Lecture 6: Fractals from Iterated Function Systems

He draweth also the mighty with his power: Job 24:22

1. Generating Fractals by Iterating Transformations

The Sierpinski gasket and the Koch snowflake can both be generated in LOGO using recursive turtle programs. But in CODO there is no recursion. How then can we generate fractals in CODO?

Consider the first few levels of the Sierpinski gasket depicted in Figure 1. We begin with a triangle. The next level contains three smaller triangles. (Ignore the central upside-down triangle.) Each of these smaller triangles is a scaled down version of the original triangle. If we denote the vertices of the original triangle by P_1, P_2, P_3, then we can generate the three small triangles by scaling by $1/2$ around each of the points P_1, P_2, P_3 -- that is, by applying the three transformations $Scale(P_1, 1/2)$, $Scale(P_2, 1/2)$, $Scale(P_3, 1/2)$ to $\Delta P_1 P_2 P_3$. Now the key observation is that to go from level 1 to level 2, we can apply these same three transformations to the triangles at level 1. Scaling by $1/2$ at any corner of the original triangle maps the three triangles at level 1 to the three smaller triangles at the same corner of level 2. We can go from level 2 to level 3 in the same way, by applying the transformations $Scale(P_1, 1/2)$, $Scale(P_2, 1/2)$, $Scale(P_3, 1/2)$ to the triangles in level 2. And so on and so on. Thus starting with $\Delta P_1 P_2 P_3$ and iterating the transformations $Scale(P_1, 1/2)$, $Scale(P_2, 1/2)$, $Scale(P_3, 1/2)$ n times generates the *nth* level of the Sierpinski gasket.

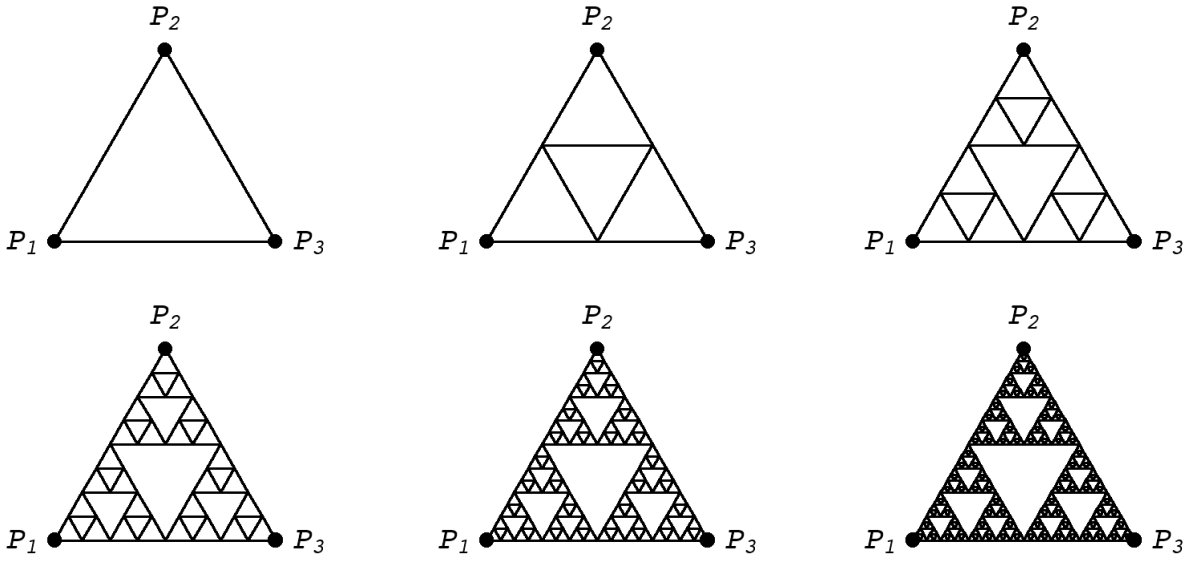

Figure 1. Levels zero through five of the Sierpinski gasket.

Let's try another example. Consider the first few levels of the Koch curve illustrated in Figure 2. Here we begin with a triangular bump. The next level contains a small triangular bump on each line of the original bump. Each of these smaller bumps is a scaled down version of the original bump. Thus there are four scaling transformations that map the original bump at level 0 to the four smaller bumps at level 1. Once again the key observation is that to go from level 1 to level 2, we can apply these same four transformations. Now we can go from level 2 to level 3 in the same way; and on and on. Thus starting with a triangular bump and iterating four scaling transformations *n* times generates the *nth* level of the Koch curve.

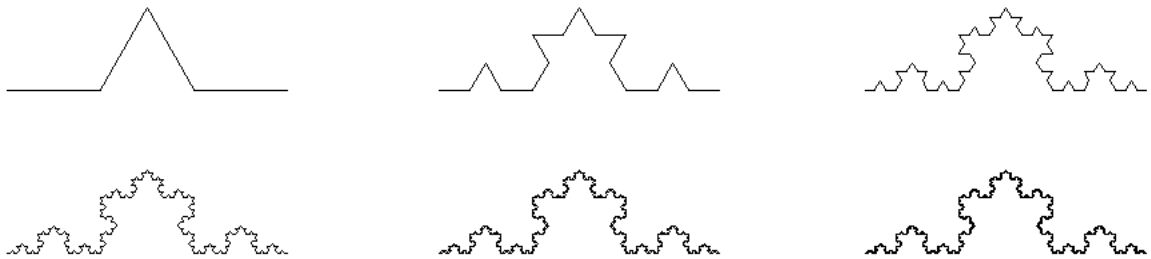

Figure 2. Levels zero through five of the Koch curve.

Evidently then, if we know the first few levels of a fractal, we can discover transformations that iteratively generate the fractal by finding transformations that generate level 1 from level 0 and then checking that these transformations also generate level 2 from level 1. But suppose we encounter a fractal with which we are not so familiar such as one of the fractals shown in Figure 3. If we have not seen these fractals before, we may not know what the first few levels should look like. We could, of course, write a turtle program to generate these fractals and then observe the first few levels of the turtle program, but this approach is rather tortuous. Fortunately, there is a better, more direct approach.

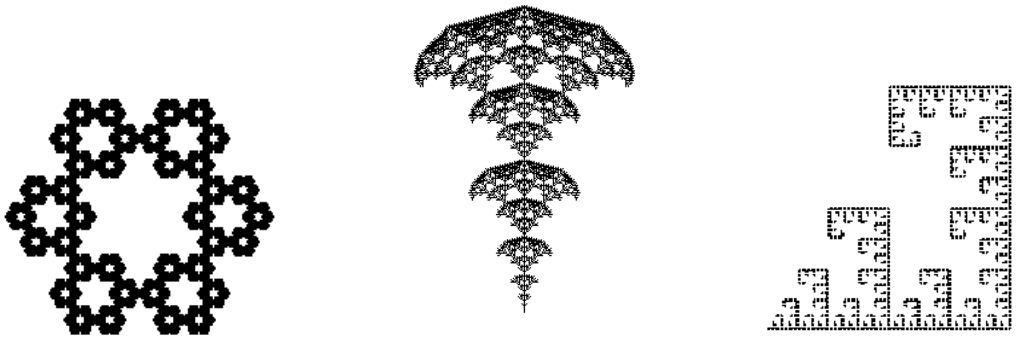

Figure 3. Three fractals for which the first few levels are not evident: a fractal flower, a fractal bush, and a fractal hangman.

2. Fractals as Fixed Points of Iterated Function Systems

There is a way to find the transformations that generate a given fractal without knowing the first few levels of the fractal. The Sierpinski gasket consists of three scaled down versions of the Sierpinski gasket. The transformations $Scale(P_1, 1/2)$, $Scale(P_2, 1/2)$, $Scale(P_3, 1/2)$ that generate the Sierpinski gasket map the Sierpinski gasket into these three scaled down copies. Similarly, the Koch curve consists of four scaled down versions of the Koch curve. The four transformations that map the *nth* level of the Koch curve to the $(n+1)^{st}$ level of the Koch curve also map the original Koch curve into these four scaled down copies.

These observations are not a coincidence. Suppose we have a collection of transformations that for all n map the *nth* level of a fractal to the $(n+1)^{st}$ level. Then in the limit as n goes to infinity, these transformations would necessarily map the entire fractal onto itself. The converse also turns out to be true. If we can find a set of transformations that map a fractal onto itself as a collection of smaller self-similar copies, then iterating these transformations necessarily generates the fractal. *Note that the requirement of smaller self-similar copies excludes the identity transformation.*

A set of transformations that generates a fractal by iteration is called an *iterated function system (IFS)*. An iterated function system maps the corresponding fractal onto itself as a collection of smaller self-similar copies. Fractals are often defined as *fixed points* of iterated function systems because when applied to the fractal the transformations that generate a fractal do not alter the fractal.

The iterated function systems that generate the fractals in Figure 3 are not difficult to construct. The fractal flower consists of six smaller self-similar parts, so six transformations are required to generate this fractal. The fractal bush has four smaller self-similar parts, so four transformations are required to generate this fractal. Finally, the fractal hangman consists of three smaller self-similar parts, so three transformations are needed to generate this fractal. We leave it as a straightforward exercise for the reader to find explicit expressions for these transformations. The required transformations are all affine, and can be constructed either by composing translation, rotation, and uniform scaling or by specifying the images of three points in the plane.

Example 1: The Koch Curve

We shall illustrate two ways to construct the four affine transformations $\{M_1, M_2, M_3, M_4\}$ for the IFS that generates the Koch curve. For the labeling of the points on the Koch curve, refer to Figure 4.

Method 1: Building each transformation by composing translation, rotation, and uniform scaling.

$M_1 = Scale(A, 1/3)$

$$M_2 = Scale(A, 1/3) * Trans\big(Vector(A,D)\big) * Rot(D, \pi/3)$$

$$M_3 = Scale(C, 1/3) * Trans\big(Vector(C,E)\big) * Rot(E, -\pi/3)$$

$$M_4 = Scale(C, 1/3)$$

Method 2: Building each transformation by specifying the image of three points.

$$M_1 = AffineTrans(A, B, C;\ A, F, D)$$

$$M_2 = AffineTrans(A, B, C;\ D, H, B)$$

$$M_3 = AffineTrans(A, B, C;\ B, I, E)$$

$$M_4 = AffineTrans(A, B, C;\ E, G, C)$$

Notice that in this example, both methods generate the same set of transformations. The first method may seem more natural, but the second method is often much simpler to apply. For yet another approach to deriving these same transformations, see Exercise 6.

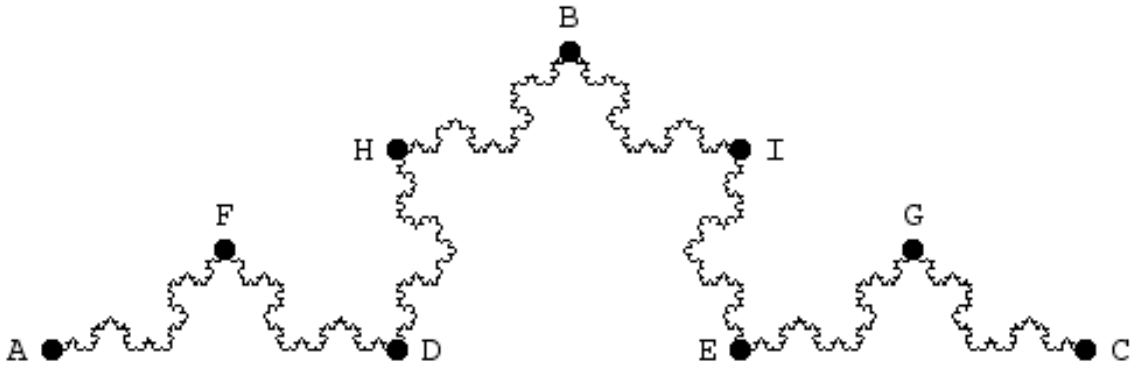

Figure 4: The Koch curve. Key points along the curve are labeled to help specify the four transformations in the IFS that generate this curve.

3. Fractals as Attractors

We have explained how to find an IFS to generate a given fractal curve, but we still do not know on what shape to start the iteration. That is, we still do not know how, in general, to find level 0 of a given fractal curve. For the Sierpinski triangle, we know to begin with an ordinary triangle, and for the Koch curve we know to begin with a triangular bump. But how do we begin for fractals with which we are not so familiar such as the fractals shown in Figure 3?

Fractals generated by recursive turtle programs are independent of the base case. Could the same independence property hold for fractals generated by iterated function systems? Yes! Figure 5 illustrates this independence for the Sierpinski gasket. The same IFS is applied to three different base cases: a square, a horizontal line, and a single point. In each case after a few iterations the figures all converge to the Sierpinski gasket.

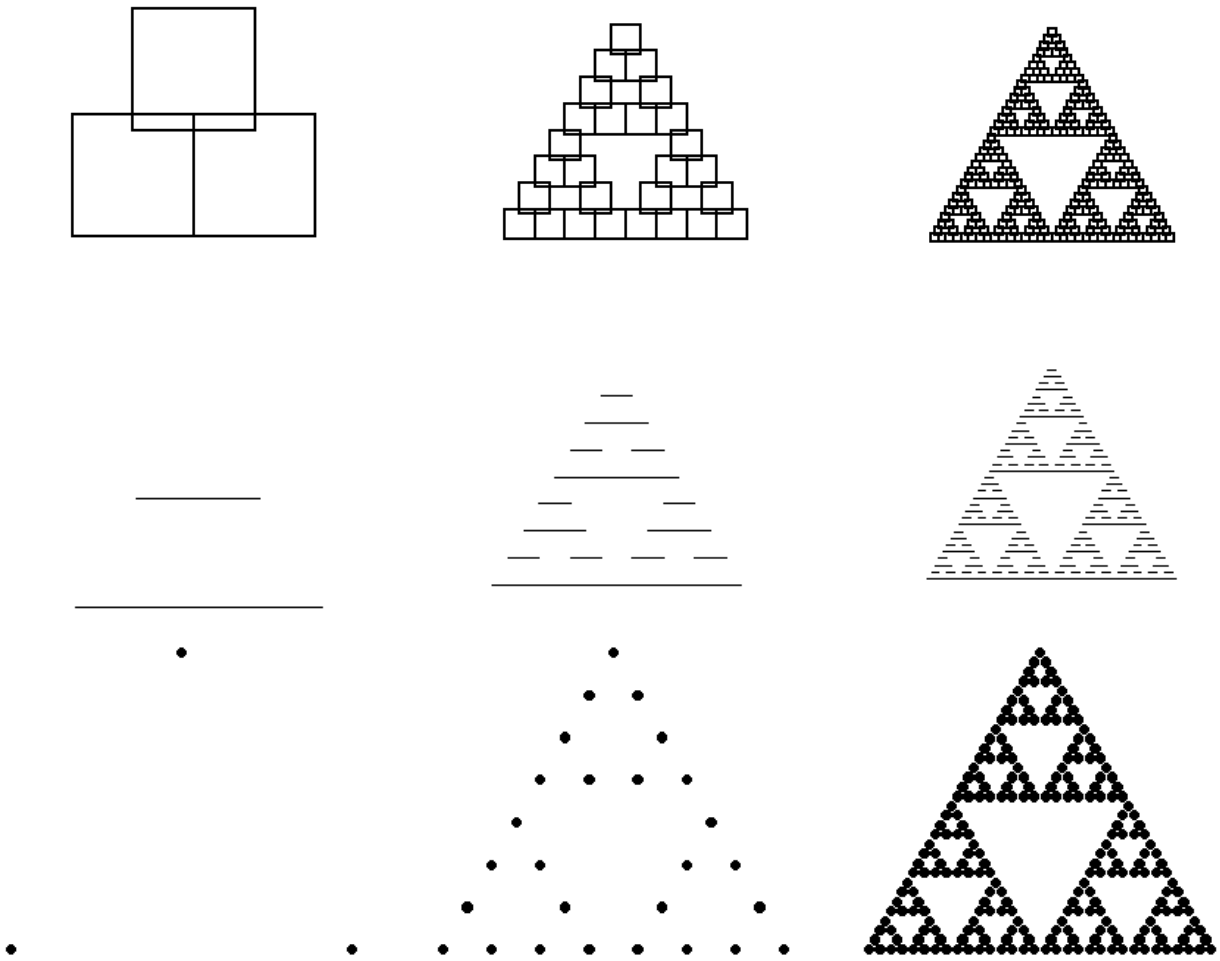

Figure 5: Levels 1,3,5 of the Sierpinski gasket. On the top, the base case is a square; in the middle, the base case is a horizontal line; on the bottom, the base case is a single point.

Fractals are attractors. No matter what geometry we choose for the initial shape, the subsequent shapes are inexorably attracted to the fractal generated by the IFS. We shall explain the reason for this attraction in the next lecture. For now it is enough to know that it does not matter how we choose the initial shape. The base case is irrelevant. If we choose the right IFS, the fractal we want will magically emerge no matter what geometry we choose at the start.

4. Fractals with Condensation Sets

There is another type of fractal that is relatively common: fractals with condensation sets. Consider the fractals in Figure 6. These fractals are not quite self-similar. The fractal staircase contains two smaller fractal staircases, but the largest square in the figure does not belong to either one of the two smaller staircases. Similarly, the fractal tree contains two smaller trees, but the trunk of the tree does not belong to either of these smaller trees. The square in the fractal staircase and the trunk of the fractal tree are called *condensation sets*. If a fractal consists of self-similar parts together with an additional set, the additional set is called a *condensation set*. Thus, in general, to generate a fractal, we need to specify three things: a set of transformations (an IFS), a base case (at which to start the iteration), and a condensation set (possibly the empty set).

There are actually two ways to deal with condensation sets. One approach is to take the base case to be the condensation set and include the identity transformation in the IFS. For the fractal staircase, the IFS consists of the two transformations that scale by 1 / 2 about the top and bottom of the staircase. If we add the identity to this IFS and take the large square as our base case, then iterating the IFS will generate a sequence of smaller and smaller squares that converge to the fractal staircase. Notice that because of the identity function in the IFS, the large square is included at every level of the iteration (see Figure 7).

The advantage of this technique is that it is simple, but the disadvantage is that the identity function must be applied at every level to each square in the figure. Moreover, in this approach the fractal is no longer independent of the base case. An alternative method is simply to add the condensation set back into the curve at every stage after applying the transformations. This approach has the advantage that we do not waste time redundantly applying the identity function at every level to every square. We can also use any base case; we are no longer restricted to starting with the condensation set. This approach is illustrated for the fractal staircase in Figure 8.

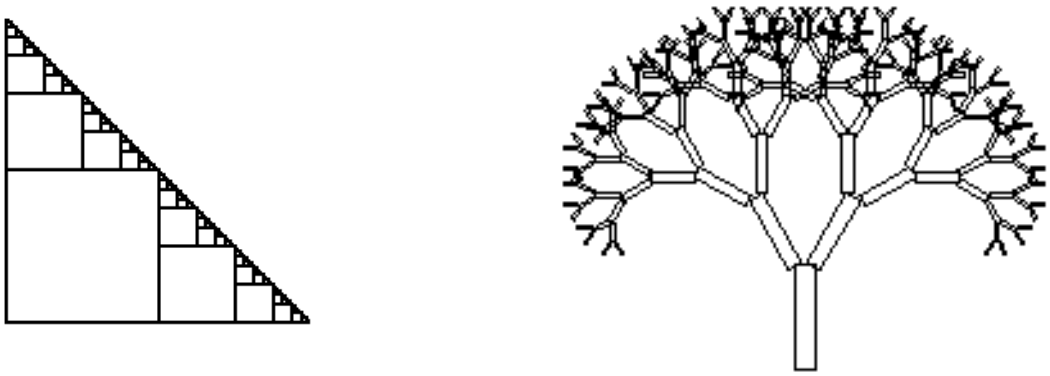

Figure 6: Fractals with condensation sets: a fractal staircase and a fractal tree.

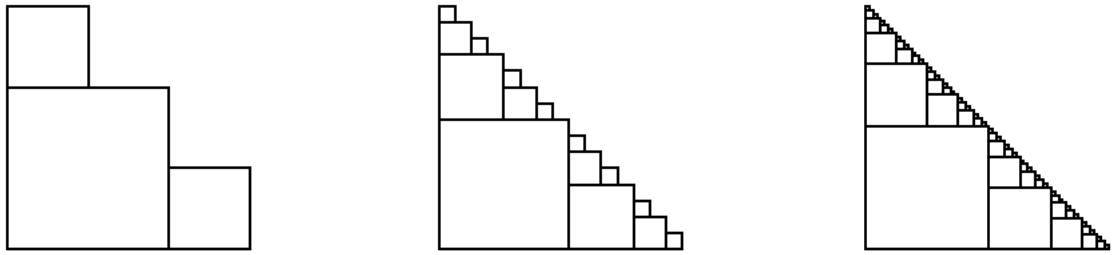

Figure 7: Levels 1,3,5 of the fractal staircase. The base case is the large square, which is also the condensation set. Here the identity transformation is included in the IFS.

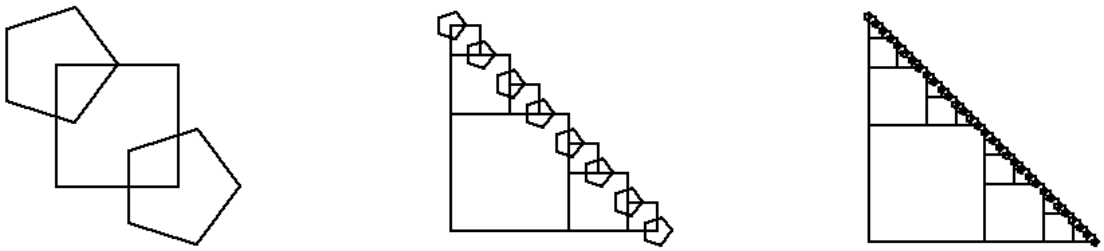

Figure 8: Levels 1,3,5 of the fractal staircase. The base case is a pentagon and the condensation set is the large square. Notice that the pentagons converge to the diagonal of the staircase, and that the square generates the main part of the fractal. Here the identity transformation is not included in the IFS, but at each level after applying the transformations in the IFS, the large square is added back into the curve.

5. Summary

Fractals are fixed points of iterated functions systems. This observation allows us to find transformations that generate a given fractal curve by finding a set of transformations that map the fractal onto itself as the union of a collection of smaller self-similar copies.

Fractals are attractors. No matter what base case we choose to initiate the iteration, for a fixed IFS the shapes we generate will automatically converge to the identical fractal.

Fractals may have condensation sets. We can incorporate these condensation sets into our scheme for generating fractals from an IFS by adding the condensation set back into the curve at every stage of the iteration after applying the transformations in the IFS.

Exercises

1. Find an IFS to generate a polygonal gasket with n sides.

2. Find an IFS to generate each of the fractals in Figure 3 by:
   a. composing translations, rotations, and uniform scalings
   b. specifying the images of three points in the plane.

3. Find an IFS to generate each of the fractals in Lecture 2, Figure 8.

4. Find an IFS to generate each of the fractals in Lecture 2, Figure 16.

5. Find explicit expressions for the IFS and the condensation set of the fractal tree in Figure 6.

6. Consider the Koch curve in Figure 4. Let M denote the point midway between D and E, and let $v = E - D$.
   a. Show that the Koch curve is generated by the IFS containing the four transformations:
   $T_1 = Scale(M, 1/3) * Trans(v)$, $T_2 = Scale(M, 1/3) * Trans(-v)$,
   $T_3 = Scale(M, 1/3) * Rot(D, \pi/3)$, $T_4 = Scale(M, 1/3) * Rot(E, -\pi/3)$.
   b. Verify that these transformations are identical to the transformations in Example 1.

7. Prove that for any self-similar fractal generated by an iterated function system,
$$Fractal\ Dimension = -\frac{Log(\#\ Transformations\ in\ the\ IFS)}{Log(Scale\ Factor)}.$$

8. Using the result of Exercise 7, compute the fractal dimension of the fractals in Figure 3.

9. The Cantor set is generated by starting with the unit interval and recursively removing the middle third of every line segment.
   a. Show that the Cantor set is the fractal generated by the IFS consisting of the two conformal transformations $Scale(Origin, 1/3)$ and $Scale(P_1, 1/3)$, where $P_1 = (1,0)$.
   b. Using part *a* and the result in Exercise 7, show that $0 < Dimension(Cantor\ Set) < 1$.
   c. Show that the length of the Cantor set is zero.
   d. Consider the set generated by starting with the unit interval and recursively removing the middle half of every line segment.
      i. Find the transformations in the iterated function system that generate this fractal.
      ii. Using the result in Exercise 7, show that the dimension of this set is $1/2$.

10. Consider the fractal pentagram and the fractal diamond in Figure 9.
    a. Find a condensation set and an IFS to generate the fractal pentagram.
    b. Find a condensation set and an IFS to generate the fractal diamond.

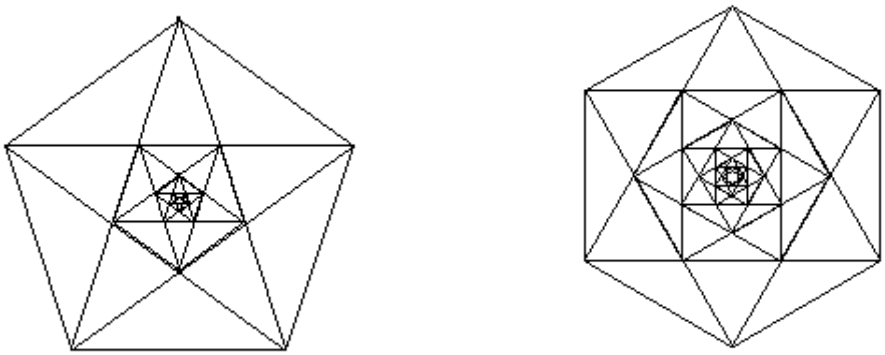

Figure 9: A fractal pentagram (left) and a fractal diamond (right).

11. Consider the fractal tower and the fractal star in Figure 10.
    a. Find a condensation set and an IFS to generate the fractal tower.
    b. Apply the SPIN operator to the fractal tower to generate the fractal star.

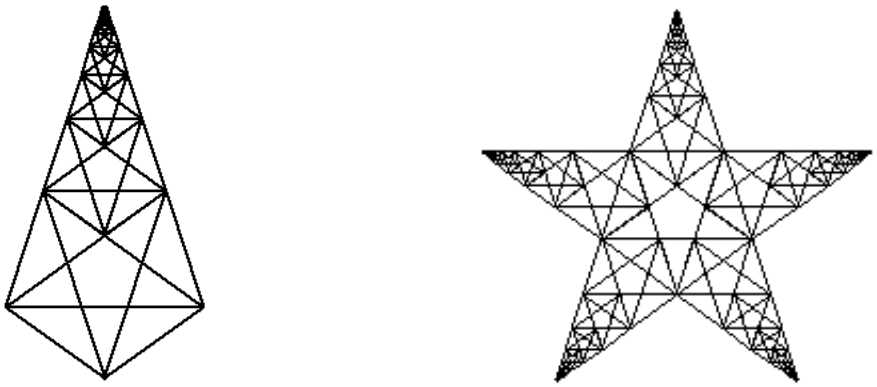

Figure 10: A fractal tower (left) and a fractal star (right). The size of successive stars in both figures varies by the golden ratio.

Programming Project:

1. *CODO*

   Implement CODO in your favorite programming language using your favorite API.

   a. Write CODO programs for the curves discussed in the text and in the exercises for Lectures 5 and 6.
   b. Write a CODO program to design a novel flag for a new country.
   c. Develop some iterated function systems to create your own novel fractals with and without condensation sets.