2.8 Experiment: Collages of Transformations

In Section 1.5 we saw that the Sierpinski triangle could be created by a process of juxtaposition. Figure 1.7.1 shows the result. The symmetry appearing in the resulting fractal agrees with the following observations. If we contract the entire fractal by half, we get the lower left portion of the fractal. If we contract by half and add half in the vertical direction, then we obtain the upper left portion of the fractal. If we contract by half and then add half in the horizontal direction, then we obtain the lower right portion of the fractal. The following three matrices and the corresponding transformations implement the three transformations just described.

```
   m0                    m1                    m2
0.5   0 0           0.5   0 0             0.5   0 0
  0 0.5 0             0 0.5 0               0 0.5 0
  0   0 1             0 0.5 1             0.5   0 1

   mp=:+/ . *

   t0=: mp & m0          t1=: mp & m1          t2=: mp & m2
```

We create a gerund from those three transformations and evaluate all three transformations via `` `:0 `` in the function collage below. We define the unit square in homogeneous coordinates and see that the image of the square under the function `collage` is three smaller squares. We can then look at the image of those three squares after applying `collage` again and we get 3 sets of 3 squares (that is, 9 of them), and so on. Figure 2.8.1 shows four iterates of `collage` on the original square. Notice the form of the Sierpinski triangle is beginning to appear. The script *dwin.ijs* should be loaded.

```
   collage=: t0`t1`t2`:0

   sq=: 0 0,1 0,1 1,:0 1

   ]sq=: sq,.1
0 0 1
1 0 1
1 1 1
0 1 1

   collage sq
  0   0 1
0.5   0 1
0.5 0.5 1
  0 0.5 1

  0 0.5 1
0.5 0.5 1
0.5   1 1
  0   1 1

0.5   0 1
  1   0 1
  1 0.5 1
0.5 0.5 1

   $collage^:2 sq
3 3 4 3

   dwin 'collage 4'

   0 255 0 dpoly collage^:4 sq
```

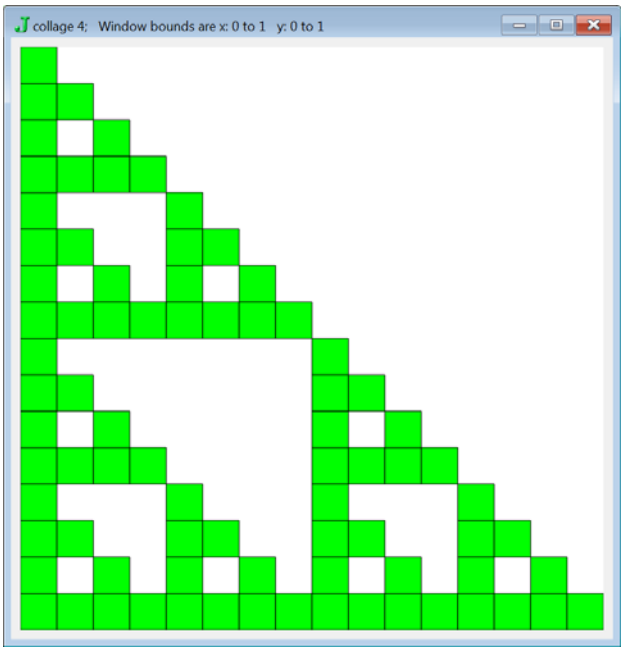

Figure 2.8.1 Collage Applied Four Times

What happens when you run six or seven iterates of `collage`? What does the fractal look like when the transformation `t2` is replaced by the transformation associated with one of the following matrices? The function `rotm` from Section 2.6 is assumed to be defined.

```
   ]m3=: m0 mp (rotm 1r60p1) mp (=i.2),0.5 0 1
 0.499315 0.026168 0
_0.026168 0.499315 0
      0.5        0 1

   ]m4=: m0 mp (rotm 1r6p1) mp (=i.2),0.5 0 1
0.433013     0.25 0
   _0.25 0.433013 0
     0.5        0 1

   ]m5=: m0 mp (rotm 1r2p1) mp (=i.2),1 0 1
   0 0.5 0
_0.5   0 0
   1   0 1
```

The list of functions we have given in collage are known as iterated function systems. There is a rich theory of these systems and we will return to further consideration of them in Section 4.4.

2.9 Simple Verbs

In Section 1.10 we saw that we can create very rich nouns; this includes character and boxed arrays. Verbs and higher order parts of speech are also available in J, but it is important to be comfortable with very basic verbs. This includes constant and identity functions. These verbs may seem almost trivial, and in some sense they are very simple; however, their use can also be subtle. We have already seen that their inclusion in the language makes it easy to express some functions very gracefully.

First we recall some of the basic verbs that we have already used briefly. Recall that the dyad `[` denotes left and `]` denotes right. Thus we can implement the dyad $h(x, y) = \sin(x) + \sin(y) + \sin\left(\sqrt{x^2 + y^2}\right)$ as follows.

```
   sin=: 1&o.

   h=: sin@[ + sin@] + sin@%:@+&*:

   4 h 6
_0.235845
```

We see that left and right can almost be thought of as standing for the "dummy" variables *x* and *y* in standard math notation. For example, we could have implemented $\sqrt{x^2 + y^2}$ that way.

```
   s=: %:@(*:@[ + *:@])

   3 s 4
5
```

In Section 2.6 we saw that constant functions for `0` and `1` were denoted `0:` and `1:` respectively. The notation for constant functions using a colon suffix is available for the integers between `_9` and `9`. However, we can change any noun into a verb of suitable rank by giving the noun as a left argument to rank. Recall that `_` denotes infinity, so `"_` denotes rank infinity. That is, it applies to the entire noun.

```
   3: i.5                    the verb 3: always results in 3
3

   (3: % 8:) i.5             a fork giving constant function three-eighths
0.375
```