of the perimeter sites? We have defined the surface sites as a subset of the perimeter sites (that is, those with maximum h for a given x). Do you think our results would be qualitatively different if we included all perimeter sites?
- b. Plot the width $w(t)$ as a function of t for $L = 32, 64$, and 128 on the same graph and estimate the exponents α and β for the Eden model. What type of plot is most appropriate? Does the width initially grow as a power law? If so, estimate the exponent β . Is there a L -dependent crossover time after which the width of the surface approaches its steady state value? How can you estimate the exponent α ? The best numerical estimates for β and α are consistent with the presumed exact values $\beta = 1/3$ and $\alpha = 1/2$, respectively.
- c.* The dependence of $w(L, t)$ on t and L can be combined into the scaling form

$$w(L, t) \approx L^\alpha f(t/L^{\alpha/\beta}) \quad (13.16)$$

where

$$f(x) \approx x^\beta \quad \text{for } x \ll 1 \quad (13.17a)$$

$$f(x) = \text{constant} \quad \text{for } x \gg 1 \quad (13.17b)$$

Verify the existence of the scaling form (13.16) by plotting the ratio $w(L, t)/L^\alpha$ versus $t/L^{\alpha/\beta}$ for the different values of L considered in part (b). If the scaling forms holds, the results for w for the different values of L should fall on a universal curve. Use either the estimated values of α and β that you found in part (b) or the exact results.

- d. *Random deposition.* The Eden model is not really a surface growth model, because any perimeter site can become part of the cluster. In the simplest deposition model, a column is chosen at random and a particle is deposited at the top of the column of already deposited particles. There is no horizontal correlation between neighboring columns. Do a simulation of this growth model and visually inspect the surface of the interface. Show that the heights of the columns follow a Poisson distribution (see (7.29)) and that $\bar{h} \sim t$ and $w \sim t^{1/2}$. This structure does not depend on L and hence $\alpha = 0$.

- e. *Ballistic deposition*. In this model a column is chosen at random and a particle is assumed to fall vertically until it reaches the first perimeter site that is a nearest neighbor of a site that already is part of the surface. This condition allows for growth parallel to the substrate. Only one particle falls at a time. How do the rules for this growth model differ from those of the Eden model? How does the deposit that you obtain compare to that of the Eden model? Suppose that instead of the particle falling vertically, we let it do a random walk as in DLA. Would the resultant surface be the same?

13.4 Fractals and Chaos

In Chapter 6 we explored dynamical systems that exhibited chaos under certain conditions. We found that after an initial transient, the trajectory of a dynamical system consists of a set of points in phase space called an attractor. For chaotic motion this attractor often is an object that can be described by a fractal dimension. Such attractors are called *strange attractors*.

We first consider the familiar logistic map (see (6.1)), $x_{n+1} = 4rx_n(1 - x_n)$. For most values of the control parameter $r > r_\infty = 0.892486417967\dots$, the trajectories are chaotic. Are these trajectories fractals? We explore this question in Problem 13.13.

To calculate the fractal dimension for dynamical systems, we use the *box counting* method in which space is divided into d -dimensional boxes of length ℓ . Let $N(\ell)$ equal the number of boxes that contain a piece of the trajectory. The fractal dimension is defined by the relation

$$N(\ell) \sim \lim_{\ell \rightarrow 0} \ell^{-D}. \quad (\text{box counting dimension}) \quad (13.18)$$

Equation (13.18) is accurate only when the number of boxes is much larger than $N(\ell)$ and the number of points on the trajectory is sufficiently large. If the trajectory moves through many dimensions, that is, the phase space is very large, box counting becomes too memory intensive because we need an array of size $\propto \ell^{-d}$. This array becomes very large for small ℓ and large d .

A more efficient approach is to calculate the *correlation dimension*. In this approach we store in an array the position of N points on the trajectory. We compute the number of points $N_i(r)$, and the fraction of points $f_i(r) = N_i(r)/(N-1)$ within a distance r of the point i . The correlation function $C(r)$ is defined by

$$C(r) \equiv \frac{1}{N} \sum_i f_i(r), \quad (13.19)$$

and the *correlation dimension* D_c is defined by

$$C(r) \sim \lim_{r \rightarrow 0} r^{D_c}. \quad (\text{correlation dimension}) \quad (13.20)$$

From (13.20) we see that the slope of a log-log plot of $C(r)$ versus r yields an estimate of the correlation dimension. In practice, small values of r must be discarded because we cannot sample all the points on the trajectory, and hence there is a cutoff value of r below which $C(r) = 0$. In the large r limit, $C(r)$ saturates to unity if the trajectory is localized as it is for chaotic trajectories. We expect that for intermediate values of r , there is a scaling regime where (13.20) holds.

In Problems 13.13–13.15 we consider the fractal properties of some of the dynamical systems that we considered in Chapter 6.

Problem 13.13. Fractal dimension of logistic map trajectories

- Write a program that uses box counting to determine the fractal dimension of the attractor for the logistic map. Compute $N(\ell)$, the number of boxes of length ℓ that have been visited by the trajectory. Test your program for $r < r_\infty$. How does the number of boxes containing a piece of the trajectory change with ℓ ? What does this dependence tell you about the dimension of the trajectory?
- Compute $N(\ell)$ for $r = 0.9$ using at least five different values of ℓ , for example, $1/\ell = 100, 300, 1000, 3000, \dots$. Iterate the map at least 1000 times before determining $N(\ell)$. What is the fractal dimension of the attractor? Repeat for $r \approx r_\infty$, $r = 0.95$, and $r = 1$.
- Generate points at random in the unit interval and estimate the fractal dimension using the same method as in part (b). What do you expect to find? Use your results to estimate the accuracy of the fractal dimension that you found in part (b).
- Write a program to compute the correlation dimension for the logistic map and repeat the calculations for parts (b) and (c).

Problem 13.14. Strange attractor of the Hénon map

- Use two-dimensional boxes of linear dimension ℓ to estimate the fractal dimension of the strange attractor of the Hénon map (see (6.32)) with $a = 1.4$ and $b = 0.3$. Iterate the map at least 100 times before computing $N(\ell)$. Does it matter what initial condition you choose?
- Compute the correlation dimension for the same parameters used in part (a) and compare D_c with the box dimension computed in part (a).
- Iterate the Hénon map and view the trajectory on the screen by plotting x_{n+1} versus x_n in one window and y_n versus x_n in another window. Do the two ways of viewing the trajectory look similar? Estimate the correlation dimension, where the i th data point is defined by (x_i, x_{i+1}) and the distance R_{ij} between the i th and j th data point is given by $R_{ij}^2 = (x_i - x_j)^2 + (x_{i+1} - x_{j+1})^2$.
- Estimate the correlation dimension with the i th data point defined by x_i , and $R_{ij}^2 = (x_i - x_j)^2$. What do you expect to obtain for D_c ? Repeat the calculation for the i th data point given by (x_i, x_{i+1}, x_{i+2}) and $R_{ij}^2 = (x_i - x_j)^2 + (x_{i+1} - x_{j+1})^2 + (x_{i+2} - x_{j+2})^2$. What do you find for D_c ?

***Problem 13.15.** Strange attractor of the Lorenz model

- Use three-dimensional graphics or three two-dimensional plots of $x(t)$ versus $y(t)$, $x(t)$ versus $z(t)$, and $y(t)$ versus $z(t)$ to view the structure of the Lorenz attractor. Use $\sigma = 10$, $b = 8/3$, $r = 28$, and the time step $\Delta t = 0.01$. Then compute the correlation dimension for the Lorenz attractor.
- Repeat the calculation of the correlation dimension using $x(t)$, $x(t + \tau)$, and $x(t + 2\tau)$ instead of $x(t)$, $y(t)$, and $z(t)$. Choose the delay time τ to be at least ten times greater than the time step Δt .

- c. Compute the correlation dimension in the two-dimensional space of $x(t)$ and $x(t + \tau)$. Do the same calculation in four dimensions using $x(t)$, $x(t + \tau)$, $x(t + 2\tau)$, and $x(t + 3\tau)$. What can you conclude about the results for the correlation dimension using two, three, and four-dimensional spaces. What do you expect to see for $d > 4$?

Problems 13.14 and 13.15 illustrate a practical method for determining the underlying structure of systems when, for example, the data consists only of a single time series, that is, measurements of a single quantity over time. The dimension $D_c(d)$ computed by increasing the dimension of the space, d , using the delayed coordinate τ eventually saturates when d is approximately equal to the number of variables that actually determine the dynamics. Hence, if we have extensive data for a single variable, for example, the atmospheric pressure, we can use this method to determine the number of independent variables that determine the dynamics of the pressure. This information can then be used to help create models of the atmosphere.

13.5 Many Dimensions

So far we have discussed three ways of defining the fractal dimension: the mass dimension (13.1), the box counting dimension (13.18), and the correlation dimension (13.20). These methods do not always give the same results for the fractal dimension. Indeed, there are many other dimensions that we could compute. For example, instead of just counting the boxes that contain a part of an object, we can count the number of points of the object in each box, n_i , and compute $p_i = n_i/N$, where N is the total number of points. A generalized dimension D_q can be defined as

$$D_q = \frac{1}{q-1} \lim_{\ell \rightarrow 0} \frac{\ln \sum_{i=1}^{N(\ell)} p_i^q}{\ln \ell}. \quad (13.21)$$

The sum in (13.21) is over all the boxes and involves the probabilities raised to the q th power. For $q = 0$, we have

$$D_0 = - \lim_{\ell \rightarrow 0} \frac{\ln N(\ell)}{\ln \ell}. \quad (13.22)$$

If we compare the form of (13.22) with (13.18), we can identify D_0 with the box-counting dimension. For $q = 1$, we need to take the limit of (13.21) as $q \rightarrow 1$. Let

$$u(q) = \ln \sum_i p_i^q, \quad (13.23)$$

and do a Taylor-series expansion of $u(q)$ about $q = 1$. We have

$$u(q) = u(1) + (q-1) \frac{du}{dq} + \dots \quad (13.24)$$

The quantity $u(1) = 0$ because $\sum_i p_i = 1$. The first derivative of $u(q)$ is given by

$$\frac{du}{dq} = \frac{\sum_i p_i^q \ln p_i}{\sum_i p_i^q} = \sum_i p_i \ln p_i, \quad (13.25)$$

where the last equality follows by setting $q = 1$. If we use the above relations, we find that D_1 is given by

$$D_1 = \lim_{\ell \rightarrow 0} \frac{\sum_i p_i \ln p_i}{\ln \ell}. \quad (\text{information dimension}) \quad (13.26)$$

D_1 is called the *information dimension* because of the similarity of the $p \ln p$ term in the numerator of (13.25) to the information form of the entropy.

It is possible to show that D_2 as defined by (13.21) is the same as the mass dimension defined in (13.1) and the correlation dimension D_c . That is, box counting gives D_0 and correlation functions give D_2 (cf. Sander et al. 1994).

There are many objects in nature that have similar fractal dimensions, but which nevertheless differ in appearance. An example of this difference is the visual appearance in three spatial dimensions of the clusters generated by diffusion-limited aggregation and the percolation clusters generated by the Leath algorithm at the percolation threshold. (Both objects have a fractal dimension of approximately 2.5.) In some cases this difference can be accounted for by *multifractal* properties of an object. For objects called *multifractals* the various D_q are different, in contrast to *monofractals* for which the different measures are the same. Percolation clusters are an example of a monofractal, because $p_i \sim \ell^{D_0}$, the number of boxes $N(\ell) \sim \ell^{-D_0}$, and from (13.21), $D_q = D_0$ for all q . Multifractals occur when the quantities p_i are not the same throughout the object, as frequently happens for the strange attractors produced by chaotic dynamics. DLA might be an example of a multifractal, and the appropriate probabilities p_i might correspond to the probability that the next perimeter site to be occupied is at i .

13.6 Projects

Although the kinetic growth models yield beautiful pictures and fractal objects, there is much we do not understand. Why do the fractal dimensions have the values that we found by various numerical experiments? Can we trust our numerical estimates of the various exponents or is it necessary to consider much larger systems to obtain their true asymptotic values? Can we find unifying features for the many kinetic growth models that presently exist? What is the relation of the various kinetic growth models to physical systems? What are the essential quantities needed to characterize the geometry of an object?

One of the reasons that growth models are difficult to understand is that typically the end product depends on the history of the growth. We say that these models are examples of “nonequilibrium behavior.” This combination of simplicity, beauty, complexity, and relevance to many experimental systems suggests that the study of fractal objects will continue to involve a wide range of workers in many disciplines.

Project 13.16. Off-lattice DLA

- a. In the continuum (off-lattice) version of diffusion-limited aggregation, the diffusing particles are assumed to be disks of radius a . A particle executes a random walk until its center is within a distance $2a$ of the center of a particle already attached to the DLA cluster. At each step the walker changes its position by $(r \cos \theta, r \sin \theta)$, where r is the step size, and θ is a random

variable between 0 and 2π . Modify your DLA program or `Class DLA` to simulate off-lattice DLA.

- b. Compare the appearance of an off-lattice DLA cluster with one generated on a square lattice. It is necessary to grow very large clusters (approximately 10^6 particles) to see any real differences.
- c. Use the mass dimension to estimate the fractal dimension of the off-lattice DLA cluster and compare its value with the value you found for the square lattice. Explain why the fractal dimensions might be different. $D \approx 1.71$ for off-lattice DLA in two dimensions while $D \approx 1.55$ for a square lattice (also see Problem 13.9). However, it is necessary to grow very large clusters to determine the effect of the lattice.

Project 13.17. More efficient simulation of DLA

In `Class DLA` we restart the walker if it wanders too far from the existing cluster to improve the efficiency of the algorithm. However, when the walker is within the distance `startRadius` of the seed, no optimization is used. Because there can be a great deal of empty space within this distance, we describe an additional optimization technique (see Ball and Brady). The basic idea is to use a simple geometrical object (a circle or square) centered at the walker such that none of the cluster is within the object. Then in one step move the walker to the perimeter of the object. For a circle the walker can move to any location with equal probability on the circle. For the square you need the probability of moving to various locations on the square. The major difficulty is to find the largest object that does not contain a part of the DLA cluster. To do this we consider coarse grained lattices. For example, each 2×2 group of sites on the original lattice corresponds to one site on the coarser lattice, and then each 2×2 group of sites on the coarse lattice corresponds to a site on an even coarser lattice, etc. If a site is occupied then any coarse site made from this site also is occupied.

- a. Because we have considered DLA clusters on a square lattice, we use squares centered at a walker. First we must find the probability $p(\Delta x, \Delta y, s)$ that a walker centered on a square of length $l = 2s + 1$, will be displaced by the vector $(\Delta x, \Delta y)$. This probability can be computed by simulating a random walk starting at the origin and ending at the edge of the square. These simulations are then repeated for many walkers, and then for each value of s . $p(\Delta x, \Delta y, s)$ is the fraction of walkers that reached the position $(\Delta x, \Delta y)$. Determine $p(\Delta x, \Delta y, s)$ for $s = 1$ to 16. Store your results in a file.
- b. We next need to produce an array such that for a given value of s and a random number r between 0 and 1, we can quickly find $(\Delta x, \Delta y)$. To do so create four arrays. The first array lists the probability distribution determined from p in part (a) such that the values for $s = 1$ are listed first, then the values for $s = 2$, etc. Call this array `p`. For example, $p(1) = p(-1, -1, 1)$, $p(2) = p(1) + p(-1, 0, 1)$, $p(3) = p(2) + p(-1, 1, 1)$, etc. Next create an array `start` that tells you where to start in the array `p` for each value of s . Then create the two arrays `dx(i)` and `dy(i)` which give the values of Δx and Δy corresponding to `p(i)`. To see how these arrays are used, consider a walker located at (x, y) , centered on a square of size $2s + 1$. First compute a random number r and find $i = \text{start}(s)$. If $p(i) > r$, then the walker moves to $(x + \text{dx}(i), y + \text{dy}(i))$. If not, increment i by unity and check again. Repeat until $p(i) > r$. Write a program to create these four arrays and store them in a file.

- c. Write a method to determine the maximum value of the parameter s such that a square of size $2s + 1$ centered at the position of the walker does not contain any part of the DLA cluster. Use coarse grained lattices to do this determination more efficiently.
- d. Modify **Class DLA** to incorporate the method from part (c) and read in the arrays from part (b). How much faster is your modified program than the original **Class DLA** for clusters of size 500 and 5000 particles? What is the largest cluster you can grow on your computer in one hour?
- e. Grow as large a cluster as you can. Is there any evidence for anisotropy? For example, does the cluster tend to extend further along the axes or along any other direction?

The following two projects introduce some of the important ideas associated with scaling.

Project 13.18. Scaling properties of the percolation cluster size distribution

- a. The scaling hypothesis for n_s , the number of percolation clusters of size s , near the percolation threshold p_c is

$$n_s = s^{-\tau} f_{\pm}(|p - p_c|^{1/\sigma} s), \quad (s \gg 1) \quad (13.27)$$

where the indices $+$ and $-$ refer to $p > p_c$ and $p < p_c$, respectively. The critical exponents τ and σ are the same above and below p_c . To understand the scaling form of n_s , first note that (13.27) implies that $n_s \sim s^{-\tau}$ at $p = p_c$ (compare with Table 12.1). We now show that the exponent σ can be related to ν and τ . Recall that the connectedness length ξ is given in terms of n_s by (see (12.11)):

$$\xi^2 = \frac{\sum_s s^2 n_s R_s^2}{\sum_s s^2 n_s}, \quad (13.28)$$

where R_s is the radius of gyration of the clusters. Close to p_c , the large clusters dominate the sum in (13.28). On length scales less than ξ , the clusters have no way of telling that the system is not at p_c and hence the large clusters are fractals with $R_s \sim s^{1/D}$. If we substitute this dependence and the scaling form (13.27) for n_s in (13.28), we obtain

$$\xi^2 \sim \frac{\sum_s s^{2-\tau+2/D} f_{\pm}(|p - p_c|^{1/\sigma} s)}{\sum_s s^{2-\tau} f_{\pm}(|p - p_c|^{1/\sigma} s)}. \quad (13.29)$$

For an infinite system the sums in (13.29) range from $s = 1$ to $s = \infty$. To calculate these sums, we transform them into integrals. For example, we can write the numerator of (13.29) as

$$\begin{aligned} \sum_{s=1}^{\infty} s^{2-\tau+2/D} f_{\pm}(|p - p_c|^{1/\sigma} s) &\sim \int_1^{\infty} s^{2-\tau+2/D} f_{\pm}(|p - p_c|^{1/\sigma} s) ds \\ &\propto (p - p_c)^{(\tau D - 3D - 2)/(D\sigma)} \int_{(p - p_c)^{1/\sigma}}^{\infty} x^{2-\tau+2/D} f_{\pm}(x) dx, \end{aligned} \quad (13.30)$$

where $x = (p - p_c)^{1/\sigma} s$. The denominator can be written in a similar form. If we assume that the integrands are nonsingular, then to lowest order in $(p - p_c)$, the lower integration limit can be set equal to zero. Show that we obtain

$$\xi^2 \sim |p - p_c|^{-2/(D\sigma)}. \quad (13.31)$$

Because $\xi \sim |p - p_c|^{-\nu}$, we obtain the desired relation between ν , σ , and D :

$$\nu = 1/(D\sigma). \quad (13.32)$$

If we write the argument $x = |p - p_c|^{1/\sigma} s$ as $x = s/\xi^D$, we can express the scaling form of n_s (13.27) in a more physical form:

$$n_s \sim s^{-\tau} f_{\pm}(s/\xi^D). \quad (13.33)$$

Equation (13.33) implies that n_s depends on s only through the ratio s/ξ^D or R_s/ξ . That is, the connectedness length represents the only important length near p_c . Show that the integrands in (13.30) are nonsingular if $2 - \tau > -1$ and fill in the missing algebra leading to (13.33).

- b. Let us try to verify the scaling form of n_s by generating percolation clusters of all sizes as we did in Chapter 12. We use the Leath algorithm to generate clusters of size s that are grown from a seed. Modify **Class Cluster** so that many clusters are generated and n_s is computed for a given input probability p . Remember that the number of clusters of size s that are grown from a seed is the product sn_s , rather than n_s itself (see Problem 13.3a). Run your program at $p = p_c \approx 0.592746$ and grow at least 100 clusters for a square lattice with $L \geq 61$. From (13.27) we see that at $p = p_c$, $n_s \sim s^{-\tau} f(0) \sim s^{-\tau}$. Hence a log-log plot of n_s versus s should give a straight line for $s \gg 1$ with a slope of $-\tau$. Estimate τ from your data. If time permits, use a bigger lattice and average over more clusters, and also estimate the accuracy of your estimate of τ .
- c. Determine n_s for at least three different values of p close to p_c , for example, 0.57, 0.58, and 0.61. Plot the product $s^\tau n_s$ versus the product $|p - p_c|^\sigma s$ using the value of τ found in part (b). Try various values of σ until your points for all values of p follow the same curve. Initially, try $\sigma = 0.4, 0.45$, and 0.5 .
- d. The mean cluster size $S(p)$ is related to n_s by (see (12.5)):

$$S(p) = \frac{\sum_s s^2 n_s}{\sum_s s n_s}. \quad (13.34)$$

Note that $S(p)$ can be regarded as the second moment of n_s . Use arguments similar to those used in part (a) to show that $S(p) \sim |p - p_c|^{(\tau-3)/\sigma}$ for p near p_c . Because $S(p) \sim |p - p_c|^{-\gamma}$, we have the relation

$$\gamma = (3 - \tau)/\sigma. \quad (13.35)$$

Use the values of τ and σ that you found in parts (b) and (c) to estimate γ .

Similar arguments can be made for P_∞ . Every site in the lattice is either empty with probability $1 - p$, or occupied and part of the spanning cluster with probability pP_∞ , or occupied but not part of the spanning cluster with probability $p(1 - P_\infty) = \sum_s s n_s$. Hence we have the exact relation

$$P_\infty(p) = 1 - \frac{1}{p} \sum_s s n_s. \quad (13.36)$$

Note that P_∞ can be regarded as the first moment of n_s . Hence using the same argument that led to (13.35), we find

$$\beta = (\tau - 2)/\sigma. \quad (13.37)$$

Compare this relation to the form of (13.35). A careful derivation of this scaling relation can be found on page 70 of Bunde and Havlin. Use (13.37) to estimate the critical exponent β .

Project 13.19. Dynamical scaling properties of the cluster size distribution in cluster-cluster aggregation

- a. The dynamical scaling assumption for the number of clusters of size s at time t for cluster-cluster aggregation is

$$n_s(t) \sim s^{-\theta} f(s/t^z), \quad (13.38)$$

where θ and z are exponents and $f(x)$ is a scaling function. The density of the particles is fixed and is related to n_s by the normalization condition:

$$\rho = \sum_s s n_s(t) \sim \int s n_s(t) ds. \quad (13.39)$$

Use the normalization condition (13.39) to show that $\theta = 2$. The scaling form (13.38) is expected to be applicable in the low density limit at large s and t .

- b. Show that the mean cluster size $S(t)$ given by

$$S(t) = \frac{\sum_s s^2 n_s(t)}{\sum_s s n_s(t)} \quad (13.40)$$

diverges for $t \rightarrow \infty$ as

$$S(t) \sim t^z \quad (13.41)$$

Hence the scaling form of $n_s(t)$ can be written as

$$n_s(t) \sim t^{-2} f(s/S(t)) \quad (13.42)$$

- c. Modify `Class CCA` so that the cluster size distribution, $n_s(t)$, is computed for $t = 2^p$ with $p = 1, 2, 3, \dots$ and $s = 1, 3, 10, 30$, and 100. For simplicity, assume that the diffusion coefficient of the clusters is size independent. Remember that $n_s = N_s(t)/L^2$, where $N_s(t)$ is the number of clusters of s particles at time t . The unit of time can be defined in various ways. The easiest way is to increase t by unity after a cluster has been moved one lattice unit. However, this choice would lead to the time changing faster when the number of clusters decreases. A better choice is to increase the time by an amount $\Delta t = s/N$, where s is the number of sites in the selected cluster and N is the (original) number of particles in the lattice. Choose $L = 50$ and $N = 500$ and average over at least ten runs. One way to test the dynamical scaling form of $n_s(t)$ is to plot $s^2 n_s(t)$ versus s/t^z for different choices of z . Another way is to make a log-log plot of $S(t)$ versus t and extract the exponent z from the slope of the linear portion of the log-log plot.

- d. Repeat part (c) for other values of L and N and discuss the accuracy of your results.
- e. A similar type of dynamics occurs in one dimension. What do you think the fractal dimension of the final cluster would be? In one dimension the surface of a cluster is two sites regardless of its size. As a result, the large clusters do not grow as fast as they do for higher dimensions. Can $n_s(t)$ still be described by a scaling form?

References and Suggestions for Further Reading

We have considered only a few of the models that lead to self-similar patterns. Use your imagination to design your own model of real-world growth processes. You are encouraged to read the research literature and recent books on growth models.

R. C. Ball and R. M. Brady, “Large scale lattice effect in diffusion-limited aggregation,” *J. Phys. A* **18**, L809 (1985). The authors discuss the optimization algorithm used in Project 13.17.

Albert-László Barabási and H. Eugene Stanley, *Fractal Concepts in Surface Growth*, Cambridge University Press (1995).

K. S. Birdi, *Fractals in Chemistry, Geochemistry, and Biophysics*, Plenum Press (1993).

Armin Bunde and Shlomo Havlin, editors, *Fractals and Disordered Systems*, Springer-Verlag (1991).

Fereydoon Family and David P. Landau, editors, *Kinetics of Aggregation and Gelation*, North-Holland (1984). A collection of research papers that give a wealth of information, pictures, and references on a variety of growth models.

Fereydoon Family, Daniel E. Platt, and Tamás Vicsek, “Deterministic growth model of pattern formation in dendritic solidification,” *J. Phys. A* **20**, L1177 (1987). The authors discuss the nature of Laplace fractal carpets.

Fereydoon Family and Tamás Vicsek, editors, *Dynamics of Fractal Surfaces*, World Scientific (1991). A collection of reprints.

Fereydoon Family, Y. C. Zhang, and Tamás Vicsek, “Invasion percolation in an external field: dielectric breakdown in random media,” *J. Phys. A* **19**, L733 (1986).

Jens Feder, *Fractals*, Plenum Press (1988). This text discusses the applications as well as the mathematics of fractals.

J.-M. Garcia-Ruiz, E. Louis, P. Meakin, and L. M. Sander, editors, *Growth Patterns in Physical Sciences and Biology*, NATO ASI Series B304, Plenum (1993).

J. M. Hammersley and D. C. Handscomb, *Monte Carlo Methods*, Methuen (1964). The chapter on percolation processes discusses a growth algorithm for percolation.

Shlomo Havlin and Daniel Ben-Avraham, “Diffusion in disordered media,” *Adv. Phys.* **36**, 695 (1987).

- H. J. Herrmann, “Geometrical Cluster Growth Models and Kinetic Gelation,” *Phys. Repts.* **136**, 154 (1986).
- Robert C. Hilborn, *Chaos and Nonlinear Dynamics*, Oxford University Press (1994).
- Benoit B. Mandelbrot, *The Fractal Geometry of Nature*, W. H. Freeman (1983). An influential and beautifully illustrated book on fractals.
- Imtiaz Majid, Daniel Ben-Avraham, Shlomo Havlin, and H. Eugene Stanley, “Exact-enumeration approach to random walks on percolation clusters in two dimensions,” *Phys. Rev. B* **30**, 1626 (1984).
- P. Meakin, “The growth of rough surfaces and interfaces,” *Physics Reports* **235**, 189 (1993). The author has written many seminal articles on DLA and other aggregation models.
- Paul Meakin, *Fractals, Scaling and Growth Far From Equilibrium*, Cambridge University Press (1995).
- L. Niemeyer, L. Pietronero, and H. J. Wiesmann, “Fractal dimension of dielectric breakdown,” *Phys. Rev. Lett.* **52**, 1033 (1984).
- H. O. Peitgen and P. H. Richter, *The Beauty of Fractals*, Springer-Verlag (1986).
- Luciano Pietronero and Erio Tosatti, editors, *Fractals in Physics*, North-Holland (1986). A collection of research papers, many of which are accessible to the motivated reader.
- Mark Przyborowski and Mark van Woerkom, “Diffusion of many interacting random walkers on a three-dimensional lattice with a personal computer,” *Eur. J. Phys.* **6**, 242 (1985). This work was done while the authors were high school students in West Germany.
- F. Reif, *Fundamentals of Statistical and Thermal Physics*, McGraw-Hill (1965). Einstein’s relation between the diffusion and mobility is discussed in Chapter 15.
- John C. Russ, *Fractal Surfaces*, Plenum Press (1994). A disk also is included.
- Evelyn Sander, Leonard M. Sander, and Robert M. Ziff, “Fractals and Fractal Correlations,” *Computers in Physics* **8**, 420 (1994). An introduction to fractal growth models and the calculation of their properties. One of the authors, Leonard Sander, is a co-developer of the diffusion limited aggregation model (see Problem 13.9).
- H. Eugene Stanley and Nicole Ostrowsky, editors, *On Growth and Form*, Martinus Nijhoff Publishers, Netherlands (1986). A collection of research papers at approximately the same level as the Family and Landau collection. The article by Paul Meakin on DLA was referenced in the text.
- Hideki Takayasu, *Fractals in the Physical Sciences*, John Wiley & Sons (1990).
- David D. Thornburg, *Discovering Logo*, Addison-Wesley (1983). The book is more accurately described by its subtitle, *An Invitation to the Art and Pattern of Nature*. The nature of recursive procedures and fractals are discussed using many simple examples.

Donald L. Turcotte, *Fractals and Chaos in Geology and Geophysics*, Cambridge University Press (1992).

Tamás Vicsek, *Fractal Growth Phenomena*, second edition, World Scientific Publishing (1991).
This book contains an accessible introduction to diffusion limited and cluster-cluster aggregation.

Bruce J. West, *Fractal Physiology and Chaos in Medicine*, World Scientific Publishing (1990).

David Wilkinson and Jorge F. Willemsen, “Invasion percolation: a new form of percolation theory,” *J. Phys. A***16**, 3365 (1983).

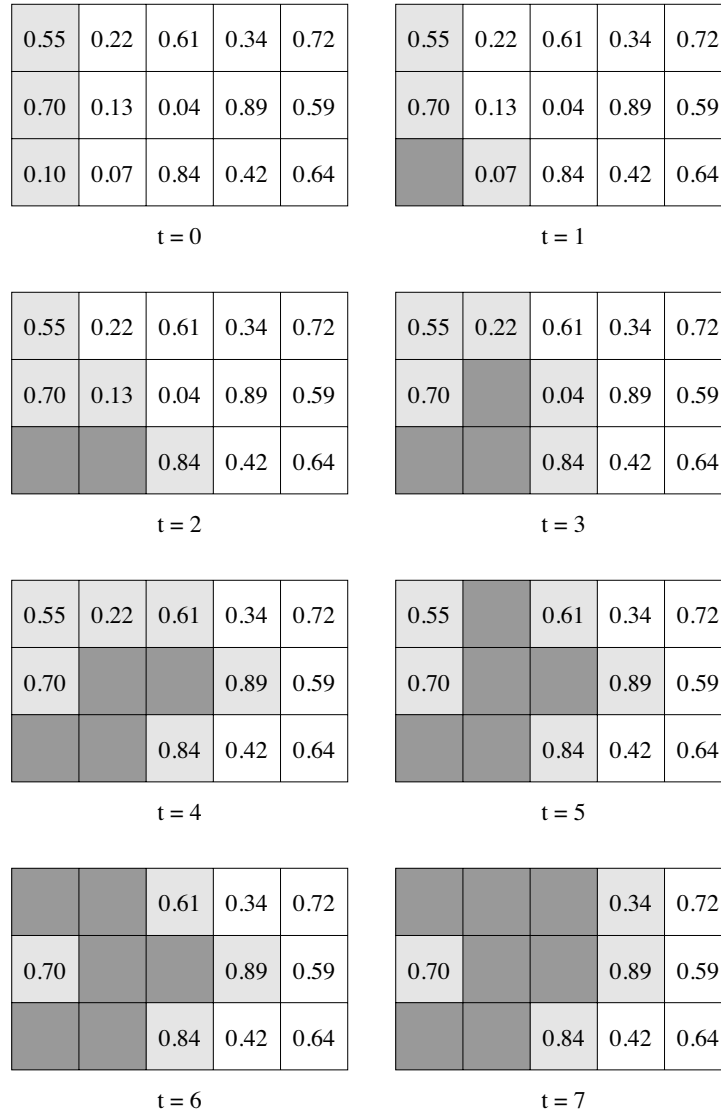


Figure 13.10: Example of a cluster formed by invasion percolation. The lattice at $t = 0$ shows the random numbers that have been assigned to the sites. The darkly shaded sites are occupied by the invader that occupies the perimeter site (lightly shaded) with the smallest random number.

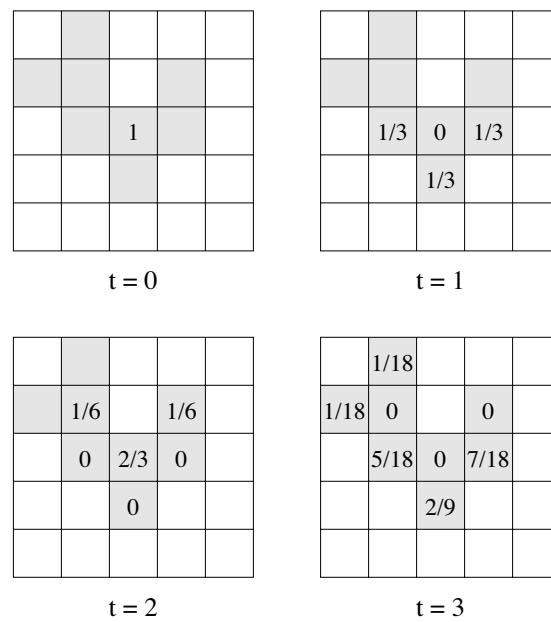


Figure 13.11: The evolution of the probability distribution function $W_t(i)$ for three successive time steps.

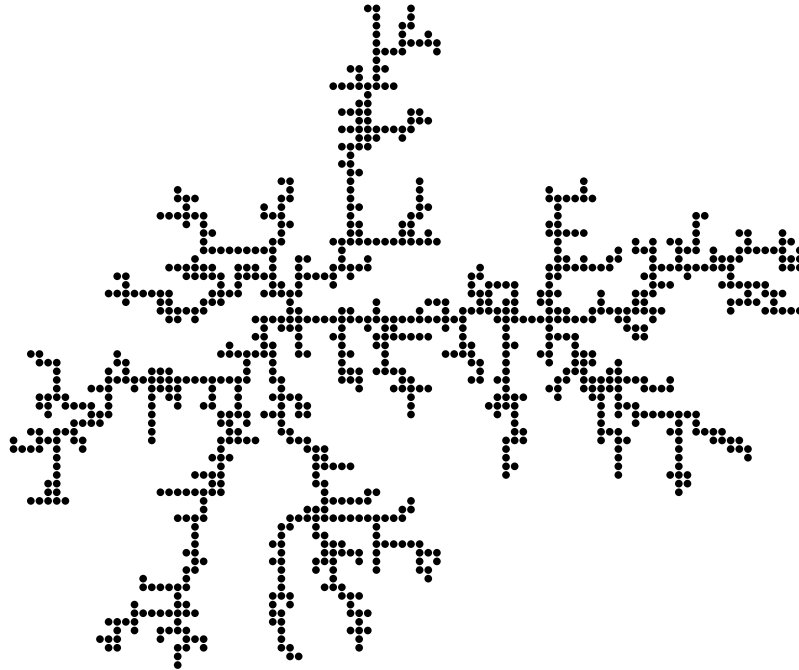


Figure 13.12: An example of a DLA cluster of 1000 particles on a square lattice.

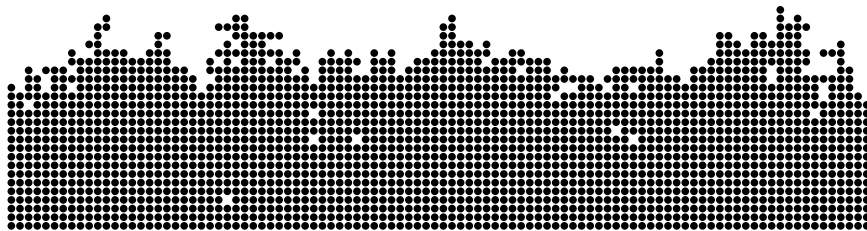


Figure 13.13: Surface growth according to the Eden model. The surface site in column x is the perimeter site with the maximum value h_x in the vertical direction. The average height for this surface is 20.46 and the width is 2.33.