Int J Comput Math Learning (2008) 13:83–88
DOI 10.1007/s10758-008-9128-4

COMPUTATIONAL DIVERSIONS

Rounded Fractals

Michael Eisenberg

Published online: 8 April 2008

> ***A prefatory note to readers:*** Please feel free–encouraged, in fact–to send in your own ideas, puzzles, challenges, solutions, and so forth to this column! This is intended to be an interactive feature of IJCML, and an opportunity for enjoyable conversation between all of us. Speaking of which: at the close of this column–after presenting two new challenges–I have reprinted a reader's solution to the column 1 challenge.

One could hardly expect a column of this sort to go for a while (in this case, three iterations) without a mention of fractals. It's fair to say that generating fractal designs is one of the most compelling means for introducing students to experimental mathematics. My first introduction to such figures was in Abelson and diSessa's book *Turtle Geometry* (Abelson and diSessa 1980), in which the authors use Logo-style turtle walks to create the Koch snowflake, Sierpinski triangle, and numerous other beautiful self-similar patterns.

The programs to be developed in this column will not employ turtle walks to produce fractal designs, but will instead make use of the (probably even better known) method of iterated function systems (IFS's). The basic idea behind an IFS is that one begins with a collection of functions (we'll employ maps from R2 to R2); in the simplest case, each of the maps in our collection is affine, with a determinant less than one. (In other words, each map is composed of an area-reducing linear map followed by a translation.) Then, we employ the following algorithmic strategy:

Algorithmic Strategy for creating an IFS Fractal

1. Begin with an arbitrary starting point P0. This will be our "current point".
2. Plot the current point.
3. Choose at random one of the maps in the IFS, and apply that map to the current point. The result of this application will be the new current point.
4. Go back to step 2.

M. Eisenberg (✉)
University of Colorado, 430 UCB, Boulder, CO 80309-0430, USA
e-mail: ijcml-diversions@ccl.northwestern.edu

That's really just about all that's involved, though it's typical to add in a few extra features. For instance, the loop represented by steps 2 through 4 above will only be repeated for some finite number of times, and usually we run the loop a few initial times without plotting any points (that is, we begin plotting after, say, the fifth or tenth or twentieth iteration). This is not the appropriate venue for a thorough introduction to iterated function systems: my favorite treatment of the subject is Michael Barnsley's text *Fractals Everywhere* (Barnsley 1993), in which the author presents an astonishing variety of beautiful programs and graphical effects created by these systems. (The book also provides a good, though not particularly easy, introduction to the mathematical theory behind these systems—for example, Barnsley shows how the fractal set produced by a particular IFS can be viewed as an attractor of a dynamical system associated with that IFS.)

Despite the wealth of examples in Barnsley's book—not to mention all the websites, calendars, and so forth devoted to fractals—it seems to me that there are galaxies of new, creative algorithmic ideas waiting to be tried. Moreover, many of these ideas are relatively easy to express, and to implement, with only a modicum of programming: mild variants of the basic recipe described above. The idea behind this column is just one representative example—and as far as I'm aware, it has not been presented elsewhere.

Here's the origin of the idea. Despite the visual appeal of many of IFS-generated fractals, they often tend to have a "prickly" or sharp-edged look to them. Consider, for example, the standard Sierpinski triangle, shown in Fig. 1. This triangle is generated by an IFS with three component maps:

*Sierpinski-Map*1: For a given input point: Scale both coordinates of the point by 0.5, then translate the resulting point by vector (0, 0.5)
Sierpinski-Map2: Scale by 0.5, translate by vector (−0.433, −0.25)
Sierpinski-Map3: Scale by 0.5, translate by vector (0.433, −0.25)

Figure 2 shows an equilateral triangle of radius 1 (i.e., a triangle that could be inscribed in the unit circle), and the result of applying each of the three component maps individually to that triangle. When these three maps are combined into an IFS and employed in the algorithmic strategy described earlier, the result is the picture shown in Fig. 1. One can

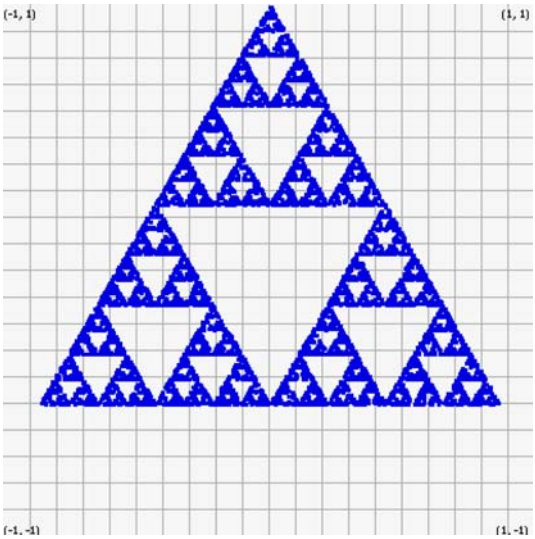

Fig. 1 The Sierpinski triangle, as generated by the three IFS maps indicated in the text

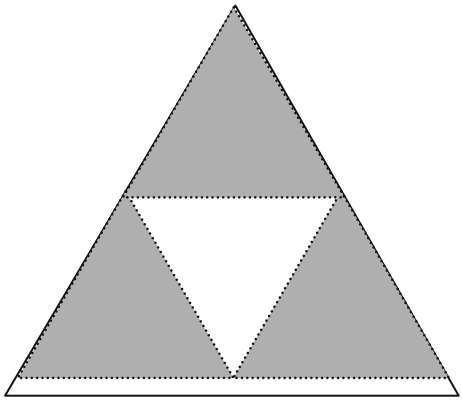

Fig. 2 The outer equilateral triangle (of radius 1), centered at the origin, has been scaled (by 0.5) and translated to produce the three smaller light gray triangles visible within its boundaries. The three gray triangles, then, correspond to the operation of each of the three IFS Sierpinski maps shown in the text

think of this figure (or really, the attractor set associated with the IFS) as the unique figure that remains invariant under the superposition of the three maps. Put less formally, our Fig. 1 triangle can be recreated by shrinking the original down by a factor of 0.5 and placing the three smaller copies down on the original as suggested by the three translation vectors of Fig. 2.

Suppose, however, we wish to "round the edges" of our triangle. A somewhat ham-fisted way of going about this would be to avoid plotting points outside a particular radius; this would produce a figure consisting of a (portion of a) Sierpinski triangle as seen through a circular porthole. The resulting figure would lose, however, the property of self-similarity apparent in Fig. 1: that is, we would be viewing a truncated figure that is only part of a larger, self-similar design (see Fig. 3).

We can try something else, however: rather than restrict ourselves to affine maps as in the Fig. 1 set, we can design our component maps to produce a "rounded IFS fractal", as shown in Fig. 4. The way that we will do this is to ensure that our maps never produce a point outside a circle of a given radius r; rather, any point that would be sent outside the

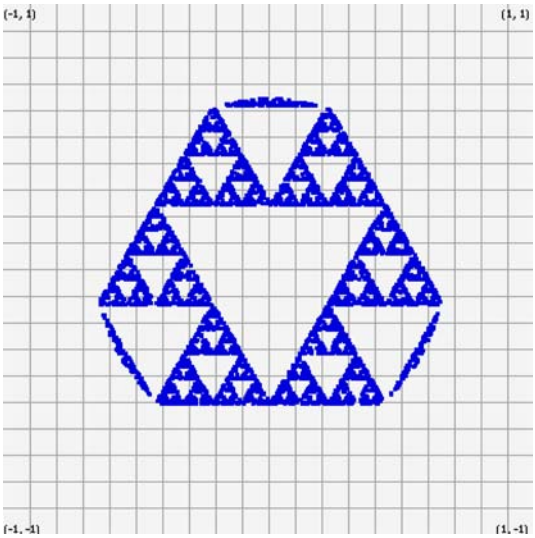

Fig. 3 The Sierpinski triangle of Fig. 1, "masked" by a circular window. (That is, we can only see that portion of the original figure within a certain radius—here, 0.65—from the origin.)

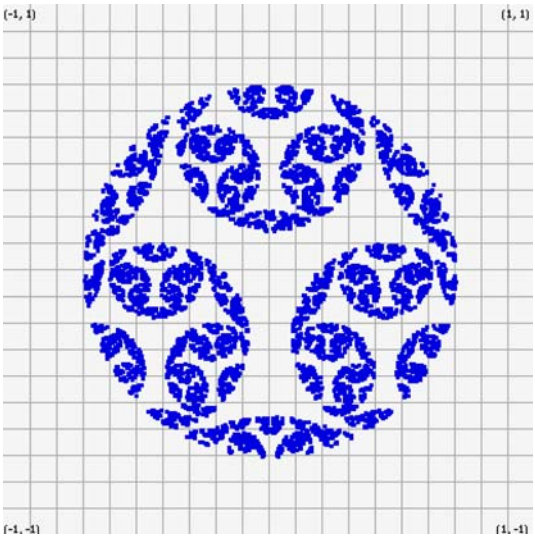

Fig. 4 A "circularized" version of the Sierpinski pattern shown in Fig. 1

desired circle by a standard Sierpinski map is then reflected back inside that circle and rotated. Here is the general form of our new maps to replace the three originals:

Circular-Sierpinski-Map-1:
For a given point p:
First scale the point by 0.5 and translate by (0, 0.5) to produce p'.

If p' is within the radius r from the origin, return p'
as the result of this map.
If p' is outside the desired radius r, invert p' in the circle of radius r to produce p''; then rotate p'' by a given angle theta to produce the output point p'''.

Circular-Sierpinski-Map-2:
Similar to Map-1, except the translation vector is $(-0.433, -0.25)$

Circular-Sierpinski-Map-3:
Similar to Map-1, except the translation vector is $(0.433, -0.25)$

Figure 4 is the result of using our three new "circular" maps in the algorithmic skeleton shown earlier. In Fig. 4, we have chosen a radius value of 0.7, and a rotation theta of 60 degrees. The resulting figure is not self-similar, but if you stare at it you will see that the entire circular figure may be shrunk down into three half-sized copies which are then "torn", bent, and placed back on top of the original to reconstitute the starting figure. Or, to put it another way: we don't have a self-similar figure, but we have something that has a visual flavor of self-similarity—enough to produce visual interest.

1 Challenge

Now that we have successfully "circularized" the original Sierpinski triangle, we can try out the same sort of idea on other standard IFS-generated fractals. For example, Fig. 5

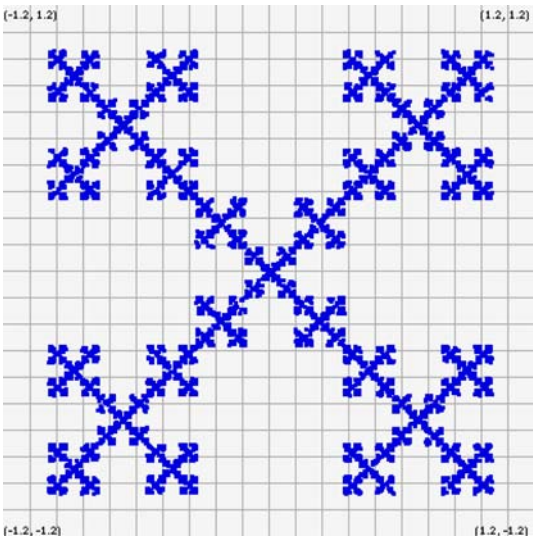

Fig. 5 A standard IFS fractal generated by five component maps, each of which consists of a uniform scale (by one-third) followed by a translation step

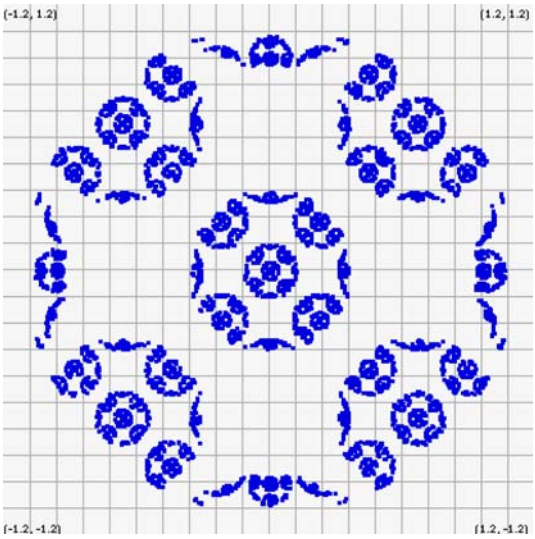

Fig. 6 A "circularized" version of the Fig. 5 pattern, generated with the same basic alteration that we used for the Sierpinski triangle

shows a fractal generated from five component maps; like the Sierpinski maps, each of the Fig. 5 maps may be described as a scale step followed by a translation step. (For these five maps, however, the scale factor is one-third, rather than one-half as in the Sierpinski maps.) The reader is encouraged to try a couple of projects. The first—the standard project—is to use the "standard" algorithmic skeleton and to find the five affine component maps that generate the fractal in Fig. 5. The second—the new project—is to use the same idea that we have already employed to generate the "circularized" fractal in Fig. 6, or something like it (the specific pattern will depend on your choice of radius for the inversion circle). Adventuresome readers who play with this technique for circularizing new fractals—or who wish to pass along similarly offbeat ideas—should send suggestions by email to this column. Correspondence should be sent to the email address given above. Please put

"IJCML Puzzle Column" in the subject line, followed by a word or two indicating what type of correspondence this is (e.g., "IJCML Puzzle Column Experimental Result", "IJCML Puzzle Column Algorithmic Idea", and so forth).

2 A Solution to the Column 1 Challenge

In the first diversions column (12:1), readers were left with a challenge. Five pictures were shown, each composed of red, blue, and yellow pixels; and each picture was created using the same basic algorithm, but with a critical Manhattan distance *D* that varied from 1 to 5. This critical distance dictated the placement of pixels; essentially, yellows did not want to be exactly *D* away from each other, but within *D* of red and blue, while blues did not want to be exactly *D* away from each other, but within *D* of red. Here's Brian Harvey's (characteristically, beautifully-written) solution to the puzzle presented in the first diversions column:

> So, the checkerboard looking ones have to have D odd, because they're full of pairs of yellows even distances apart. (By the way, let me be the 400th person to tell you that it's not nice making us guess which grey level is which color!)
>
> In particular, the extreme checkerboard on page 84 is D = 1, because it has no adjacent same-color pairs at all.
>
> So the top row of page 83 are D = 3 and D = 5 in some order. To figure out which, you just find a pair of yellows three apart; the one that jumped out at me was the
>
> ```
> ..Y..
> YY.YY
> ..Y..
> ```
>
> structure in the left picture, which has several pairs of yellows three apart. So the left one is D = 5 and the right one is D = 3.
>
> Of the even ones, the domino-looking left picture is clearly trying hard to avoid any two-apart pairs, whereas there are several such in the right picture. So the left one is D = 2, the right one is D = 4.

An editor's note on Brian's admonition about color figures is in order. The short version: er, sorry! The long version: since color figures are expensive, they will unfortunately be rare in the printed version of this column. However, the online version of IJCML is in full color and you can view all figures there. As we all know, color can be a beautiful and at times necessary element of computational play. In the future, when color plays an important role in a figure or problem, I will place color figures on a public website associated with this column and point readers to that site.

References

Abelson, H., & diSessa, A. (1980). *Turtle geometry*. Cambridge, MA: MIT Press.
Barnsley, M. (1993). *Fractals everywhere* (2nd edn.). San Francisco, CA: Morgan Kaufmann.