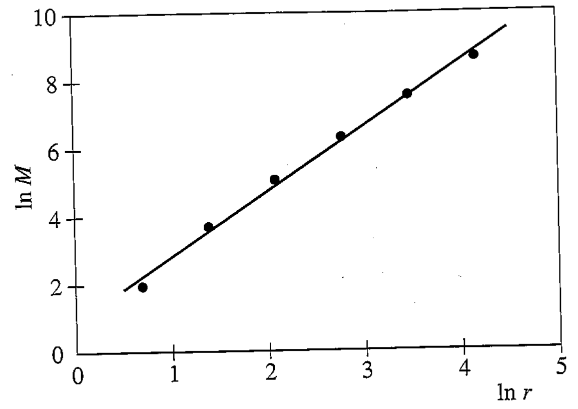

Figure 13.4 Plot of $\ln M$ versus $\ln r$ for a single spanning percolation cluster generated at $p = 0.5927$ on a $L = 129$ square lattice. The straight line is a linear least squares fit to the data. The slope of this line is 1.91 and is an estimate of the fractal dimension D. The exact value of D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

(b) 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?

(c) Choose $p = 0.5927$ and $L \geq 61$ and generate at least ten configurations of spanning clusters. Determine the number of occupied sites $M(r)$ within a distance r of the seed site of each cluster. (Better results can be found by choosing the origin to be center of mass of each cluster.) Average $M(r)$ over the 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) Generate clusters at $p = 0.65$, a value of p greater than p_c, for $L = 101$. Make a log-log plot of $M(r)$ versus r. Is the slope approximately equal to the value of D found in part (c)? 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?

(e) The fractal dimension of percolation clusters is not an independent exponent but satisfies the scaling relation

$$D = d - \beta/\nu, \tag{13.5}$$

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

$$M(L) \sim P_\infty(L)L^d, \tag{13.6}$$

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 (12.11)), and we can write

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

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?

*(f) Rewrite the `SingleCluster` class so that the lattice is stored as a one-dimensional array as is done for class `Clusters` in Chapter 12.

*(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?

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 Listing 13.3. Note that method `iterate` calls itself. Use class `KochApp` to generate the curves shown in Figure 13.5.

Listing 13.3 Class for drawing the Koch curve.

```
package org.opensourcephysics.sip.ch13;
import java.awt.Graphics;
import org.opensourcephysics.controls.*;
import org.opensourcephysics.frames.*;
import org.opensourcephysics.display.*;

public class KochApp extends AbstractCalculation implements Drawable {
   DisplayFrame frame = new DisplayFrame("Koch Curve");
   int n = 0;

   public KochApp() {
      frame.setPreferredMinMax(-100, 600, -100, 600);
      frame.setSquareAspect(true);
      frame.addDrawable(this);
   }

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