Int J Comput Math Learning (2008) 13:263–270
DOI 10.1007/s10758-008-9139-1

COMPUTATIONAL DIVERSIONS

Computational Diversions: Julia Variations

Michael Eisenberg

Published online: 2 December 2008

One of the favorite display shelves in every recreational programmer's cabinet of curiosities belongs to the Julia sets. Julia sets, named after the French mathematician Gaston Julia, have a variety of equivalent mathematical definitions; they are also remarkably easy graphical entities to generate with a minimal degree of programming.

Here's the basic idea: suppose you have a complex polynomial function f(z). (A good example-to-think-with is the quadratic function $f(z) = z^2 + c$, where c is some constant complex number.) If you iterate f(z) a very large number of times, starting from a particular initial value z0, and follow the values of the successive points under iteration, you will find that the sequence of complex number values either becomes arbitrarily large (goes to infinity), or it doesn't. So some starting z0 points "go to infinity" and some don't. Plot the values of z0 whose sequences go to infinity in one color (say, white); and those whose sequences don't in another (say, black). The black points constitute the filled Julia set of the function f, and the boundary between black and white is the Julia set.

Given this definition, you might imagine that these sets are of interest only to a dedicated subculture among mathematicians; but the graphical plots of the Julia sets are in fact remarkably appealing, and form an easily-programmed class of fractals for computer programming students. Here's the general algorithm (written in pseudocode) through which you can program a Julia set for a quadratic polynomial:

```
Define JuliaSet (c, zstart)
  n ← 0
  z ← zstart
  while (n < 40) and (magnitude (z) < 2)
    z ← z^2 + c
    n ← n + 1
  color zstart according to the final value of n
```

A good description of Julia sets along these lines can be found in A. K. Dewdney's book *The Magic Machine* (Dewdney 1990), and more technical discussions of Julia sets can be

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

found in just about any reference on fractals. The idea behind the algorithm is that you first select a constant *c* that you wish to investigate. Then, you can choose a color scheme telling you how to plot zstart depending on the final value of n; that is, the color is chosen according to whether the point "escaped" to infinity, and if so how quickly it escaped (i.e., how many iterations it took for the point to escape). A small value of n means the point escaped quickly; a value of 40 means that, as far as your algorithm can tell at this level of accuracy, the point doesn't escape at all. Note that this algorithm shows how to plot the color for one point, zstart; to make a "Julia set picture" you plot a pixel value corresponding to zstart for an entire array of pixels on the screen. By fiddling with the choices of colors corresponding to values of n, you can get a range of gorgeous fractal pictures: if you don't believe me, do a web search on "Julia Sets" and take a look at some of the images that emerge from this simple algorithm. (One final tiny point: the algorithm's test for "going to infinity" is that the magnitude of the iterated value becomes greater than or equal to 2. Without going into detail, it should be clear that as long as the value of c has a complex magnitude less than 2, once the sequence of values goes beyond this magnitude it will in fact get larger and larger in magnitude at each iteration. So we'll assume, for the purposes of this exposition, that our values of the constant *c* are chosen to be sufficiently small.)

All of this is prelude to the purpose of this column, which is to go beyond the usual recipes for creating Julia sets, and to explore a few natural variations—natural, that is, to the recreational programmer. After all, looking at the algorithm above, it seems as though there are all sorts of elements that one could vary. Probably the most immediate experiment to try is to vary the function itself—instead of a quadratic polynomial, one could find the Julia set of a cubic, or indeed of any complex polynomial whatever. In fact, beautiful pictures of Julia sets of these sorts can also be found in the literature, and I encourage the reader to explore on his or her own.

For the purposes of this column, I'll show some pictures that were generated by varying the basic algorithm in still other ways. Let's begin:

1 Variation 1. Alternating values of c

In our standard algorithm, we fixed a value of *c* and plotted a Julia set for a range of starting points given that fixed value. Suppose, on the other hand, we try something like this:

```
Define JuliaSetVariation1 (c1, c2, zstart)
  n ← 0
  z ← zstart
  while (n < 40) and (magnitude (z) < 2)
    if n is even: z ← z^2 + c1
    if n is odd: z ← z^2 + c2
    n ← n + 1
  color zstart according to the final value of n
```

In other words, we'll start with two constants, *c*1 and *c*2, and alternate between them at each iteration of the algorithm. What sorts of pictures might we generate then?

Figures 1 and 2 show the result of this algorithm when the two complex numbers are chosen as (−0.8 + 0.36i) and (−0.4 + 0.18i). The first picture shows the "escape map" for this algorithm in a box ranging from (−0.5, −0.5) at bottom left to (0.5, 0.5) at upper

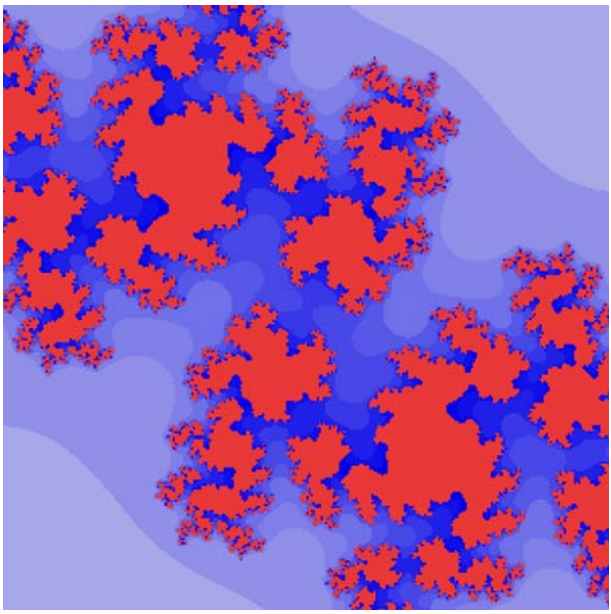

Fig. 1 An "alternating-value" escape map

right. The second picture shows a "close-up" of the first, in the box ranging from (−0.1, −0.1) at bottom left to the origin at upper right. I have chosen the same color scheme[1] for all the pictures in this column: orange corresponds to the "non-escaping points", while increasingly deep shades of blue correspond to values going from 0 to 39. The resulting pictures have a "Julia-set-like" look to them, but they don't quite match any standard Julia set that I have ever encountered.

Of course, these choices of $c1$ and $c2$ represent just one particular experiment (and I chose them so that $c2$ would be half the value of $c1$). Figure 3 represents still another pair of choices for $c1$ and $c2$: here, the values are (0.76 + 0.15i) and (−0.3 − 0.3i), respectively, and the points are plotted in the box ranging from (−1, −1) to (1, 1).

2 Variation 2. Choosing a Value of c from Among a Discrete Set

In the first variation above, we simply changed the original algorithm so that, instead of iterating the function with one value of c, we switched back and forth between two alternating values. The enterprising reader might extend this idea to switching off between (say) three values $c1$, $c2$, and $c3$, or an even larger set. I wished to try a slightly different idea, in which we choose between $c1$ or $c2$ depending on which value brings us closer to the origin. In other words, we'll vary the algorithm as follows:

```
Define JuliaSetVariation2 (c1, c2, zstart)
  n ← 0
```

[1] Color versions of the pictures in this column may be found at the following website: http://www.cs.colorado.edu/~ctg/ijcml/

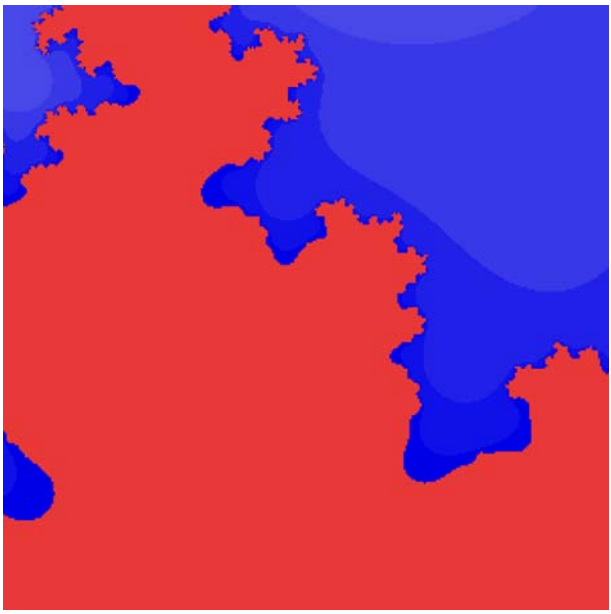

Fig. 2 The same escape map as in Fig. 1, at higher magnification

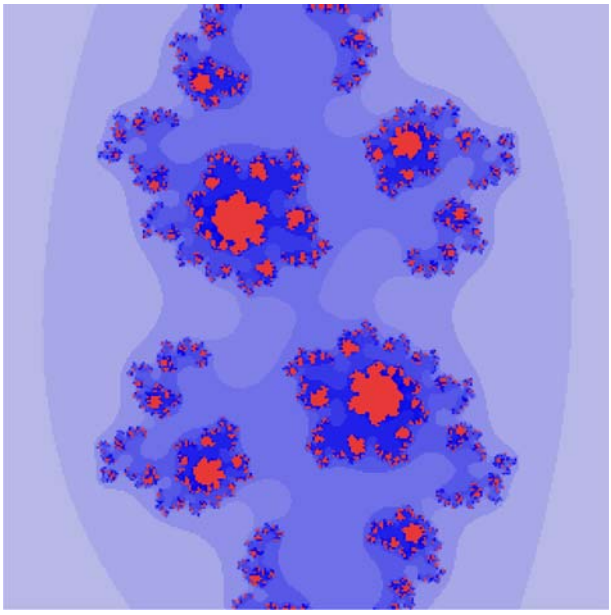

Fig. 3 Another "alternating-value" escape map

z ← zstart
while (n < 40) and (magnitude (z) < 2)
  z1 ← z^2 + c1

$z2 \leftarrow z^2 + c2$
$z3 \leftarrow z^2$
if magnitude(z1) > magnitude (z2) then $z \leftarrow z2$
if magnitude(z2) > magnitude (z1) then $z \leftarrow z1$
if magnitude(z1) = magnitude (z2) then $z \leftarrow z3$
$n \leftarrow n + 1$
color zstart according to the final value of n

The essential idea here is that we are trying to extend the "life" of our starting point by choosing a value of c at each iteration that will keep our successive values at lower magnitudes. The result of an experiment with this algorithm can be seen in Figs. 4 and 5 below. Here, the two values of c are $(-1.1 + 0.5i)$ and its complement $(-1.1 - 0.5i)$. Figure 4 shows a window between $(-0.5, -0.5)$ and $(0.5, 0.5)$, and Fig. 5 a close-up in the window from $(-0.25, -0.25)$ at bottom left to the origin at upper right. Here, as expected, there are much larger regions of points plotted in orange—that is, these are the regions that don't escape after 40 iterations. What's especially intriguing is that the blue (escaping) regions seem to have similar shapes; even under the increased magnification of Fig. 5, the small blue regions appear to be copies, or near-copies, of each other.

3 Variation 3. Making c a Function of the Current Value of z

In the previous variation, we essentially chose our value *c* depending on the current value of z at each iteration; but in that case, we only chose *c* among a small set of values. We could extend that idea by making the value of *c* a continuous function of z itself. In one

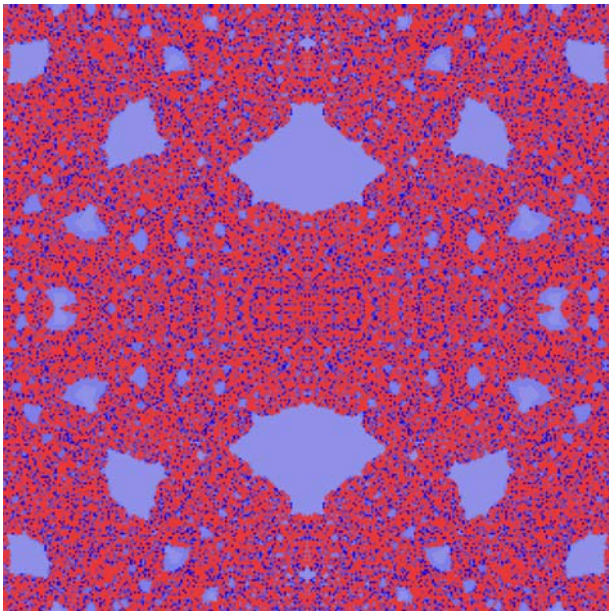

Fig. 4 A "choice between two constants" escape map (variation 2 in the text)

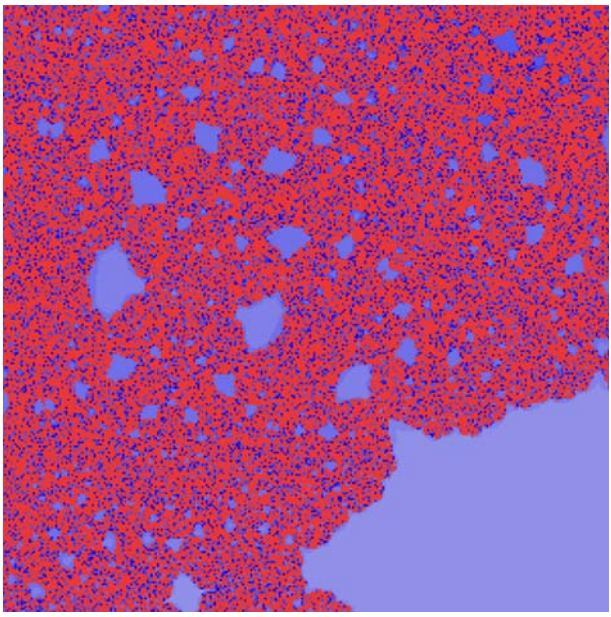

Fig. 5 The same map as in Fig. 4, at higher magnification. Note that the *darker gray* (escaping) areas seem to have similar shapes at any magnification

sense, if *c* is a polynomial function of z, we arrive back at our original notion of the Julia set: that is, we are simply iterating a polynomial function of z, looking to see which starting points escape under iteration. I tried a slight alteration of this idea, making c a continuous function composed of distinct polynomial regions. Here's the basic algorithm:

```
Define JuliaSetVariation3 (cfunction, zstart)
  n ← 0
  z ← zstart
  while (n < 40) and (magnitude (z) < 2)
    z ← z^2 + cfunction(z)
    n ← n + 1
  color zstart according to the final value of n
```

Figures 6 and 7 show two experiments with this idea. In the first case (Fig. 6), the function by which we choose the new value of *c* is as follows:

```
Cfunction1 (z):
Real part ← −1 * (max 0.6, (min 1.5, magnitude(z) * 0.9))
Imag part ← (max 0.5 (min 1, magnitude(z) * 0.3))
```

The function looks a little complicated, but really all we're doing here is making sure that our resulting z value does not exceed specific bounds. For example, the real part of our new z will never be less than −1.5 or greater than −0.6.

In Fig. 7, the new value of c is chosen as follows:

```
Cfunction2 (z)
Real part ← If magnitude (z) = 0, then return 0.8; else
```

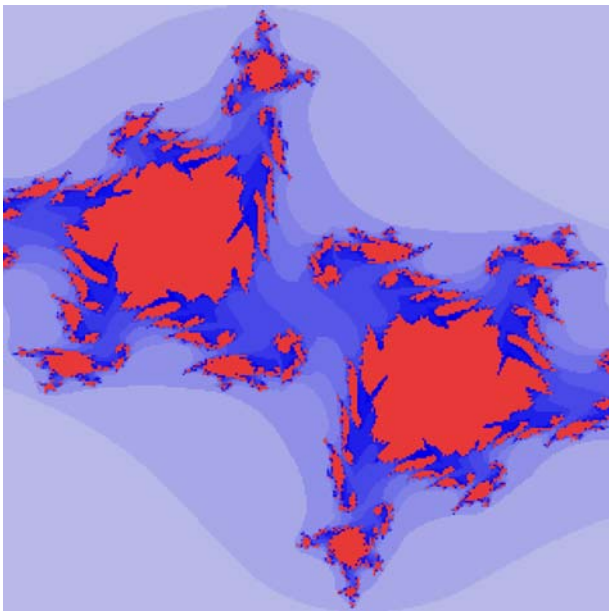

Fig. 6 An escape map where the constant is a function of z

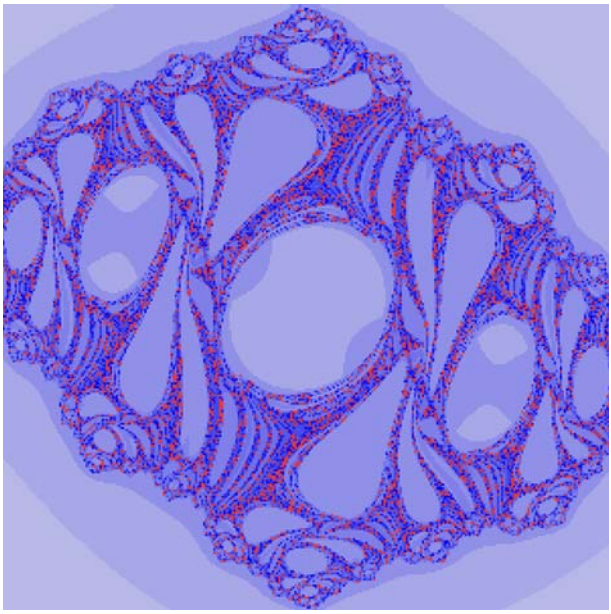

Fig. 7 Another escape map where the constant is a function of z

(−1 * (max 0.2,
(min 0.8, (1/(5 * magnitude(z)))))
Imag part ← If magnitude (z) = 0 then return 0.8; else
(max 0.2, (min 0.8, (1/(5 * (magnitude(z))))))

This is similar to the first function, but the portion of the function within the two bounds varies inversely with the magnitude of z: that is, we tend to choose smaller constants for larger magnitudes of z. The resulting escape map looks almost like an organic form.

These experiments represent a tiny portion of the landscape of Julia variations. Readers are encouraged to experiment for themselves: if you come up with new variations, or interesting pictures resulting from the algorithmic ideas in this column, please by all means send them in! I can be reached at: ijcml-diversions@ccl.northwestern.edu.

Reference

Dewdney, A. K. (1990). *The magic machine*. New York: W. H. Freeman.