A Software Driven Undergraduate Fractal Course

Douglas C. Ravenel
University of Rochester
Rochester, New York 14627
email: drav@harpo.math.rochester.edu

April, 1995

[Submitted to *Fractals, Computer Graphics, and Mathematics Education*, edited by M. Frame and B. Mandelbrot]

The ambiguity in the title is intentional; the phrase 'software driven' refers both to the curriculum of the course and the students taking it. For the past three I years I have taught a sophomore level course at the University of Rochester on the mathematics of fractal images. The classroom I use is equipped with 25 50MHz 486 IBM compatible PCs, and an 8 foot overhead projector. Students taking the course are required to have a year of calculus and some computer experience. The second requirement appears to be redundant, since anyone who would want to take such a course would also be computer literate, in many cases more so than I am. The course appears to have more appeal to computer science majors than math majors. The level of enthusiasm in the students is rare for an undergraduate math course. On many occasions they have surprised me with their energy and originality.

Software used

Before describing the content of the course, I will describe the software used. All of it, with the exception of Mathematica and some programs I wrote, is freely available on the internet. I list them in approximate order of their importance in the course.

- Fractint, an extremely well written omnibus program for creating fractal images quickly, and the workhorse program for the course. (Its existence accounts for the absence of commercial fractal software for MSDOS. Unfortunately, it is *not* available on the Macintosh, but there is an XWindows version.) The program includes code for 32-bit integer arithmetic, which runs quite a bit faster than comparable floating point arithmetic, and gives the program its remarkable speed. It has a comprehensive menu of fractal types, enabling one to reproduce almost every fractal image that has appeared in print anywhere, and a versatile programming language enabling a user with modest skills to create many types not on the menu. Each image produced can be rotated, reflected, and magnified by many orders of magnitude, and one can manipulate the color scheme in various ways.

- Cobweb, Orbit and Curves, three homemade BASIC programs which illustrate the behavior of dynamical systems related to Mandelbrot and Julia sets. In Cobweb, the function $y = f(x) = x^2 + c$ (for a specified real value of c) is plotted for x in a specified interval, along with the diagonal line $y = x$. For a specified x_0 (usually 0), a vertical line is drawn from $(x_0, 0)$ to $(x_0, f(x_0))$, followed by a horizontal line to $(f(x_0), f(x_0))$. This process (vertical line to the curve followed by horizontal line to the diagonal) is repeated a specified number of times, 25 iterations being the default. One can replace the function f by any of its iterates. With this program one can produce graphic illustrations of stable and unstable fixed points, periodic and preperiodic points, bounded and unbounded orbits, and chaos. The Curves program plots several iterates of f simultaneously. The Orbit program plots orbits for complex values of c and x and has similar features.

- Mathematica, which I use for certain symbolic and numerical calculations, such as locating periodic points near a given preperiodic point in the Mandelbrot set, which in turn can be located by solving certain polynomial equations either numerically or symbolically.

- FDesign, a mouse driven program for generating IFS attractors geometrically. One specifies a collection of contracting affine maps (an iterated function system) by drawing a reference triangle and then drawing its images under the various maps. The program displays the corresponding IFS attractor instantly, either in a corner window or on the full screen, with the images of the attractor under the various maps coded by color. The parameters of the affine maps can be exported to Fractint.

- Animation software, for animating a sequence of image produced by Fractint. One needs a program such as DTA to assemble a sequence of `gif` files (`gif` is the graphics format used by Fractint to store images) into an `fli` or `flc` file (`fli` and `flc` are animation formats), which can then be viewed as an animation by a suitable display program.

Organization of the course

I assign two texts for the course, Barnsley [Bar88] and Lauwerier [Lau91], but I do not follow them closely. (Devaney's new book [Dev92] appears to be closer to the mark, but I have not had a chance to use it yet.) I use [Bar88] only in the second half of the course when I talk about iterated function systems. I place most of the books on fractals in our library (our librarian tells me that they are the ones most often stolen) on reserve and occasionally assign readings from them.

I begin the course by showing the video [PJSZ90], which serves as an introduction to both the subject as a whole, and to the first of two major topics in the course, Mandelbrot and Julia sets associated with the function $f(z) = z^2 + c$. Then I introduce the relevant concepts from dynamical systems, using the BASIC programs described above to provide illustrated examples.

The second half of the course is devoted to iterated function systems (IFSs), which is the subject of Barnsley's book. This entails discussing 2×2 matrices and metric spaces, which I do as informally as possible. I stress that the proof of what I call the main theorem of the subject (see below) contains in it the ideas behind the computer algorithms used to depict IFS attractors. Such a direct link between a rigorous proof and an efficient algorithm is rare in mathematics.

The course work consists of four projects, which I encourage students to do in groups of up to three people. I offer a long list of possible topics for each project and encourage the students to invent their own topics. Each project involves some degree of programming, and I have found that they rarely need my help with that aspect of their work. Each is handed in as a printed report with computer generated illustrations, often accompanied by program listings.

Early on I tell them how to use the software provided to make an animated sequence of fractal images; this is explained in more detail below. Everybody is required to make at least one animation as part of a project, and they seem to like doing this. The course ends with a 'fractal film festival' with films made by the students, with prizes offered for the best ones, as determined by class vote.

Mandelbrot and Julia sets

In the following paragraphs, I will say briefly here what I spend several weeks explaining to my students in the class.

For an analytic function such as $f(z) = z^2 + c$, the filled in Julia set K_f is the set of points z with bounded orbits under iteration of f. Computationally one needs an *escape criterion* to recognize an unbounded orbit. For $f(z) = z^2 + c$, one knows that the orbit of z is unbounded if $|z| > 2$. To create an image of K_f, a program such as Fractint must compute the orbit for the value of z corresponding to each pixel on the screen, up to a specified maximum number of iterations, typically 150. As soon as a value with modulus greater than 2 is reached, the pixel is colored according to the escape time, the number of iterations required to meet the escape criterion. If the modulus is still less than 2 after the maximum number of iterations, the program assumes that the pixel is in K_f and colors it accordingly. Fractint has various shortcuts for recognizing in advance when certain points have bounded orbits, thereby saving itself the trouble of computing the full 150 (or more) iterations in many cases.

A theorem of Julia and Fatou (proved before 1920, without the aid of Fractint) says that for polynomial f, the set K_f is connected if and only if it contains the critical points of f, i.e. iff the critical orbits are all bounded. For f as above there is only one critical point, namely $z = 0$. The Mandelbrot set for a one (or more) parameter family of analytic functions (such as the family $z^2 + c$ for varying c) is defined to be the set of parameter values for which each critical orbit of the corresponding function is bounded. The boundedness of an orbit can be determined computationally with the help of an escape criterion as above. In a picture of the Mandelbrot set, the points on the screen correspond to parameter values, and they are colored according to the escape time of the associated critical orbit or orbits.

Let M denote the usual Mandelbrot set, the one for the set of functions $f(z) = z^2 + c$ for varying c. The first thing one sees upon looking at it is a cardioid shaped region around $c = 0$. Simple calculations show that for each c inside this cardioid, the critical orbit converges to a stable fixed point, and for $c = 0$ the orbit itself is fixed. To the left of the cardioid is a circular region of radius 1/4. The critical orbit for its center, $c = -1$, is a cycle of period 2, and for each c inside the circle, the critical orbit converges to such a cycle. Also attached to the main cardioid are infinitely many smaller regions usually called buds, roughly circular in shape. For each value of c in such a region, the critical orbit converges to a cycle of the same period. There is one such region for each rational number between 0 and 1, the denominator of the number being the period of the cycle. One has a precise formula for the point where each bud is attached to the main cardioid, and an approximation for the size of each bud.

Closer inspection reveals many miniature replicas of M known as 'baby Ms.' The largest of these has the cusp of its cardioid at $c = -7/4$, is roughly 1/50th the size of the original M, and the critical point for the center of its cardioid is a 3-cycle. Further investigation shows that these baby Ms occur in clusters around values of c for which the critical orbit is *preperiodic*, i.e. it becomes periodic after some noise at the beginning. Mathematica, used as a numerical tool, can be of great help in locating and analyzing these preperiodic and periodic points in M. A theorem of Lei [Lei90] says that in a small neighborhood of a preperiodic value of c, the Mandelbrot set M exhibits self-similarity with a predictable scaling factor, and that this neighborhood is nearly identical to a similar neighborhood of the point $z = c$ in the Julia set corresponding to c. I will explain below how one can program Fractint to illustrate this result.

The combinatorics of the Mandelbrot set are best encoded in Douady-Hubbard's theory of external angles, which assigns one or more real numbers between 0 and 1 to each point on the boundary of M. The external angles of preperiodic points are always rational. Informal accounts of this theory can be found in Chapter 5 and Douady's paper in [PR86], and in [Dou86]. So far I have not much luck in enticing students to pursue this subject their projects.

One can also consider Julia sets for other functions, and Mandelbrot sets for other families of functions. Here are some interesting examples.

- The cubic function $f(z) = z^3 - 3a^2z + b$, which has critical points at $z = \pm a$. Fractint can be programmed to look at both critical orbits (for given values of a and b) and to end its computation when either orbit escapes. This is a 2-parameter family of functions, so the associated Mandelbrot set is 4-dimensional. One can look at a 2- dimensional slice of it by imposing some relation between the two parameters, such as holding one of them constant. One can make an animation by varying this slice.

- The rational function $f(z) = c(z^n + z^{-n})$ for an integer n. Each $2n$th root of unity ω is a critical point, but each of the $2n$ critical orbits behaves in the same way. Hence the Mandelbrot set (with c as parameter) can be obtained by looking just at the orbit for $z = 1$. It has a garland like appearance with $2n$-fold symmetry, with a copy of the original M at each $c = \omega/2$, and a hierarchy of smaller copies. Each Julia set has a similar appearance.

- For each function $f(z)$, one has the associated Newton function

$$N_f(z) = z - f(z)/f'(z).$$

  Newton's method for finding roots of f consists of iterating N_f; for most initial values of z the orbit converges rapidly to a root. One gets interesting fractals by looking at the set of initial values for which Newton's method fails, which is called the *Newton-Julia set* of f. To display it we color pixels according to *capture time* rather than escape time; we stop computing an orbit of N_f when $|f(z)|$ becomes sufficiently small. For a family of fs one gets a *Newton-Mandelbrot* set by considering the critical orbits of N_f for each f; z is a critical point of N_f if $f''(z) = 0$.

Iterated function systems

IFS attractors in the plane are compact subsets which in many cases can be depicted on a computer screen with far less computing power (both in terms of hardware and software) than is needed for Mandelbrot and Julia sets. Fractint has an IFS option in which it reads data from an `ifs` file and generates the image in a manner I will describe below.

An *iterated function system* (IFS) in $\mathbf{R}^2$ (or more generally any complete metric space X) is a collection $\{F_1, \ldots, F_n\}$ of contraction mappings on $\mathbf{R}^2$. The main theorem of the subject says there is a unique nonempty compact subset $A \subset X$ (called the *attractor*) such that

$$A = F_1(A) \cup \cdots \cup F_n(A).$$

This is proved by considering a new metric space $\mathcal{H}(X)$ whose points are the nonempty compact subsets of X. The (Hausdorf) metric on $\mathcal{H}(X)$ is defined as follows. The distance $d(K, L)$ between compact subsets K and L of X is the smallest number r such that each point in K is within r of some point in L and *vice versa.* If the metric space X is complete, so is $\mathcal{H}(X)$.

Given an IFS on X we define a mapping G of $\mathcal{H}(X)$ to itself by

$$G(K) = F_1(K) \cup \cdots \cup F_n(K).$$

G is a contraction mapping. If each of the F_i shrinks distances by a factor of at most $s < 1$, then so does G. The contraction mapping theorem says that a contraction mapping on a complete metric space has a unique fixed point. Applying it to the contraction G on the complete metric space $\mathcal{H}(X)$ gives the existence and uniqueness of A.

Given a compact $K_0 \subset \mathbf{R}^2$, consider its orbit $\{K_i\}$ under G, defined by $K_{i+1} = G(K_i)$. One sees easily that

$$d(A, K_i) \leq \frac{s^i d(K_0, G(K_0))}{1-s}.$$

If one knows s and $d(K_0, G(K_0))$, one can choose i so that this upper bound is less than the radius of a pixel, regarding the computer screen as a rectangle in the plane. This means that for practical purposes, A is the same as K_i. For a convenient choice of K_0 (such as a single point in the center of the screen), this gives us a finite computation of A known as the *deterministic algorithm.*

The deterministic algorithm is easy to implement but not very fast, since one may have to compute each of the F_i on thousands of pixels in each iteration. A much faster method is the *random algorithm*, which is as follows. Choose a point $P_0 \in K_0$ and obtain P_{i+1} from P_i by applying *one* of the functions F_j, *chosen at random.* Then $P_i \in K_i$, so we know that for sufficiently large i, this point is within a pixel's radius of some point in A, and for such i we plot P_i on the screen. Each iteration requires computing just one function on one pixel, so this algorithm is much faster.

The software (and much of Barnsley's book) deals with the case when the functions F_i are affine, i.e. of the form

$$F_i(x, y) = (a_i x + b_i y + e_i, c_i x + d_i y + f_i)$$

for $6n$ constants a_i through f_i. One also needs to assign a probability p_i to each of these, for the purpose of making the random choice. Experience has shown that it is best to make p_i proportional to the absolute value of the determinant $a_i d_i - b_i c_i$, assigning a minimal positive probability when the determinant vanishes. FDesign does this automatically (after the user defines the $6n$ constants geometrically), but Fractint does not. With this choice of p_i, the points produced by the random algorithm are evenly distributed over A.

It is instructive to see what happens when one does *not* choose the probabilities this way. In version 17.2 of Fractint (which is about 3 years old) one can edit the `ifs` file and see the result immediately. Unfortunately this feature is not present in later versions of the program.

Making fractal animations

In the summer of 1993, Jeffrey Lampert, a student in my first fractals course, hired on as a summer intern for me, and found the software used for animations.

Fractint can save the images it creates as `gif` files. A sequence of such files can be assembled into a `fli` or `flc` file, which is in effect a film, for which there are several display programs available. Fractint also has a batch language (not the same as the formula language described below); it can respond to a series of instructions stored in a `key` file. (The Fractint package comes with a demonstration program which is a DOS batch file telling Fractint to read a file called `demo.key`. One can learn its batch language by examining that file.)

An animation typically consists of 100 or so incrementally varying images produced by Fractint. It is prohibitively tedious to make them all by hand, or to make them with a handwritten `key` file. Instead the students write a program in their favorite language that will generate the desired `key` file for them. If they are animating a sequence of IFS images, it is usually necessary to generate a specially designed `ifs` file as well.

Once produced and saved by Fractint, the images have to be assembled into an animation file. The entire process of image production, saving and assembly can take up to a minute of computing time per frame, depending on the speed of the computer and the resolution (usually 320×200 or 640×480) and complexity of each image.

Some examples of Fractint formulas

Fractint has a formula mode which is in effect a simple programming language. It allows the user to define the function $f(z)$, the initial point of each orbit, and the escape criterion. These typically vary with the pixel, and may also be controlled by two user defined complex parameters `p1` and `p2`. The formulas are stored in a text file (with the extension `frm`) read by Fractint, and the user may change the values of `p1` and `p2` each time a new image is created.

Fractint computes the orbit assigned to each pixel and colors it accordingly. When the program is written one assumes the default screen coordinate system, which has the corners of the screen at $\pm2\pm1.5i$, with `pixel` as a predefined complex variable. Once the image has been produced one has all of the usual Fractint options of zooming, reflecting, recoloring, etc.

Here are descriptions of some `frm` programs that I have found useful.

- *Illustrating Lei's theorem.* [Lei90] (This concerns the similarity between small regions of M and small regions of corresponding Julia set.) One can program Fractint to show different images in different quadrants of the screen in the following way. Define functions characteristic functions $c_1(z)$, $c_2(z)$, $c_3(z)$, and $c_4(z)$, for each the four quadrants. (The `frm` syntax includes the absolute value, real and imaginary parts of a complex variable, so it is possible to make this definition.) Then by defining

$$f(z)=c_1(z)f_1(z)+c_2(z)f_2(z)+c_3(z)f_3(z)+c_4(z)f_4(z)$$

  one is effectively looking at four different functions in the four quadrants of the screen. Using the user defined parameter `p1` for c, one can define f and the initial point of each orbit in such a way that one sees in the four screen quadrants

  - a neighborhood of c in M magnified 5 times,
  - the same neighborhood magnified 100 times,
  - the Julia set for c, and
  - a neighborhood of c in the Julia set magnified 100 times.

- *Illustrating Feigenbaum points.* (A Feigenbaum point in M is a limit point of a converging sequence of periodic points in which the period increases exponentially, rather then linearly as in the case of a preperiodic point.) This is similar to the previous example, but we show four successive magnifications of M around the point $c =$ `p1`, with the magnifications being powers of the number $\delta =$ `p2`. For complex values of δ, this 'magnification' includes a rotation. The best known example of a Feigenbaum point is $c = -1.401155\ldots$ (the limit point of the period doubling scenario) with $\delta = 4.669201\ldots$, the Feigenbaum number. Some other examples are listed in [Mil89]. One can also use this program to illustrate self-similarity around preperiodic points, the simplest example being $c = -2$ with $\delta = 4$.

- *The Curry-Garnett-Sullivan experiment.* [CGS83] The authors study the Newton-Mandelbrot and Newton-Julia sets for the cubic function

$$f(z) = z^3 + (c-1)z - c.$$

  The Newton function N_f has four critical points, three of which are roots of f. The orbit of the fourth critical point, $z = 0$ (where $f''(z) = 0$), may or may not converge to a root. The values of c for which it fails to do so comprise the Newton-Mandelbrot set. The papers shows a small copy of the original M in this set, centered at $c = .31 + 1.62i$, for which the critical orbit converges to a cycle of period 2. To see this Newton-Mandelbrot set with Fractint, set $c =$ `pixel`, and iterate the function N_f starting at $z = 0$ and stopping when $|f(z)|$ gets small, say less than 10^{-6}. For a Newton-Julia set, set $c =$ `p1`, $z =$ `pixel` and iterate as before.

Conclusion

My experience with this course has convinced me that the study of fractals is a great way to draw students into mathematics. The image of a fractal on a computer screen has a fascination for both the specialist and the casual observer. Anyone curious about its properties and how it is produced quickly finds herself in deep mathematical waters. There is readily available software that makes it easy to illustrate lectures on fractals in real time. This subject is a motivational tool that the mathematical community should make the most of.

References

[Bar88] M. F. Barnsley. *Fractals Everywhere.* Academic Press, Boston, 1988.

[CGS83] J. H. Curry, L. Garnett, and D. Sullivan. On the iteration of a rational function: computer experiments with Newton's method. *Communications in Mathematical Physics*, 91:267–277, 1983.

[Dev92] R. L. Devaney. *A first course in chaotic dynamical systems : theory and experiment.* Addison-Wesley, 1992.

[Dou86] A. Douady. Algorithms for computing angles in the Mandelbrot set. In M. F. Barnsley and S. G. Demko, editors, *Chaotic Dynamics and Fractals*, pages 155–168, Academic Press, 1986.

[Lau91] H. Lauwerier. *Fractals: Endlessly Repeating Geometrical Figures.* Princeton University Press, Princeton, 1991.

[Lei90] T. Lei. Similarity between the Mandelbrot set and Julia sets. *Communications in Mathematical Physics*, 134:567–617, 1990.

[Mil89] J. W. Milnor. Self-similarity and hairiness in the Mandelbrot set. In M. C. Tangora, editor, *Computers in Geometry and Topology*, pages 211–257, 1989.

[PJSZ90] H.-O. Peitgen, H. Jurgens, D. Saupe, and C. Zahlten. *Fractals, an animated discussion [videorecording], interviews with E. Lorenz and B.B. Mandelbrot.* W.H. Freeman, New York, 1990.

[PR86] H. O. Peitgen and P. H. Richter. *The Beauty of Fractals.* Springer-Verlag, New York, 1986.