COLORING DYNAMICAL SYSTEMS IN THE COMPLEX PLANE

Francisco Garcia, Angel Fernandez, Javier Barrallo, Luis Martin

The University of the Basque Country
Plaza de Oñati, 2.
20009 San Sebastián. SPAIN

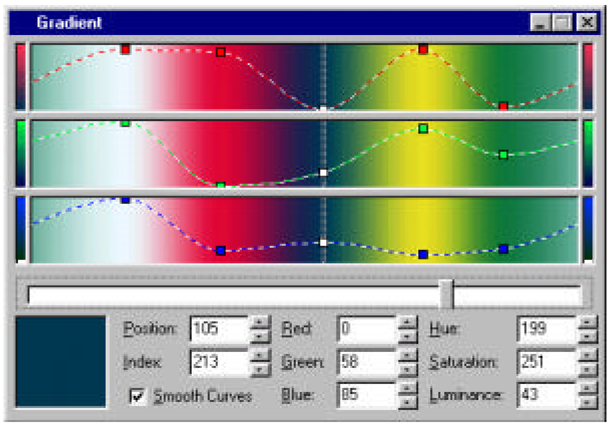
Introduction

Dynamical systems are a well-known branch of mathematics, but until the arrival of computers the sheer number of calculations involved made them impractical for real use. The computer's ability to perform rapid calculations allows us to condense billions of calculations into results we can digest. Benoit Mandelbrot was the first one to use computers to produce graphical representations of dynamical systems in the complex plane, based on the quadratic formulas described by the French mathematician Gaston Julia at the beginning of the 20th century.
During the 1980s, fractal enthusiasts began exploring fractals for their *artistic* merit, not for their mathematical significance. While mathematics was the tool, the focus was art. As the fractal equation itself was the most obvious mathematical element, fractal artists experimented with new equations, introducing hundreds of different fractal types. By carefully choosing parameters to refine form, color, and location, these explorers introduced the concept of *fractal art*.
After 1995, few new major fractal types have been introduced. This is because the newest innovations in fractal art do not come from changing the fractal equation, but from new ways of coloring the results of those equations. As these coloring algorithms move from simple to complex, fractal artists are often returning to the simpler, classical fractal equations. With the increased flexibility these sophisticated algorithms provide, there is even more room for personal artistic expression.

Coloring Algorithms

Every dynamical system produces a sequence of values z_0, z_1, z_2… z_n. Fractal images are created by producing one of these sequences for each pixel in the image; the coloring algorithm is what interprets this sequence to produce a final color.
Typically, the coloring algorithm produces a single value for each pixel. Since color is a three-dimensional space, this one-dimensional value must be expanded to produce a color image. The common method is to create a *palette*, a sequence of 3D color values; these are connected end-to-end and the coloring algorithm value is then used as a position along this multi-segmented line (the *gradient*). If the last palette color is connected to the first, a closed, segmented loop is formed and any real value from the coloring algorithm can be mapped to a defined color in the gradient. This is similar to the pseudo-color renderings often used for infrared imaging. Gradients are normally linearly interpolated in RGB space (Red, Green, Blue), but they can also be interpolated in HSL space (Hue, Saturation, Lightness) and interpolated with spline curves instead of straight line segments.

The selection of the gradient is one of the most critical artistic choices in creating a good fractal image. Color selection can emphasize one part of a fractal image while de-emphasizing others. In extreme cases, two images with the same fractal parameters, but different color schemes will appear totally different.
Some coloring algorithms produce discrete values, while some produce continuous values. Discrete values will produce visible stepping when used as a gradient position; until recently this was not terribly important, as the restriction of 8-bit color displays introduced an element of color stepping in gradients anyway, and discrete coloring values were mapped to corresponding discrete color in the gradient. With the introduction of inexpensive 24-bit displays, algorithms which produce continuous values are becoming more important, as this permits interpolating along the color gradient to any color precision desired.
With the increasing significance of coloring algorithms it is surprising that there has been no attempt to classify the variety of algorithms that are popping up. The following sections of this paper may be considered as an initial classification system for the most important algorithms.


Gradient

Position: 105
Index: 213
[✓ Smooth Curves]
Red: 0
Green: 58
Blue: 85
Hue: 199
Saturation: 251
Luminance: 43

Fig 1. Gradient window linearly interpolated in RGB space with spline curves.

I. The Escape-Time Algorithm

The *escape-time algorithm* is one of the earliest coloring algorithms, and in many programs it is still the only option available. Its simplicity makes it a favorite with those learning to create fractal software. From an artistic viewpoint it is becoming less important, because it produces discrete values; continuous coloring algorithms have generally superceded it.
The algorithm itself is based on the number of iterations necessary to determine whether the orbit sequence tends to infinity or not. It can be strictly demonstrated that when the orbit of any value of z_0, z_1, z_2… z_n exceeds a border region *R*, it always diverges towards infinity. The minimum size and shape of *R* are different for each fractal type, of course. If the orbit sequence is stopped as soon as z_n is outside the border region *R*, then the coloring value for the escape-time algorithm is simply the length of the sequence, that is, n. (This value is readily available in actual implementations of fractal software, because a limit must be placed on iterations in order to prevent infinite iterating of a single orbit.)
Traditionally, *R* is set as a circle, centered at the origin, with radius 2. This is because for the Mandelbrot set, it can be proven that as soon as $|z| > 2$, the orbit will diverge. Interesting variations have been created by changing the shape and position of *R*; ellipses, triangles, stars, and so on may be used instead. And while mathematically, *R* must include the circle of radius 2 to be accurate in testing for divergence, this has not stopped some artists from experimenting with smaller radii.

II. Distance Estimators

The escape-time algorithm can be considered as a measurement of the (non-Euclidean) distance from any point z_0 to the border of the set. Its use of a discrete value (the number of iterations, always an integer) produces a *banding* effect similar to the contour lines of topographic survey maps. Creative use of gradients can actually take advantage of this effect (so-called “tiger striping”) but a large number of artists have explored algorithms to hide this effect. Clearly, the goal has been to develop continuous functions for this distance measurement. None of the algorithms here pretend to give accurate Euclidean distances, but generally they provide acceptable continuous values.

The first historical approach to continuous color values was the *distance estimation algorithm*. It represents the distance of every point to the fractal set, calculated with the formula [1]

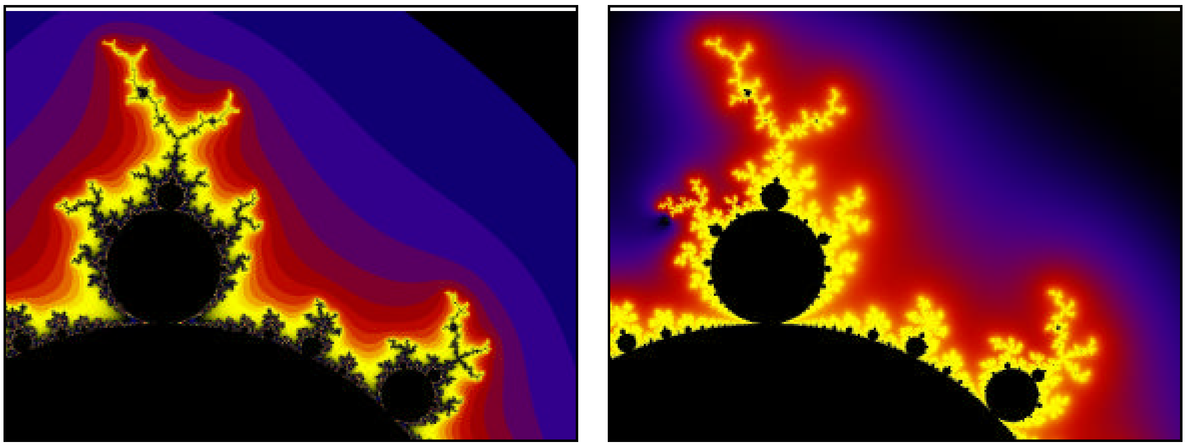
$$\frac{2|z_n|\log|z_n|}{|z_n'|} \qquad [1]$$

This value is closer to the Euclidean distance. Its isocontours are shaped differently than the visible bands from the escape-time algorithm.

Another variation is the *continuous potential algorithm*. If the fractal shape in the complex plane were extended infinitely above and below the plane as a fractal-shaped “cylinder”, and treated as a wire that produces an electrical field, then the *electrostatic potential* for any point z_0 is approximated by [2]

$$\frac{\log|z_n|}{2^n} \qquad [2]$$

provided enough iterations (n) are done that $|z_n|$ is large. Most fractal artists do not care about electrostatic potential, but they are very interested in continuous values.



Figs. 2a, 2b. Discrete (Escape Time) and continuous (Distance Estimation) color algorithms. Note the banding effect in 2a compared with the continuous coloring of 2b.

Of particular interest to fractal artists who have many images created with the escape-time algorithm is the *normalized iteration count algorithm*. The primary advantage of this algorithm is that it produces isocontours which are the same as the escape-time algorithm, if given the correct parameters, yet it produces continuous values rather than discrete ones. This direct adaptation can remove the banding from older escape-time images while preserving the rest of the fractal shape. The formula for calculating the normalized iteration count is

$$n+1-\frac{\log\left(\log|z_n|\right)}{\log 2} \quad [3]$$

An interesting quirk of this formula is that it does not depend on the size of the bailout circle used; for matching up normalized iteration counts with the simpler discrete iteration counts, it is more convenient to use the formula [4]:

$$n+\frac{\log(\log b)-\log\left(\log|z_n|\right)}{\log 2} \quad [4]$$

where b is the bailout radius.

As a final distance estimator, we introduce $e^{-|z|}$ *smoothing*. This simply sums $e^{-|z|}$ over all iterations; as $|z|$ increases, $e^{-|z|}$ approaches zero and further iterations change the sum very little. Like normalized iteration counts, as long as the bailout radius is sufficiently large, the precise value bears little impact on the resulting image.

It should be noted that the formulas given above generally apply to the classic z^2+c formula. For similar equations where only the exponent is changed, replacing the 2 with the correct exponent is often sufficient. For other fractal equations, more radical reworking may be required. For $e^{-|z|}$ smoothing, however, no changes are necessary; it works well for almost any divergent series.

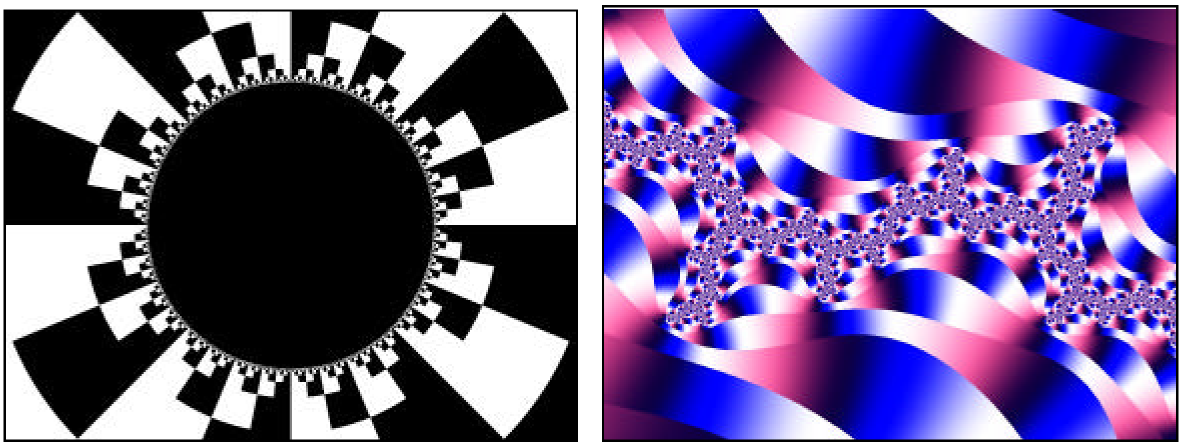
III. Escape Angle

The formulas discussed so far have only considered the magnitude of z and the iteration count. If we think of the magnitude of z as one part of the polar coordinates of z, then it becomes natural to consider the other part—the angle of z—for coloring. The *escape angle* family of algorithms covers those algorithms based on the angle of z_n.

The first algorithm here is *binary decomposition*. With this algorithm, z_n with angles above the real axis are given one color, and z_n with angles below the real axis are given another. This is of course a fundamentally discrete algorithm, but it provides a visual approximation to the binary form of the field lines surrounding the fractal. While this can provide some valuable insights into the structure of fractals, it is not so often used in artistic fractal images.

Variations on the binary decomposition scheme increase the number of divisions of the plane. For example, quaternary decomposition would assign a separate color to each quadrant, with z_n angles in each quadrant producing the corresponding color. Increasing the number of divisions allows more colors to be used.

Of course, these are all discrete methods; with *continuous decomposition,* the angle is read directly and used as a continuous position along the color gradient.



Figs. 3a, 3b. Binary decomposition and continuous decomposition of two different Julia sets.

IV. Curvature Estimation

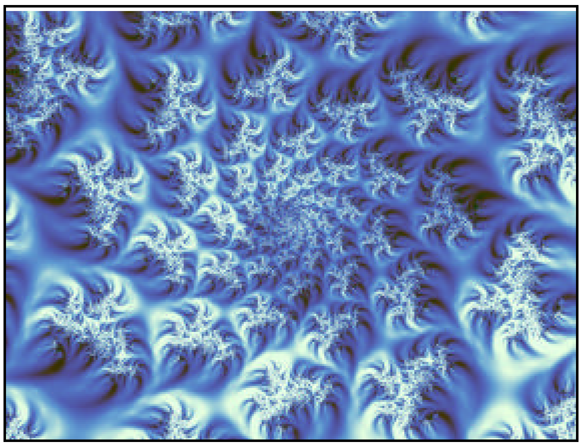
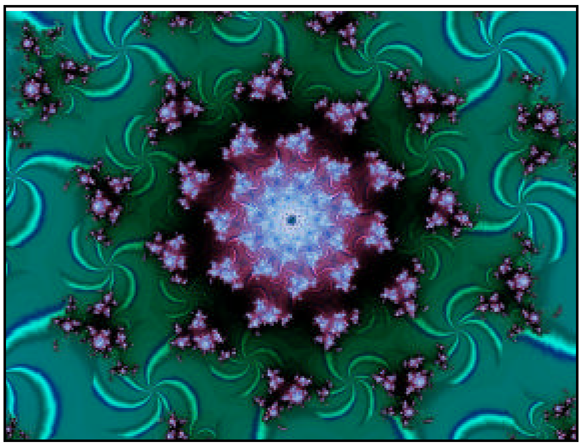
Another aspect of the sequence z_0, z_1, z_2… z_n that can be measured is the curvature between consecutive iterations. A quick estimate can be obtained by using only the last two orbit points. A more thorough measure can be obtained by averaging [5] over all iterations.

$$\left| tan^{-1}\left[\frac{z_n - z_{n-1}}{z_{n-1} - z_{n-2}}\right]\right| \qquad [5]$$

A variation on this method determines the radius of a circle passing through three consecutive orbit points. This factors in not only the angle between consecutive points, but the distance between them as well.

V. Statistics

Iterating any value z in the complex place produces a sequence of values z_0, z_1, z_2… z_n, and the magnitude of each term can be treated statistically. Mean, range, standard deviation, and other common statistical values can be calculated, and the values used directly for coloring.

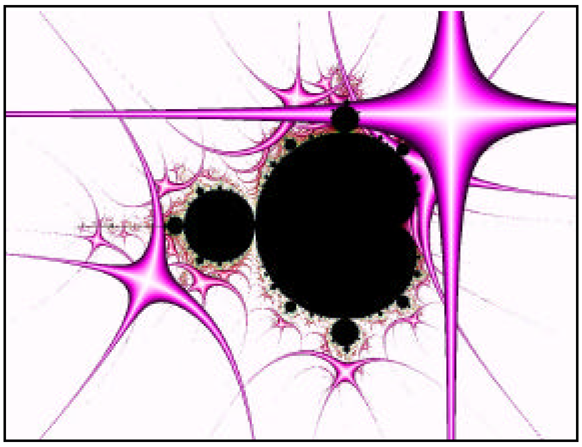
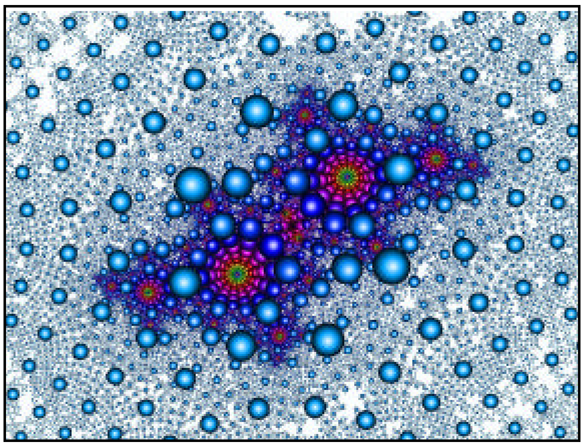
Figs. 4a and 4b. Both images show the same formula and region of the complex plane, but different coloring algorithms. 4a uses a curvature estimation and 4b is the result of the application of several statistical methods to the orbit of z_n.

Other more uncommon statistical methods, like the fractal dimension and Lyapunov exponent, can also be used, although these work best with bounded data sets. Sometimes it is possible to transform the unbounded magnitude of z_n into a bounded data set by inverting it ($1/z_n$).

VI. Orbit Traps

This is one of the largest families of coloring algorithms, because it provides so many options for experimentation. In fact, entire software packages have been built specifically to explore this family of coloring algorithms. The basic idea is to choose a region of the complex plane (denoted by T) and watch the relationship between the z_n values and T. T is usually defined as a central shape (usually a simply-calculated item, such as a point, line, or circle) and a threshold distance. Everything within the threshold distance of the trap is considered “inside” the trap.

The earliest implementations of orbit trap algorithms simply watched for any z_n that fell inside the trap, and at the first such value, stopped iterating and colored based on the distance to the central trap shape. (This is where the term “trap” comes from; once the orbit fell into T it was “trapped” and iteration stopped.)

Figs 5a and 5b. Hypercross orbit trap in the Mandelbrot set (left). Gaussian integers algorithm in a Julia set (right).

There are many more variations, however. The first class of variations covers the shape of the trap region, T. With the basic line and circle trap shapes, points of equal distance from the trap are the same shape as the trap, e.g. points equidistant from a line form a line. By using non-Euclidean distance measures, contours of equal distance can be transformed into other shapes; points (which with Euclidean distances would generate circular contours) can generate squares, astroids, and hyperbolas. Other interesting trap shapes can be created by using complex curves as the central trap shape, such as spirals and flowers. Furthermore, these trap shapes can be distorted by rotation, skewing, and stretching, often incrementally with each iteration.

Another class of variations deals with the relationship between the distances of each z_n to T. The classical implementation mentioned above stopped at the first z_n within the threshold distance to T. Other variations use the *last* z_n to enter the trap, or the closest, or the farthest that is still inside the trap. More exotic variations use different methods for combining all distances below the threshold together.

The last major class of variations deals with the actual value used to produce the color. The most common method is simply the distance to the trap shape T, be it closest, farthest, first, etc. Other choices include the magnitude or angle of z_n, or some combination of z_n values related to the trap distances.

With so many variations possible, and so many combinations of variations, it is nearly impossible to predict exactly what results will be achieved. Sometimes it is even difficult to tell a particular image has used the orbit trap algorithm at all. This is one reason this family of algorithms is so vastly popular.

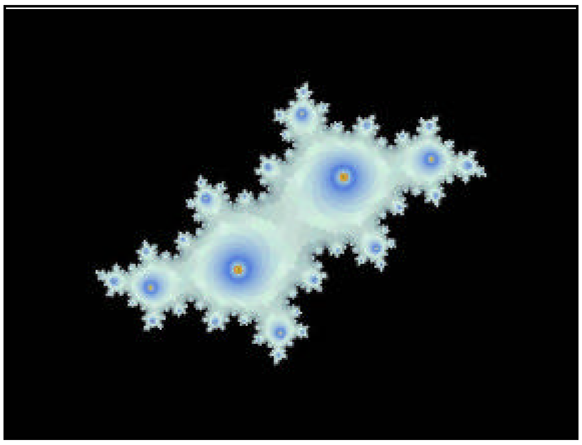
VII. Gaussian Integers Algorithm

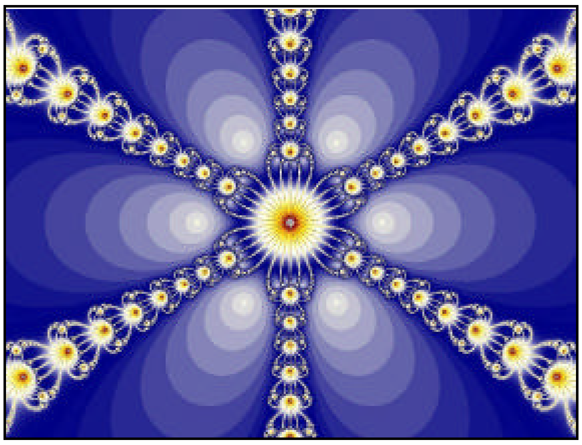
A Gaussian integer is a complex number whose real and imaginary components are both integers. The algorithm works by computing the distance from each z_n to the nearest Gaussian integer, and then coloring based on the smallest such distance for all orbit values.

Conceptually, this is like using an orbit trap where the trap shape T (a point) is repeated over the complex plane in a regular grid, coinciding with the Gaussian integers. Once perceived in this manner, it is clear the technique can be extended to any other trap shape, with different grid spacing, and even non-rectangular grids. Radial grids and triangular grids are just two possibilities.

VIII. Finite Attractors

Not all orbit sequences z_0, z_1, z_2… z_n tend towards infinity. (For some fractals, very few do!) Orbits that do not tend to infinity most often converge to a single point or a periodic cycle. While many of the techniques discussed so far can be applied directly to such convergent sequences, some require a little tweaking. Still, it is often useful to search for finite attractors directly. The simplest method for doing this is to look for a decreasing change in z. As z_n converges on a fixed point, $|z_n-z_{n-1}|$ tends towards zero. Once this difference drops below some threshold, we consider the point sufficiently converged to the finite attractor and we color accordingly. (We could color based on the iteration count or any other algorithm we want.) Using the magnitude of the difference between consecutive iterations produces circular iteration bands, but just as changing the shape of the bailout region R for divergent sequences changes the shape of discrete iteration bands, so can a similar change can be used for convergent sequences.
This technique works well for sequences which converge to a single point, but some adaptations must be made to catch points which converge to a periodic cycle. To do this, one iterates to a fixed number of iterations, and then the entire sequence is examined to find the first z_k which is within some threshold distance of the final z_n. In this way, no matter which point in the periodic cycle z_n settles on at the predetermined iteration limit, at least one value along the way almost certainly has approached within a small distance of that point as the sequence converges on the periodic cycle.





Figs. 6a, 6b. Figure 6a shows the attractors of a connected Julia set z^2+c. Figure 6b shows the attractors (roots) from the equation $z^6-1=0$ iterated under the Newton-Raphson Method [$z_{n+1} = z_n - f(z_n) / f'(z_n)$].

IX. 3D Effects

Although fractal images are typically created with a two-dimensional view, it is possible to create an image with a three-dimensional appearance through programming effects. Essentially, several points very close together (closer than one screen pixel) are iterated in parallel; once iteration is completed, a “height” value for each point is computed, a plane passed through the three points (now oriented in 3D space), and a vector perpendicular to the plane calculated. This gives us a simulated “surface direction” in the vicinity of these points. By measuring the angle between this surface normal vector and a light vector L, we can compute the amount of light falling on that surface and light it realistically.

If the light vector L is fixed at the real or imaginary axis, the calculations can be simplified substantially and only two points need be iterated in parallel. When the cost of iterating is high, this can be an attractive optimization.
Any of the distance estimator functions provides an excellent height value to use, but the technique works fairly well with almost any function. Interesting "embossing" effects can be achieved by using a discrete value for height; this produces a terraced appearance.

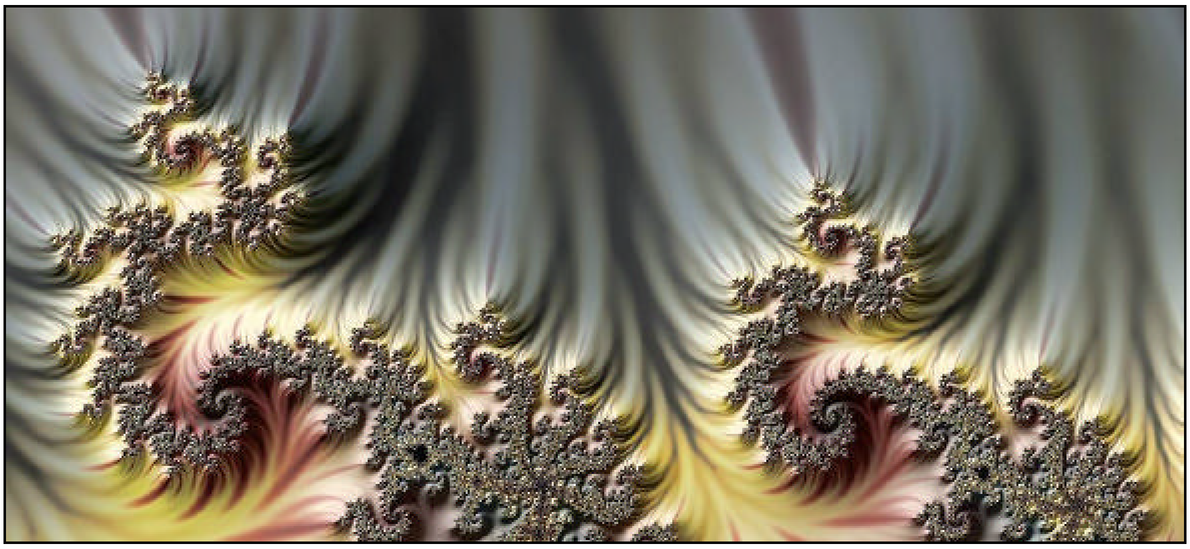


Fig 7. The Mandelbrot set surface with a three-dimensional terraced appearance.

X. Other Variations

There are many fractal coloring algorithms which we have not covered here, because they are either not common in fractal explorations or because they are specializations of the general forms we have already presented. There are also many ways to vary the techniques here to produce new algorithms. For example, instead of using $|z_n|$ for statistical computations, we might use only the real or imaginary component, or the angle of z_n. We might use the value of z_n relative to some other point (perhaps z_0). We cannot describe them all!
A more significant technique involves *combining* techniques. We call these *multi-layer* fractals, and they are the source of some of the richest fractal imagery being produced today. In this way, even a limited collection of fractal coloring algorithms can be combined in almost endless ways, with each combination becoming in effect an entirely new algorithm.

Bibliography

- Barrallo, J. ***Fractal Geometry***. 2nd Edition. ED. Anaya Multimedia. Madrid 1992. ISBN 84-7614-452-0
- Jones, D. Fractalus web page. ***http://www.fractalus.com***