Project Number: 48-MH-0223

FRACTALS AND ART

An Interactive Qualifying Project Report

Submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

in partial fulfillment of the requirements for the

Degree of Bachelor of Science

by

Steven J. Conte

and

John F. Waymouth IV

Date: April 29, 2003

Approved:

1. Fractals
2. Art
3. Computer Art

Professor Mayer Humi, Advisor

Abstract

Fractals are mathematically defined objects with self-similar detail on every level of magnification. In the past, scientists have studied their mathematical properties by rendering them on computers. Recently, artists have discovered the possibilities of rendering fractals in beautiful ways in order to create works of art. This project provides some background on several kinds of fractals, explains how they are rendered, and shows that fractal art is a newly emerging artistic paradigm.

Contents

1 Executive Summary

This Interactive Qualifying Project explores the overlap between the subjects of fractals and art. We sought to find out how mathematics could be involved in the process of creating art and to explore the humanistic side of fractals. To do this, we explore several different kinds of fractals, explain their mathematical definition, and show how they could be used to generate aesthetically beautiful images. Then we explore the sociological side of fractal art, and we analyze the psychology behind the aesthetic beauty found in several pieces of fractal art.

This report begins with a brief introduction to the topic of fractals and then addresses the question of whether fractals can be considered a form of art. At this point, we establish the goal for this project: to prove that fractal art is a new form of art. Proceeding from this point, we give a general mathematical background of some of the mathematical details of several kinds of fractals, including the Mandelbrot and Julia sets, iterated function systems, and L-Systems. Then we describe the concept of fractal dimension, and demonstrate how to calculate the dimension of several example fractals. Using this infromation, we explore how the concept of fractal dimension can be applied to works of art.

From this point the report delves into more humanistic issues. It is interesting to generate classic fractals such as the Koch curve and Sierpinski gasket, but how can we use fractals to create truly beautiful art? We show how fractals can be used to generate music, and we discuss the artistic value of such music. Next, we explore the relationship between art and society. Theories of how society is reflected in art are discussed, and we try to answer the question,"What does fractal art say about our society?" Finally, we apply existing psychological studies about the perception of art to a few pieces of work by fractal artists, to show that many of the same qualities found in traditional art can also be found in fractal art.

2 Introduction

The field of mathematics has seen many recent discoveries related to fractals. Fractals hold many secrets that mathematicians are only beginning to unlock. From the intricate and colorful Mandelbrot set to the infinitely repeating pattern of the Koch curve, fractals hold the interest of many. Recent discoveries have found fractals and similar formations in many areas of research such as physics and geometry. Other links can be found between fractals and natural formations such as mountain ranges. Fractals can even be found in works of art, where a painting has more detail visible when you examine its edges arbitrarily closely.

This Interactive Qualifying Project (IQP) focuses on the artistic possibilities of fractals. It is the goal of this IQP to explore the applications of the growing field of fractals to the arts. We will give a general introduction to fractals, and then show how these ideas relate to various areas of art. In this way, we intend to demonstrate that, along with having intriguing scientific ramifications, fractals have a very important humanistic side. This, we feel, relates science and sociology, which is the purpose of an IQP.

The authors hope to broaden their own knowledge of fractals as well. This project provides two interesting opportunities: we not only will explore what mathematically defines a fractal and what mathematical properties they have, but we will also see how fractal images can be considered works of art.

Science thrives on the benefits it gives to average people. There would be no purpose to scientific discovery if it did not bring improvement to the lives of many people. Medicine is an obvious example: medical discoveries lead directly to improving the health of patients. Fractals can also provide a benefit in that even purely mathematical fractal images such as the Mandelbrot and Julia sets are strikingly beautiful.

The authors hope to broaden the impact of their existing scientific knowledge by seeing how it can positively affect peoples' lives. Instead of studying only the pure mathematics of fractals, we will forge a link between mathematics and beauty. It is our hope that we can gain some experience with what beauty itself is. In this way, our future scientific efforts will be improved because we will be more attuned to how our efforts will be viewed by an average person. And as was outlined above, this is arguably a very important consideration when exploring science.

One can find many examples of beauty in fractals. It would not be effective to try to cover all of the many types of fractals at once, and it is beyond the scope of an IQP. However, we feel that we can gain some of the insight described above by focusing on a few links between fractals and art. By exploring some examples, we hope to show how fractals and art are deeply related as a whole.

2.1 Fractals

The term "fractal" encompasses a somewhat loosely defined type of mathematical object. Traditionally, fractals are defined as objects exhibiting self-similarity; that is, objects containing parts that look like the whole when magnified. Objects like the Koch Curve and Iterated Function Systems, which will be explored later, are self-similar objects. When using this definition of "fractal", it is implied that these objects must have new detail on every scale of magnification. In fact, fractals are commonly described using just this last criterion: that they have fine detail of a similar structure at every scale of magnification. This is the definition used for this IQP, which allows us to explore such objects as the Mandelbrot and Julia sets, which do not exhibit true self-similarity.

The fine detail common to fractal images is what helps to make them beautiful and fascinating to behold. Fractal images tend to be wildly chaotic, with new and interesting formations that draw the eye. They usually have vivid color and often exhibit self-similarity, or at least some general form of self-mimicry, at many levels. The eye is often drawn to explore a tiny detail, which, on closer inspection, is very similar to the image as a whole. This plays on the natural human instinct to correlate and explore similar visual stimuli.

It is more interesting, for artistic purposes, to concentrate on the representation of fractals in a computer graphics environment. Fractals themselves are infinitely detailed mathematical constructs, so it is not possible for humans to visually and aesthetically experience a fractal object. It is the rendering process that brings some sense of order out of chaos: it gives us a way to visualize a fractal. It is important to understand that the images produced by processes described in this IQP are not the fractals themselves; they are just a graphical approximation of the fractal object.

As we will demonstrate, the rendering process plays just as important a role in creating an aesthetically pleasing image as the fractal formula itself. To represent a fractal graphically, it is necessary to make many choices, such as color, size of image, parameters, rendering effects, and many other possibilities. Choosing a good rendering method can produce an image of startling depth and beauty, while choosing a poor rendering method can create an image that seems like a random collection of differently colored dots. This project will demonstrate several popular fractal rendering methods and explore their aesthetic appeal.

3 Are Fractals Art?

Before we explore different kinds of fractal art, it is important to consider this question: can fractals, in fact, be considered a form of art? Certainly one can see, by flipping through the diagrams later in this report, that there are plenty of examples of intriguing and beautiful fractals. But the question remains, can we place our work in the same field as da Vinci and Michaelangelo? Can the process of manipulating mathematical formulas be likened to a process of artistic creation?

At the outset, some people will argue that the answer is no. They claim that, because fractals have mathematics as their basis, there is no chance for a person to introduce meaning and intrigue. Mathematics is, after all, a dry field of unwavering truths and equations; numbers don't express emotion. Others might point out that fractal images exist only as the creation of a computer. Computers follow specific instructions and cannot think for themselves or experience emotion. Or perhaps opponents will point out the fact that the fractal image was "always there", just waiting for the mathematician to stumble across it. Finally, perhaps conceding that there are infinite possible images to be found in a given fractal, opponents have argued that the "artist" is merely punching in random numbers to create his images.

The purpose of this project is to show, by example and by discussion, that fractal imagery is an art form in and of itself. The number of possible images in each fractal is infinite, and not every one of these is even remotely aesthetically pleasing. But some fractal images are strikingly beautiful. They have infinite detail that can always yield new shapes at closer inspection. Their self-similarity tantalizes our natural need to make associations between concepts in our minds. They can be intriguing, and undeniably beautiful. . . but are they art?

What *is* art, anyway? Authors have written long philosophical volumes attempting to tackle that question. Indeed, the question is beyond the scope of this project. We will have to settle with a working definition, and use this to give the reader a sense that there is art in fractals.

We know that certain works by the great artists strike us as beautiful. For some reason, a picture will intrigue or fascinate us. Other works communicate emotions to us, or try to make a point. Some works, like the drawings of M. C. Escher, try to catch our interest with a visual riddle. All of these qualities can also be found in fractal art.

But what about the idea of artistic process? Fractals are static images, generated predictably from a specific set of input numbers. How do we get past the arguments above? To see how, we must examine the procedure involved in creating fractal images. There is a process of trial and error, of experimentation, of making small refinements to make a more interesting picture. The mathematics may stay rigid, but the pictures don't create themselves. A fractal image is only a visual realization of the underlying mathematical process. It is by varying the method of visualization that the artist can express himself.

Is photography a form of art? After all, one is only reproducing what is already around us. How can we call photography art, if the photographer is merely copying what we can already see? But photography is already a well accepted form of art. The photographs of Ansel Adams are very popular, and many art institutes have entire departments devoted to photography. Photography is art, but why?

Pictures don't take themselves, and they don't develop themselves. A photographer chooses what to shoot, what lighting to use, what shutter speed, what kind of film, where to focus, and so on. Then he chooses how long to expose the film, what kind of developing techniques to use, how long to develop the film, how large to make the print, and many other factors, some of which are unique to each photographer. Here lies the heart of the matter: photography is a form of art because the photographer uses his own taste and aesthetic sense in choosing how to make his picture. We may not be able to define beauty and art, but we can sense it in our creations. We can learn from experience how each factor affects the end result, and therefore, we can express ourselves through our art.

Fractal art is very similar. A fractal artist must choose what kind of fractal to use, where to zoom in, what iteration algorithm to use, what kind of coloring to use, how large an image to use, how much detail to capture, and other features. Some fractal artists even take their raw fractal images and edit them in a graphics editor, which allows for further expression. A fractal artist is not pressing random buttons and seeing what comes out, he is experimenting with many kinds of variation, and deciding if he likes the end result. In this way, fractal art is much like photography, and, indeed, like any other kind of art.

Another reason why fractals hold artistic appeal is that they are closely related to the manner in which the world around us works. As we will discuss later in this paper, fractal geometry can describe many natural formations much more accurately than traditional geometry, which is based on straight lines and geometric shapes. Fractal functions have been crafted to look like land masses and mountain ranges, and such images have been successfully used in movies.

Fractal images can strike us on an instinctual level because they resemble the way things work in the real world. As a child, John remembers watching with fascination a short segment about fractals in a children's show. The imaginary camera kept plumbing the depths of the Mandelbrot set (described in the next section), demonstrating the concept of self-similarity. What was once a tiny detail in the entire shape turns out be a copy of the whole shape in miniature. And the tiny details in that shape, in turn, are copies of the whole shape again. No matter where the camera explored, more detail could be found, always with the same recurring themes.

Now imagine looking at a greatly detailed picture of a cloud. Under a microscope, the picture would demonstrate that, what looked like a small offshoot of the main cloud is, in fact, reminiscent of a cloud itself. And a small offshoot of this sub-cloud looks like a cloud too. Now consider a head of cauliflower. Look at the whole thing, and then break off a piece of it. Examine this piece, and notice that it remarkably resembles a whole cauliflower plant. Break off one of these pieces, and it, too, resembles its own plant. The similarity stops after about three levels, but it is still intriguing. Fractals exhibit this self-similarity down to infinity, while nature exhibits it on a limited, but still significant, scale.

Coloring techniques also help add to the beauty of a fractal. High contrast can highlight certain details of a fractal and can help self-similarity to stand out. We can mimic the vibrancy of nature in our fractals, and we can use this as a tool to heighten a fractal's similarity to a natural form. Smooth gradients help guide the eye from one part of a fractal to another. Bright, wildly changing colors catch the eye and encourage exploration.

Fractals provide a new medium for artistic expression. Fractal artists learn the skills necessary to express themselves through fractals, and, in this way, they have opened a new field of art. While we, the authors, are not artists, we hope to demonstrate our own explorations in this growing field and give the reader an idea of the nature of this new art form.

4 The Mandelbrot and Julia Sets

4.1 The Mandelbrot Set

The Mandelbrot set, first visualized by Benoit Mandelbrot in 1977, is probably the most commonly known fractal. Images of it can be found, among other places, in science exhibits and on children's shows such as 3-2-1 Contact. The algorithm is simple and yet the visual effect it creates can be stunning. It exhibits a certain degree of self-likeness, although small differences can be found, preventing the Mandelbrot set from being completely self-similar in a strict sense.

4.1.1 Definition

The Mandelbrot set is fairly simple to generate, and is a common choice for programmers interested in learning about fractals. It involves characterizing the behavior of the following recurrence relation which operates on complex numbers z_n:

$$z_{n+1} = z_n^2 + c \tag{1}$$

In this equation, the initial value z_0 is a parameter. In order to generate the classical Mandelbrot set, we choose $z_0 = 0$. Given this beginning value, and a value for c, one can generate z_1. Then, using this value of z_1, one can generate z_2, and so on. Each new calculation is called an iteration. The behavior of z_n for increasing values of n is examined. For any given value of c, as n approaches infinity, z_n will either approach infinity, or it will not. In the latter case, it may exhibit a cycle, or it may vary chaotically. Those values of c for which z_n does not approach infinity are in the Mandelbrot set.

Thus, it should be fairly simple to create a computer image of the Mandelbrot set. Choose a region of the complex plane, defined by all $a + b\imath$ for which $-1.5 \leq a \leq 1.5$ and $-1.5 \leq b \leq 1.5$. Now, for each pixel in the image, map it to a complex number c in the region, and decide if c is in the Mandelbrot set by examining the convergence of z_n. If it is in the set, mark the pixel black, and if it is outside the set, mark the pixel white.

But there's a problem. The convergence or divergence of z_n is not obvious. Fortunately, it can be shown that, as long as $|c| < 2$, if it is ever true that $|z_n| > 2$, then subsequent iterations will always tend toward infinity. Therefore, we can define a critical magnitude, $m = 2$, and we can stop

M	maximum number of iterations
m	critical magnitude
x_{min}	lowest value of real part of c
x_{max}	highest value of real part of c
y_{min}	lowest value of imaginary part of c
y_{max}	highest value of imaginary part of c
w	width of image in pixels
h	height of image in pixels
z_0	initial value of z

Table 1: Parameters to the Mandelbrot set generation algorithm

iterating if z_n ever exceeds m. But we still need a way to tell if c is in the Mandelbrot set, without testing all possible z_n. For this, we will define a threshold called the "maximum number of iterations", denoted as M. If $|z_n| \leq 2$ for all $n \leq M$, that is, if z_n never exceeds the critical magnitude before reaching the maximum number of iterations, then the point is assumed to be in the Mandelbrot set. This gives a fairly good approximation, assuming a sufficiently high M. In practice, M values of 100-200 yield good results.

4.1.2 Mandelbrot Image Generation Algorithm

Given the above information, it is possible to generate an image of the Mandelbrot set. We will now present a more rigorous algorithm. Before generating an image, certain information must be chosen. How many pixels will be in the image? What region of the complex plane will be graphed? These necessary parameters are summed up in Table 1. The algorithm is as follows:

1. Calculate Δx and Δy. These are the width and height in the complex plane of each pixel. They are calculated as follows:

$$\Delta x = \frac{x_{\text{max}} - x_{\text{min}}}{w} \qquad \Delta y = \frac{y_{\text{max}} - y_{\text{min}}}{h}$$

   Computer pixels are square, so parameters should be chosen such that $\Delta x = \Delta y$, or the image will appear distorted.

2. Iterate through all pixel positions in the image as x and y.

3. For each pixel, calculate c as

$$c = \left(x \cdot \Delta x + x_{\text{min}}\right) + \left(y \cdot \Delta y + y_{\text{min}}\right) \imath$$

   This is equivalent to dividing up the region of the complex plane into w columns and h rows, corresponding to the pixels of the image. Each pixel will be represented by the location of its upper left corner.

4. Decide whether c is in the Mandelbrot set. Calculate z_n for successive values of n using Equation 1, until $|z_n|$ exceeds m, or until n exceeds M. If z_n reaches the critical magnitude before n reaches the maximum number of iterations, then c is not in the Mandelbrot set; otherwise, it is in the set.

5. Plot the point (x, y) black if c is in the set, or white if c is not in the set. Exercise care, because most computer graphics implementations choose to start y at 0 at the top of the screen, and increase y for lower points on the screen. It may be necessary to plot the point $(x, h - y)$ instead.

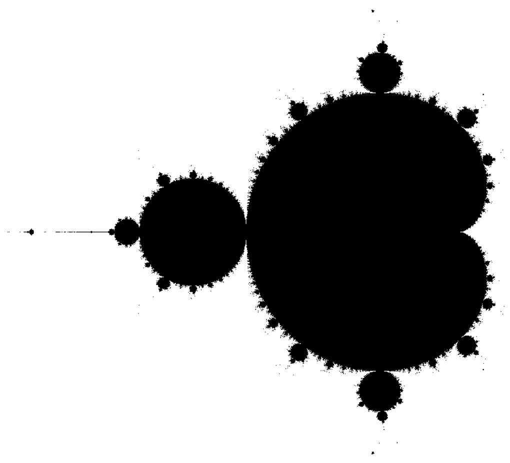


Figure 1: Black and white Mandelbrot set, with M=100

This algorithm can be used to create an image like that in Figure 1. This gives the basic shape of the set, with its characteristic cardioid main body and bulb protuberance on the left. Closer examination reveals that the border is very complex, and that there are smaller bulbs all along the border of the cardiod. The self-similarity of the Mandelbrot set will be examined in closer detail below.

For now, it is enough to notice that the edge between the white and black regions is very detailed and irregular. This is the heart of what makes the set a fractal. Upon closer magnification, this detail grows and grows, with new detail visible at every magnification level down to infinity. Using the algorithm above, smaller details can be examined by adjusting $x_{\text{min}}, x_{\text{max}}, y_{\text{min}}, \text{and} y_{\text{max}}$. However, infinite levels of detail cannot be attained with a computer, because the computer cannot calculate arbitrarily many places after the decimal point without implementing arbitrary-precision arithmetic, as discussed later.

4.1.3 Adding Color

A slight modification to the algorithm will highlight the detail at the edge of the set, and simultaneously make the image more visually appealing. Points in the set will still be plotted black, but points outside the set will be plotted in color. The color will be chosen from a color table (also known as a "palette") based on the number of iterations it took to reach the critical magnitude (called the "escape time"). In this way, it will be possible to visualize *how quickly* z_n approaches infinity.

In a computer, the color of a pixel is represented by three values, corresponding to the amount of red, green, and blue light that are mixed together (recall that the primary colors of light are red, green, and blue). Each color is assigned an integer between 0 and 255, with higher colors representing more of that color present. In this system, the triple $(0, 0, 0)$ represents black, $(255, 255, 255)$ represents white, and $(255, 0, 0)$ represents bright red.

Using this system, and choosing $M = 100$, we can build a table in which bright shades of red indicate a quick divergence to infinity, and dark shades of red indicate slow divergence. As before, black will indicate that the point is in the set. The table will be 101 items long, so as to provide a color for every possible escape time. The table will be denoted with triples t_i, which indicate the color to use with escape time i. Therefore, the red table is calculated as

$$t_i = (255 - 255\left(\frac{i}{101}\right),\ 0,\ 0)$$

This divides the 255 possible values of red into 101 equal parts (because i can be 0 to 100), and implies that $t_0 = (255, 0, 0)$, and $t_{100} = (3, 0, 0)$, which is nearly black. To see an example of the Mandelbrot set generated with this color table, see Figure 2.

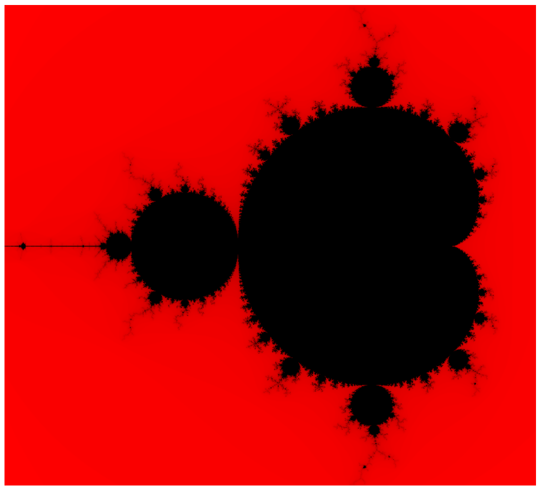


Figure 2: The Mandelbrot set, generated with a red palette

For a much more colorful image, we generated a table that starts with red, and cycles through the colors of the rainbow several times. See Table 2 for a list of the color values of each major color of this computer-generated "rainbow". For each color, the red, green, and blue values are either 0 or 255. To create the rainbow effect, first choose an "increment" value, a factor of 256, which is used to decide the amount of light separating two colors in the table. Start the palette with red, and produce colors that are gradually closer to yellow, by adding successive multiples of the increment. Then decrement the red value to produce green, and continue this process for the rest of Table 2.

For the image in figure 3, we chose an increment value of 32. Thus, a full rainbow is only 48 palette entries. A palette entry is needed for every possible escape time, so we chose to repeat the rainbow enough times to create at least M palette entries.

color	red	green	blue
red	255	0	0
yellow	255	255	0
green	0	255	0
cyan	0	255	255
blue	0	0	255
magenta	255	0	255

Table 2: The colors of a computer-generated rainbow

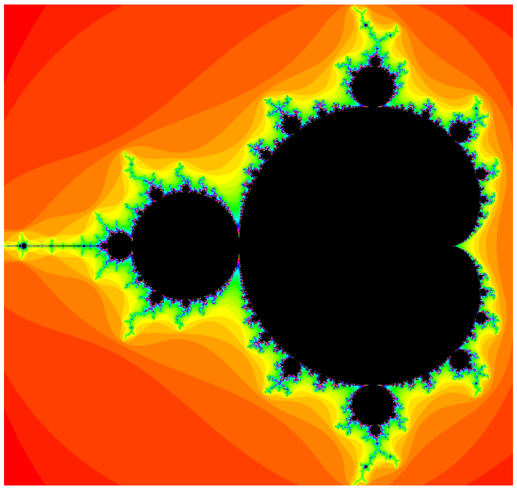


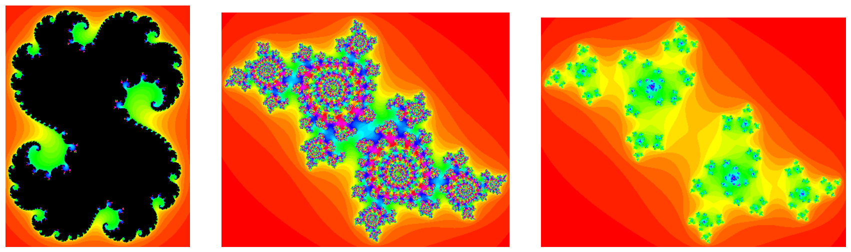
Figure 3: The Mandelbrot set, generated with a rainbow palette

4.1.4 Other Rendering Methods

There are many other ways to choose the color of each pixel. Programs like Fractint (see Appendix B, page 92) have many different formulas to color points outside of the mandelbrot set. For example, some take into account the value of c, or the final value of z, orone of many other things, and they end up with dazzling effects. XaoS will also use a variety of methods to color the points inside the set, which our algorithm rendered as black. The possibilities are endless, and many people have found very visually appealing coloring techniques. Since the Mandelbrot set is at the base of it, the images will always be varicolored and intricately detailed.

4.2 The Julia Set

The Julia set is generated using Equation 1 just like the Mandelbrot set. However, the roles of z_0 and c are reversed. The variable c becomes a parameter to the Julia set, kept the same over all points in the complex region. To test if a complex number b is in the Julia set, we let $z_0 = b$ and iterate, using the same acceptance criteria as in the Mandelbrot set. All other aspects of image generation are the same, so it is very simple to convert a Mandelbrot set generator into a Julia set generator, or to write a program that can do both. In Appendix A, page 81, you will find a listing of *mandelbrot.cpp*, a C++ program that generates images of the Mandelbrot or Julia set, using the rainbow palette described above.



Black region Connected color "Fractal dust"

Figure 4: Examples of the three main types of Julia sets

The Julia set and Mandelbrot set have some interesting ties. Julia sets tend to fall under three categories, shown graphically in figure 4. Depending on the value of c, a Julia image has a black central area (meaning a region with points in the set), or it has one connected area of color, or it is broken up into infinitely many areas of color. It turns out that the type of Julia set for a given c value depends on where the value falls in the Mandelbrot set. If c is in the black body of the Mandelbrot set, the corresponding Julia set will have a black area. If c is in the colorful boundary of the Mandelbrot set, the corresponding Julia will be one connected area of color. Finally, as c moves outside of the colored area, the Julia will break up into "fractal dust", and will consist of disconnected parts. In this way, and other more subtle ways, the Mandelbrot set consists of a "road map" of the Julia sets.

4.3 Self-Similarity of the Mandelbrot and Julia Sets

The Mandelbrot and Julia set rendering program mentioned above is useful for creating static images of the sets, but it is tedious to use it to zoom in and examine areas in finer detail. Fortunately, many programs have been written to serve this purpose, such as Fractint. Using these programs, one can easily look closer at certain details, and, in the case of Fractint, examine down to near-infinite precision (although deeply zoomed images can take days to render).

Look again at the images of the Mandelbrot set above. The shape of the black area is, in general, a cardioid main body with a bulb on the left. Now look at the side of the cardioid, and notice that the bulb shape is repeated much smaller all around the edge of the set. If you zoom in on one of these, you'll find that it's quite a lot like the main bulb. So much so that it, in fact, has its own protuberances, and they each have their own, all quite similar to the main shape. But eventually the pattern disintegrates into one or another of the infinitely varying patterns of the set.

Now examine the whole picture again, and notice the "antenna" off the left side of the bulb. A small distance out is a black lump. Examination in a program such as Fractint reveals that this lump bears a striking resemblance to the entire Mandelbrot set. In fact, there is another lump off of the left end of its spire, and so on, down to infinite levels.

Many other examples of similar structures at different levels of magnification can be found in the Mandelbrot set, and also in the Julia sets. However, the self-similarity is never *quite* perfect, it is more of a "self-likeness". Many fractals are completely self-similar, like the Sierpinski Gasket (which will be examined later). But this is not an absolute requirement. The Mandelbrot and Julia sets are still fractals, because they exhibit fine detail on every level of magnification.

4.4 Higher Exponents

Look once more at Equation 1. What happens if we modify the equation slightly, as follows?

$$z_{n+1} = z_n^3 + c$$

The basic concept of the sets still applies when the exponent is 3, and when it is higher. A sample of Mandelbrot and Julia sets at higher exponents is shown in figure 5. Notice that the Mandelbrot set of degree 3 (M_3) is rotationally symmetric. J_2 already was rotationally symmetric, but J_3 now has three degrees of rotational symmetry. The general rule is that M_n has $n-1$ degrees of rotational symmetry, and J_n has n degrees of rotational symmetry. M_n is also symmetric about the real axis for any n.

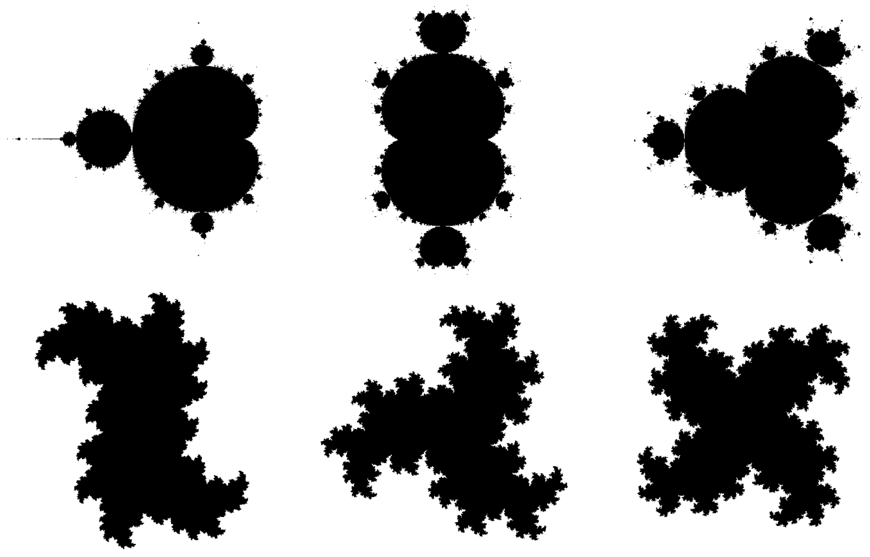


Figure 5: These images show successively higher exponents of z in the Mandelbrot and Julia sets. In the top row are mandelbrot sets, and in the bottom are the corresponding Julia sets. The exponents range from 2 to 4 from left to right.

4.5 Beauty and Art in the Mandelbrot and Julia Sets

Exploring the Mandelbrot and Julia sets can be exciting and enjoyable, especially with programs such as XaoS. XaoS (likely pronounced "chaos") is a real-time fractal movie renderer. This means that you can zoom into an area of the Mandelbrot or Julia sets and watch, in real time, as part of the image gets larger, gradually filling in with more and more detail. You can watch as a miniature mandelbrot set described above comes into view, and then you can explore a small offshoot of that, and then another offshoot, and discover that the tiniest detail of the original image is actually a whole object in and of itself. Then you can zoom all the way back out to the main view of M_2, and you'll see that your playground from a moment ago is now all but invisible. For more information about XaoS, see Appendix B, page 93.

The very concept of infinite detail is fascinating to the mind and often leads to very aesthetically pleasing images. It has analogs to the real world: the tiniest crack in your floor is possibly the home to thousands of little forms of life, and perhaps each one of those is the home to a smaller parasitic life form. Or consider that our planet, when viewed from outside of the galaxy, is insignificantly small, but if you were to "zoom in" on the earth, you'd discover that it has an amazing amount of previously unnoticed detail. In this way, the Mandelbrot set is a "world" of its own, with one important difference: the images obtained from the Mandelbrot set contain infinite levels of detail. All of this results from the deceptively simple Equation 1.

Explorers of the Mandelbrot and Julia sets seem to come in one of two varieties. The first group see the set as a playground for artistic creation. They use the images they find to express their own creativity and artistic ability in new ways. These explorers are interested in coloring methods and artistic manipulation of fractal images. The second group is perhaps more mathematically based, in that they seek to map the world of Mandelbrot, and try to explore previously uncharted areas. These explorers are more interested in the performance of the rendering engine, such as numerical precision and speed of rendering. The division is very approximate, because many artists use techniques from both camps to create their images.

Examples of fractal art are becoming increasingly easy to find on the Internet. Using the popular search engine Google, queries such as "fractal gallery" produces an estimated 96,600 results, although this number is subject to fluctuation. Allowing for cross linking and mentions, one can still draw the conclusion that there are hundreds, if not thousands of fractal artists using the Internet as a place to publish their works. Examining just a small sampling of galleries, one can find amazingly vivid abstract pictures derived from fractals. Some represent natural objects, such as flowers or butterflies, while others seem to have been designed merely to enhance their own chaotic form. And much like classical forms of art, these pictures draw your attention, make you explore their edges, and often convey emotion or even analogies to objects in the world.

It is apparent that many people feel that they are using images of the Mandelbrot and Julia sets, and similar equations, as a playground for their artistic expression. They feel that finding pretty fractal images requires skill, not mere luck. It is the contention of many fractal artists that, while there is a bit of chance involved in finding their images among the sea of possibilities, the image only begins to become artistic expression through careful manipulation.

It is not uncommon for fractal artists to compare their art form to photography, and photography has become a well accepted form of art. Photography went through its own period of acceptance, in which critics expressed the opinion that this form of "art" could not be art, as it lacked an element of creation. The analogy is quite close when you consider that fractal artists are exploring the infinite world of the Mandelbrot set, and selecting and carefully developing the parts they feel show beauty.

4.5.1 Finding interesting fractal images

Almost every fractal artist and explorer has a different method for finding fractal images. It seems that one must get a feel for the patterns behind the set and for what will result from modifications such as zooming, coloring, and changing formulas. There is no definite algorithm for finding interesting Mandelbrot images, such that a computer could be made to produce fractal art. Because of this, human input is essential and is more than simple random selection. After exploring the fractal for a while and learning when certain patterns appear, the artist begins to get a feel for where to look to find interesting pictures. This is not a mathematically describable phenomenon, and it seems to be similar in nature to the kind of aesthetic "sense" that other types of artists develop. By experimenting with the various possible manipulations available to the artist, he begins to learn how to generate the kinds of images he likes.

The most interesting fractal art comes from careful manipulation of the raw images discovered in the Mandelbrot set. These manipulations include choice of coloring method, blending together multiple images, stretching, skewing, rotation, and even modification of the image with standard graphics editing software. Mandelbrot art is far from a purely mathematical art form; there are many places where an artist can add his own expression. This is what separates artistic Mandelbrot creations from mere snapshots of the set.

4.5.2 Deep zooming

Many Mandelbrot set enthusiasts are in awe of the mathematics involved, where a simple formula leads to a map of an infinitely complex world. These artists are interested in exploring and classifying the kinds of images they find, along with what paths are taken to get such images. The self-similarity of the set is apparent to them, as they continue to find the same classes of images in different places in the set. This leads them to explore the technique of deep zooming, in which they attempt to see if some previously undiscovered image is just waiting to be discovered, if only they zoom in far enough.

Casual experimentation with deep-zooming tends to yield very boring results. This is because a vast majority of the areas of the Mandelbrot set turn out to produce similar or identical images no matter how much they are magnified. To see an example of this, use a program such as Fractint, and start zooming in on a colorful region until you find a spiraling shape. If you continue to zoom in on this image, you will find that, while it continues spiraling forever, there is no “new” image to be found no matter how far you zoom in.

At this point, it is necessary to revisit the mathematical side of the Mandelbrot set. Remember that the algorithm described above chooses a sampling of the points in the coordinate system to test for inclusion in the Mandelbrot set. It is important to realize that the images produced are not, in fact, perfect representations of the set and that, no matter how many pixels are used to represent an image, it is always an approximation. The reason behind this is that computer pixels, by their nature, have a finite width, whereas points in the coordinate system are infinitesimally small. Zooming in merely means that you are choosing that your pixels are representative of smaller and smaller portions of the plane, and so their coordinates will differ by smaller and smaller quantities. Unfortunately, computer representation of real numbers is limited in its precision, and so there comes a point at which the computer simply can't accurately represent the difference between the points corresponding to adjacent pixels on the screen. This is referred to as "floating point inaccuracy".

It is possible to get around this, and Fractint, among other programs, attempts to do so. While standard computer processors can represent real numbers to about 16 decimal places, Fractint can represent up to 1600 decimal places. It does this by simulating a single very precise computation through many computations at a level of precision that the computer can handle. And so this high-precision calculation system comes with a price: the deeper the zoom, the longer the computation takes. Some images can take thousands of hours to render completely.

This problem is compounded by the fact that deeper zooms require a higher maximum iteration value. This is because explorations deep into the images of the Mandelbrot set are actually deep examinations of the border between points in the set and out of the set. As you examine more closely, you bring to the foreground previously unnoticed points that take a longer time to reach the critical magnitude, m. A common symptom of this is that the image has large areas of black, often at the center of a spiral. These black images are actually collections of points that take longer than M iterations to reach the critical magnitude, and are mistakenly assumed to be in the set. By raising the maximum iteration count, one will quickly discover that there was more detail waiting to be uncovered. And of course, a higher maximum iteration count means that even more calculations will be required to render each image.

So deep zooming involves even more patience than other kinds of Mandelbrot exploration. Often the effort is futile as the explorer discovers that he has fallen into a spiral that will never look different. To avoid this, practitioners of deep zooming share their opinions and insights as to how to avoid falling into these "self-similarity traps". One technique involves searching for the miniature copies of the Mandelbrot set, affectionately referred to as "minibrots", that seem to spring up at any level of zooming. Experience seems to indicate that, by recognizing the indicators of a minibrot and finding it, one can avoid the deluge of boringly self-similar spirals that can be found all over the Mandelbrot world. The commonly described technique for finding new and interesting shapes in the set involves zooming in deeper and deeper while hopping from minibrot to ever smaller minibrot. The more minibrots you pass, the more likely it is that the structures you find will be different from what has been seen before.

4.5.3 Coloring techniques

Many fractal artists agree that colors are very important to the overall appeal of a fractal image. By choosing what coloring method to use, and which colors to use, the artists find that they can bring out details that weren't visible before. Choosing carefully what colors go next to each other can cause interesting formations to appear. Choosing carelessly can completely hide an interesting formation in a pattern that looks something like television "snow".

Described above is only one example of how to bring color to a fractal image. Unfortunately, no matter how the palette is arranged, there is a fundamental limit to the number of colors used in the image, and that is the maximum number of iterations. Other coloring methods exist that can assign different colors to pixels even if they share the same escape time.

Previously, the only information used to make the decision was the escape time. But one could just as easily color a pixel based on the magnitude of z at the time it reached the critical magnitude, or the angle z makes with the x-axis, or other aspects of z such as its real and imaginary parts. XaoS has many coloring modes available. Some of them involve different functions of the real and imaginary part of z at the time it escapes, involving division, addition and cosines. Some modes also involve the magnitude of z, and the value of c, the point being tested for inclusion in the set. The possibilities are endless, and they often create pleasing effects.

XaoS, among other programs, also has another interesting coloring mode. In the algorithm developed above, points in the set are unilaterally black. However, these points don't all behave exactly the same. Coloring methods sometimes explore the kind of cycles that points in the set fall into, or simply choose colors based on the value of z_M, the value reached at the maximum number of iterations.

Now, we have run up against another limitation of computers. The number of colors that can be represented by triples of integers less than 256 is 16,777,216, and the smallest possible difference between colors is undetectable by the human eye. Unfortunately, earlier computers could not simultaneously represent 16 million colors at once. Original graphics systems represented just black and white. VGA systems could display at most 16 colors at a time, and later 256. In fact, current versions of Fractint still display only 256 unique colors at a time. Much like the palette of colors generated for escape-time coloring as described above, 256-color systems allowed the programmer to select 256 red-green-blue triples to use on the screen at any one time. The reason for this was that each pixel required much less memory to define, because it only required a number up to 256 as an index to the palette.

Present technology allows programmers to use all 16 million colors, with no limit to the number of colors used on the screen at one time. This is referred to as “true color mode”, because it is possible to represent a photograph on the screen with as many colors as can be detected by the human eye. However, it should be noted that not all 16 million colors may be used on the screen at once: at most each pixel will represent a unique color. Most systems have resolutions of 1024x768, or 1600x1200 pixels, which is still under 2 million pixels.

Most fractal artists agree that true color graphics are necessary to fully capture the details of a fractal. They allow the use of free-form calculations to determine he color values of each pixel, such as the coloring modes described above for XaoS. True color mode also allows two images to be blended seamlessly together, as if they were painted on glass and stuck together. True color mode is necessary for very subtle manipulations to the fractal image using standard graphics tools.

4.5.4 Layering

Fractint was originally written as a mathematical toy, to aid in the discovery of new fractals and the exploration of existing fractals. Ultra Fractal is a program that was designed with input from artists, with the purpose of artistic creation rather than mathematical exploration. Ultra Fractal includes many of the features of Fractint, such as deep zooming and user-defined formulas, but it also includes special features of interest to artists.

The most interesting feature is the creation of layered images. Before Ultra Fractal, some artists were beginning to discover that, by layering or combining two different fractal images together, new kinds of images could be created. This involved rendering separate images in a program like Fractint, and combining them in a separate graphics program such as Photoshop. With Ultra Fractal, these operations are available in the same program, and this allows the artist much more freedom to do what he wants with the fractal. Ultra Fractal’s features are describe in more detail in Appendix B, page 95.

Janet Parke is a fractal artist who primarily uses Ultra Fractal to produce her art. On her web site, she describes her method for creating fractal art.

She starts with a palette of shades of grey, so that she can see the objects in the Mandelbrot set, and not get caught up by the colors. When she finds an object that she decides she likes, she tries many different coloring methods, to see which accentuates the image best. She saves many images in this process, until she decides she has explored enough. Then she uses various layering techniques to combine these images with other fractal images, which can bring out details she likes in certain areas of the image. She says that, at this point, some kind of design or style emerges in her mind, and she manipulates Ultra Fractal's layering techniques to express this idea.

This is another place where artistic talent and expression are exercised by fractal artists. Much like the techniques used in finding a fractal image, the artist uses the tools available to her, such as composites, gradients, and transparency, to coerce the fractal image into her idea of beauty. These tools are like the brush and canvas of a painter. The artist learns to use her tools, and learns the effects that they produce. Eventually, she learns to express herself artistically, no matter what the medium is.

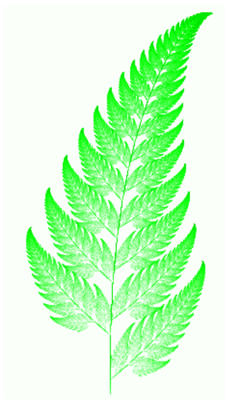


Figure 6: Barnsley's famous fern fractal

5 Iterated Function Systems

Iterated function systems (IFS's) are used to generate fractal images from a very small input data set. Figure 6 shows an image that bears a striking resemblance to the fern *asplenium resiliens*, also known as the black spleenwort. This image was created by Michael Barnsley, and is completely described by a set of only 28 numbers. Iterated function systems use a very small data set to produce highly detailed images that are very provocative to their audiences. In order to understand what these numbers are, and how they're used, some mathematical background is needed.

5.1 Affine Transformations

Iterated function systems are based on affine transformations. An affine transformation in $\mathbb{R}^2$ is just a class of mapping from one image to another. Given an image, an affine transformation may rotate, scale, mirror, skew, and translate an image relative to the origin. Linear algebra tells us that any such transformation on points (x, y) in $\mathbb{R}^2$ can be represented like this:

$$w\left(\begin{bmatrix} x \\ y \end{bmatrix}\right) = \begin{bmatrix} a & b \\ c & d \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} e \\ f \end{bmatrix}$$

The matrix $A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$ will apply rotation, scaling, mirroring, and skewing, and the additional $\begin{bmatrix} e \\ f \end{bmatrix}$ vector will translate the result.

5.2 Contraction Mappings

A contraction mapping is a special kind of affine transformation. As the name suggests, for an affine transformation to be a contraction mapping, it is necessary for it to make the image "smaller". To state this more mathematically, it must be true that the distance between any two points after transformation must be less than or equal to their distance before transformation.

5.3 Example: Sierpinski Gasket

An iterated function system is just a set of contraction mappings. Each of these mappings requires 6 numbers to quantify exactly, using the above definition. One additional number must be attached to each mapping, but we will explain this number later. The fern example mentioned above uses just 4 contraction mappings to fully describe the fractal.

How are the contraction mappings used? We will demonstrate using a simple example with three contraction mappings:

1. Scale the entire image to half size

$$w_1\left(\begin{bmatrix} x \\ y \end{bmatrix}\right) = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

2. Scale the entire image to half size, and translate 0.5 units along the y-axis

$$w_2\left(\begin{bmatrix} x \\ y \end{bmatrix}\right) = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 0 \\ 0.5 \end{bmatrix}$$

3. Scale the entire image to half size, and translate 0.5 units along the y-axis and 0.5 units along the x-axis

$$w_3\left(\begin{bmatrix} x \\ y \end{bmatrix}\right) = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix}$$

For this example, the process starts with the image shown in the upper left of Figure 7. Next, apply each contraction mapping w_i to the image, and remember the result. Finally, combine these three results into one new image (upper right). Now, using this resulting image, repeat the process. Each

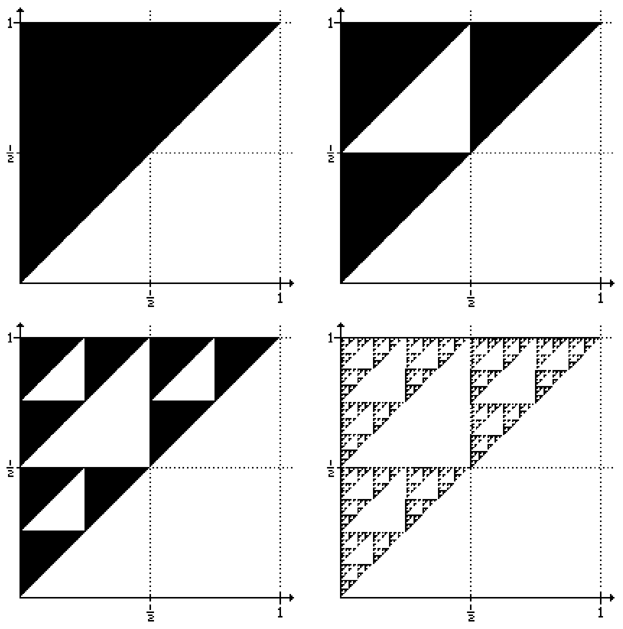


Figure 7: Steps along the way to creating the famous Sierpinski gasket, also known as the Sierpinski triangle. Upper left is the starting image, upper right shows the first iteration, lower left shows 2 iterations, and lower right shows as many iterations as can be printed.

repetition is called an iteration, and the first few iterations are shown in the lower left. If this process is repeated indefinitely, the image will eventually look like the Sierpinski gasket, shown in the lower right. The resulting image is known as the attractor of the IFS.

The original image is eventually scaled down so far that its features are no longer distinguishable. For this reason, one can start with any original image, and the process will always result in the same Sierpinski triangle [Barnsley 1988]! The essence of the final image is completely described by the three contraction mappings, and is not affected by the starting image. The necessity that affine transformations must be contraction mappings is now obvious: otherwise, the image would expand without limit.

5.4 The Non-Deterministic Rendering Algorithm

The process described above is very time-consuming. Computers can help, but there are still many calculations that must be done, and the process is still slow. Fortunately, there is a simpler method which is less CPU-intensive and results in the same image, called the random iteration algorithm. This is where the extra number for each contraction mapping comes in. A probability, p_i, is associated with each contraction mapping i, such that the sum of the probabilities of all mappings is 1. Barnsley suggests that the probability p_i of the affine transformation defined by matrix A_i should be calculated using this formula:

$$p_i \approx \frac{|\text{det}A_i|}{\displaystyle\sum_{i=1}^{N} |\text{det}A_i|}$$

where N is the number of mappings.

To carry out the random iteration algorithm, first select a starting point. Now, randomly select one of the contraction mappings, weighted by the probability. Apply this contraction mapping to the point, and plot the result. Using the resulting point, select a new mapping randomly, and repeat the process. If this process is repeated indefinitely, the resulting image will be exactly the same as above [Barnsley 1988].

In practice, the process is repeated for a set number of iterations. The more iterations, the more accurately the resulting image will represent the attractor of the IFS. With the careful selection of probability values, a clear approximation of the image can be rendered with fewer iterations. Since this randomized algorithm is much faster and easier to program than the deterministic method shown above, most fractal representation programs use it to render IFS's.

5.5 Fractint

Fractint is a very useful program for rendering many types of fractals. Among its many specialties, it can render images from a user-specified or built-in IFS. One such built-in IFS is the Barnsley Fern, depicted in Figure 1.

Fractint takes as input a file with the extension ".IFS". This file can contain one or more IFS, specified as follows:

```
ifs name {
   $a_1$  $b_1$  $c_1$  $d_1$  $e_1$  $f_1$  $p_1$
   $a_2$  $b_2$  $c_2$  $d_2$  $e_2$  $f_2$  $p_2$
   ...
   $a_n$  $b_n$  $c_n$  $d_n$  $e_n$  $f_n$  $p_n$
}
```

Using Fractint, one can generate an amazing array of figures, suggestive of dragons (Figure 8), trees (Figure 9), and even fanciful symbolic structures (Figure 10). The recursive structures that result can mimic nature's self-similarity, as shown by the fern and tree examples, among many more. Also, abstract but captivating structures can be generated. Many beautiful examples of IFS's are included with Fractint. For more information about Fractint, see Appendix B, page 92.

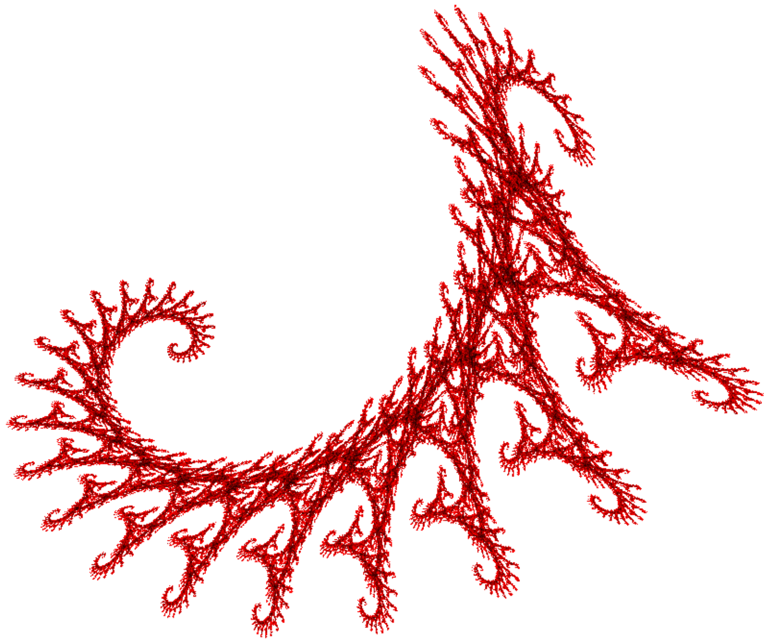


Figure 8: A Fractint built-in IFS that looks like a dragon.

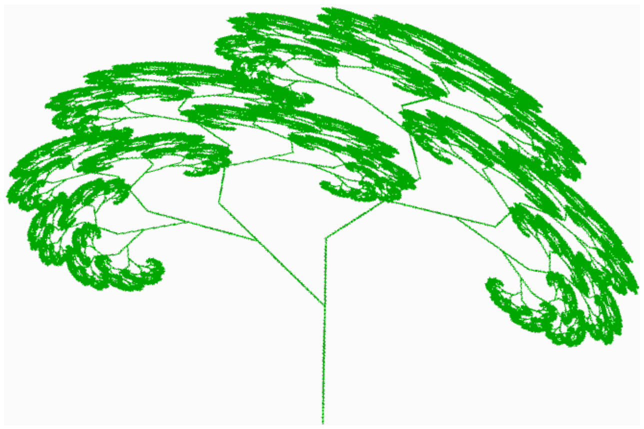


Figure 9: An IFS reminiscent of a tree, created by the authors using FDESIGN and Fractint

5.6 FDESIGN

It is not very easy to design an IFS by hand. Each contraction mapping vaguely specifies that a copy of the "whole image" is to be rotated like so and placed like so, but all of the contraction mappings pull and tug at each other. If a small change is made to just one number of the fern fractal, the stem may become detached, or the branches will no longer attach to the main stem. As such, the process of designing or modifying an IFS is largely trial and error, with the latter occurring frequently.

Enter FDESIGN. FDESIGN was written in 1990 for computers with Intel 8087 math coprocessors by Doug Nelson. It is amazing that this 12 year old program will still run, and it is very interesting that it is still the most useful program of its kind that the authors could find. Given today's processor speeds, FDESIGN can render an IFS very quickly.

FDESIGN allows the user to visually indicate contraction mappings, and watch the IFS render as the mappings are added and modified. Mappings are represented by the action they take on a triangle. First, the user draws a starting triangle, used as a base for all transformations. Then, one draws what happens to this triangle after it undergoes the desired contraction mapping. By observing the starting and ending positions of the three vertices of the triangles, FDESIGN can uniquely determine the parameters for the contraction mapping. This is done by solving a system of equations (Barnsley pp 53).

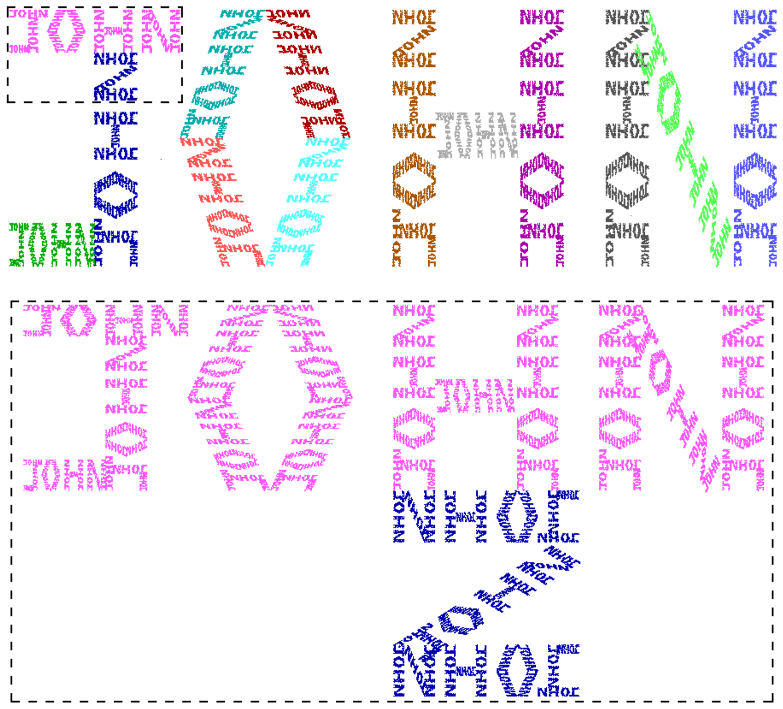


Figure 10: Another IFS created with FDESIGN. The lower image is a zoom-in of the outlined part of the upper image. In the fractal structure approximated by this image, the name “JOHN” repeats down to infinity. John created this image after seeing a similar image designed by the authors of Fractint.

John used FDESIGN to create the "JOHN" fractal in Figure 10. Once he created the contraction mappings, he directed FDESIGN to export the IFS into a format recognized by Fractint and rendered the image shown. By loading the IFS code provided into FDESIGN, the reader can see the triangles that were used to construct the system. There are 13 different contraction mappings in this IFS. One can see them visually in Figure 10, each in a different color. In essence, John used the triangles to specify that the entire image was to appear in each of 13 places so that they spelled out his name.

Though FDESIGN is a powerful tool, it is not trivial to design a meaningful IFS. One must be aware of the meaning of the triangles, and what it means to combine all of the contraction mappings at once. Fortunately, FDESIGN comes with numerous examples, and the interface allows the user to experiment with modifications to the examples. For more information about FDESIGN, see Appendix B, page 95

Iterated Function Systems can create stunning visual effects, as seen above. Because of their recursive nature, they can be made to mimic figures found in nature. They can be used to provoke interesting emotions in people, because they explore our fascination with the concept of infinity. For example, the "JOHN" fractal is extremely interesting because the name seems to be made of itself, with an infinite number of levels that we can never quite understand. This is why Iterated Function Systems evoke emotion, because they intrigue our senses and make us ponder the infinite.

6 L-Systems

From the structure of mountain ranges to the self-similarity of trees and ferns, we see fractal features throughout nature. The purely mathematical study of fractals tends to focus on the attractors of iterative processes, rather than the intermediate steps in the production of the fractal. As natural systems cannot have infinite self-similarity, it is important, when using iterative processes to model natural systems, to pay as much attention to the intermediate results of the process as to the final product. L-Systems are an attempt to model the development of natural systems.

6.1 Origin

In 1968, biologist Aristid Lindenmayer invented a formalization of the description of plant growth that lends itself easily to computer implementation. Today, these formal descriptions are known as parallel rewriting systems, or L-Systems. Lindenmayer noticed that simple life forms, such as some kinds of algae, had very predictable and easy to model reproductive patterns. For example, consider a fictitious plant that grows in long filaments the thickness of a single cell, like some forms of algae. There are two kinds of cells in this imaginary species of algae, type A, and type B. When type A divides, it produces a cell of type A on the left, and type B on the right. Type B splits into a cell of type B on the left, and type A on the right. This is typical of the reproductive patterns of this kind of algae.

We can represent a single filament as a string of letters A and B. To model one step of cell division, simply replace all occurrences of A with AB, and all occurrences of B with BA. See Table 3 for an example of the growth pattern of this algae.

	AB
Step 1	ABBA
Step 2	ABBABAAB
Step 3	ABBABAABBAABABBA
Step 4	ABBABAABBAABABBABAABABBAABBABAAB

Table 3: The first 5 divisions in the growth of the AB algae.

6.2 Lexical Substitution

L-Systems are very close to context-free grammars, a fundamental concept in the field of computer science. The definition of an L-System consists of three major parts: a set of symbols, an axiom, and a set of substitution rules. The set of symbols is the "language" of the L-System; it is the set of characters that make up the string. In the example above, the set of symbols was A, B. To model more complex growth systems, more symbols are used. The axiom is simply the string used to begin the process, and is also referred to as the initial state. In the above example, we used "AB". The substitution rules, also referred to as production rules, are the meat of the L-System. A substitution rule specifies that every occurrence of some symbol should be replaced with a new string. During each step of processing, every substitution rule is applied in order to produce the next string. This is the AB algae example modeled as an L-System:

L-System name	AB Algae
Symbols	A, B
Axiom	AB
Substitution rules	A $\rightarrow$ AB B $\rightarrow$ BA

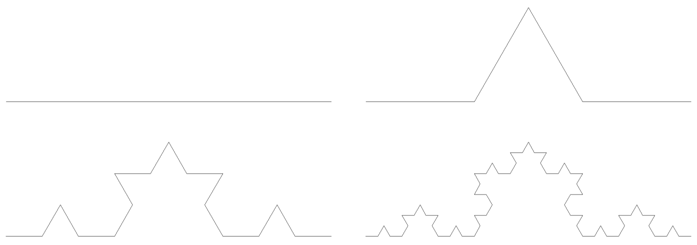
6.3 Graphical Interpretation

L-Systems, by their definition, grow their strings exponentially. The results of the growth process quickly become too big for a human to handle, much less make any useful observations. The strength of the L-System concept is that the grown string can be processed and represented graphically, often with fascinating results. The standard method of visualizing L-Systems is to create special symbols, and give them a graphical interpretation. The growth process is iterated several times, the resulting string is interpreted as a series of graphical commands, and the graphical figure is rendered on a computer. The computer maintains a current position and direction, and it responds to drawing commands such as the following:

F	Move forward some predefined length, while drawing a line.
G	Move forward some predefined length, without drawing a line.
$+$	Turn counterclockwise some predefined angle δ.
$-$	Turn clockwise some predefined angle δ.

Repeated substitutions using the production rules can create a string that is extremely large. It is impossible to tell before rendering how much space the string will require on screen[Peitgen, et al 1992]. Therefore, programs used to render L-Systems must render the entire object in memory, and then discover what scaling factor is required so that the entire object may be viewed on the screen.

With the simple yet powerful definition we have so far, we can render many classical fractals such as the Koch Curve. The definition of the Koch Curve is simple. Start with a line segment of length n, and divide it into three equal parts of length $\frac{n}{3}$. Remove the center segment, and replace it with two segments of length $\frac{n}{3}$, joined to the two existing segments. The process is repeated on every segment in the figure, ad infinitum. Figure 11 shows an example of the rendering of the Koch Curve L-System. How can we model this fractal as an L-System?



L-System name	Koch Curve
Axiom	F
Substitution rules	F $\rightarrow$ F+F–F+F
δ	60 degrees

Figure 11: Images produced from the first four steps of the Koch Curve L-System

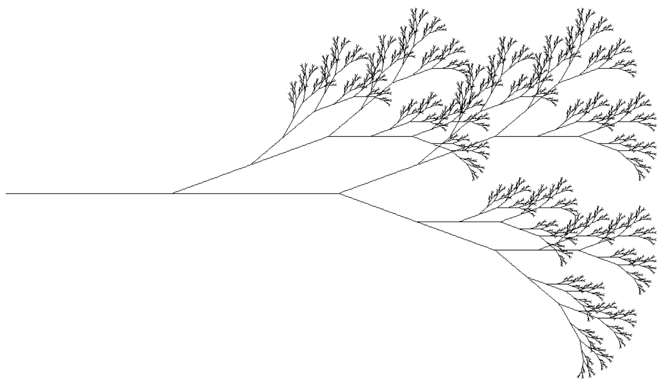
6.3.1 Branching

The graphical symbols defined so far allow one to create complex and interesting figures, but these figures will always be traced by one line or path, whether or not the line is actually drawn on the screen. To extend the capabilities of L-Systems and to allow the rendering of more complex plant growth, more powerful graphical symbols are needed. Most plants of interest don't grow in a straight line; they branch. To render a branching structure, L-Systems provide a method to save and restore the current position and direction. To render a branching point in a plant, first render the stem, then save the position and direction, render the left branch, restore the position, and finally render the right branch. To allow for this, two new symbols are needed:

[	Push the current location and position onto a stack.
]	Pop the top location and direction off of the stack, and use them for further drawing operations.

Because the branches of a structure might, themselves, branch again, it is necessary to be able to store many locations at once. Locations are stacked onto one another in memory. When the pop command is reached, the most recently pushed location is restored as the current location. An example of a branching L system is the simple weed shown in Figure 12.

This L-System has a few notable features. First of all, the symbol B is used, although it has no graphical definition. When an L-System rendering program finds a symbol that has no graphical meaning, it will simply ignore it and continue rendering. This allows for more flexibility in designing the L-System. In this plant, B represents the full plant. An intuitive interpretation of this L-System is that the entire plant is reproduced in three branches off of the main trunk. The infinitely recursive nature of this simple definition is the result of replacing B with a string that also contains B. Also note the first production rule: F $\rightarrow$ FF. This means that, during every stage of growth, the pre-existing portions of the plant double in length. This has the effect of making the structure smaller further away from the trunk, much like a real plant. Figure 12 shows the results of the first several steps of rendering.



L-System name	Weed
Axiom	B
Substitution rules	B → F[+B]F[-B]+B F → FF
δ	20 degrees

Figure 12: Weed L-System

6.4 Fractint

Fractint has a powerful mode for rendering L-Systems. The syntax of an L-System is much like the syntax used for defining an IFS. For example, this is the Koch Curve definition for Fractint:

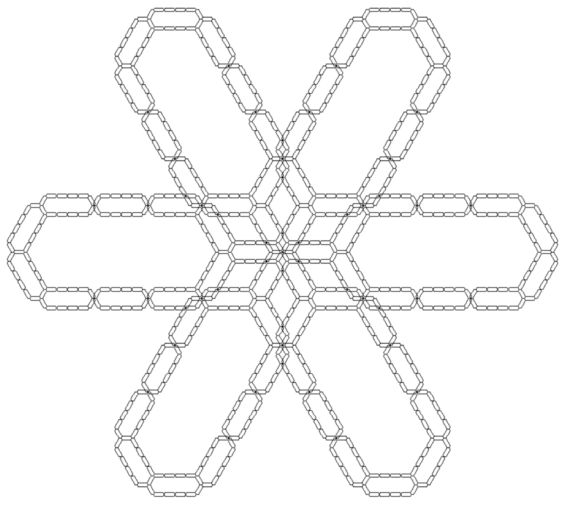
```
KochCurve {  ; This is the name of the L-System
  Angle 8    ; Set the angle to 45 degrees (360/8)
  Axiom F    ; Initial state
  F=F+F--F+F ; production rule for F
}
```

Fractint also defines some more powerful drawing commands:

|	Turn 180 degrees, or the largest possible turn less than 180 degrees, depending on δ
Cnn	Set the color value to nn
>nn	Increment the color by nn
<nn	Decrement the color by nn
!	Switch the meanings of + and -
@nnn	Multiply line segment size by nnn
[,]	Also stores color, line segment size, and ! status

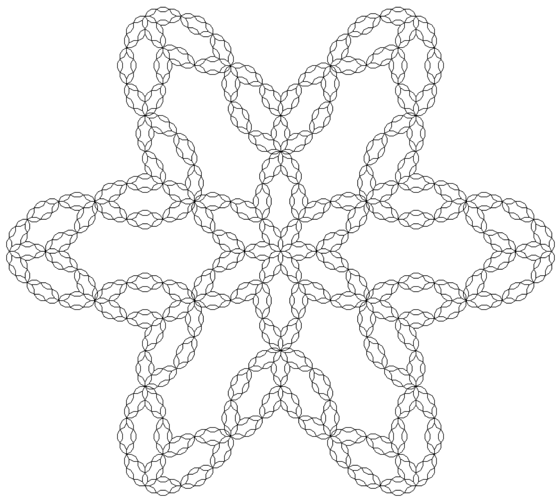
Fractint comes with many predefined L-Systems. To render a custom L-System, use the above definition syntax, and write the L-System to a file with extension ".L". Direct Fractint to load this file instead of its default, and select the L-System by name from the menu.

The authors set out to create an L-System reminiscent of a snowflake. If the Koch Curve is rendered, using "F+F+F" (an equilateral triangle) as the axiom, it looks remarkably like a snowflake. The authors tried to capture this feeling, but using a different style of L-System. The results are shown in Figures 13, 14, and 15.



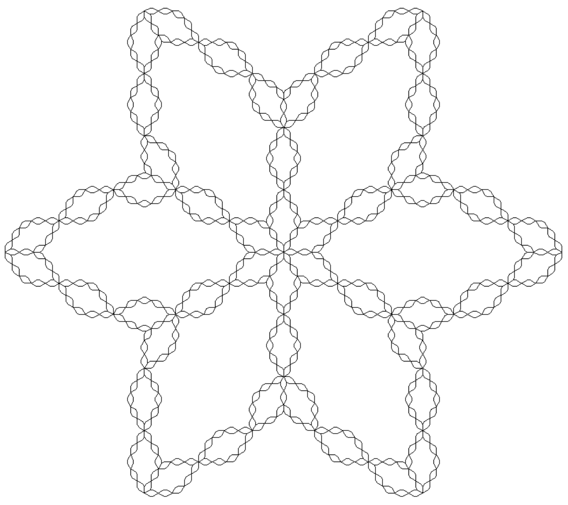
```
SnowFlake1 {
  Angle 12
    ; 30 degrees
  Axiom [F]++[F]++[F]++[F]++[F]++[F]
    ; six 60 degree radial lines
  F=[--F++FFFF++F--]++F--FFFF--F++
    ; replace f with a long pointed bar
}
```

Figure 13: Snowflake L-System 1



```
SnowFlake2 {
  Angle 12
    ; 30 degrees
  Axiom  [F]++[F]++[F]++[F]++[F]++[F]
    ; six 60 degree radial lines
  F=[+F-F-F+]-F+F+F-
    ; replace F with a trapezoid
}
```

Figure 14: Snowflake L-System 2



```
SnowFlake3 {
  Angle 12
    ; 30 degrees
  Axiom [F]++[F]++[F]++[F]++[F]++[F]
    ; six 60 degree radial lines
  F=[+FF-F-FF+]-FF+F+FF-
    ; this trapezoid has longer legs
}
```

Figure 15: Snowflake L-System 3

7 Fractal Dimension

Many people are familiar with the term dimension when it is applied to classical geometric objects such as the line, plane, or cube. It is well known that a line is one dimensional, a plane is two dimensional, and a cube has three dimensions. When we consider the dimension of objects from nature such as trees, clouds, and mountains, we usually say that they are three-dimensional. They are, in fact, real objects that reside in our three-dimensional world. However, when we apply the techniques of fractal geometry to these objects, we get dimension values that are fractions. The fractal dimension of the branching structure of a tree is not three but some number between two and three. This fractional dimension value can sometimes be more useful in describing the structure of an object than our conventional concept of dimension.

We now present one of the many definitions for the notion of dimension that is useful for characterizing the forms of natural objects as well as purely mathematical ones. It will be shown that this definition of dimension provides a convenient method of measuring fractal forms and produces the same dimension values that we would expect to see for classical geometric shapes.

7.1 The Hausdorff Dimension

We choose the Hausdorff dimension, D_H, for our measure of fractal objects. The Hausdorff dimension of a set embedded in a Euclidean space is defined as

$$D_H = \lim_{\epsilon \to 0} \frac{\ln M(\epsilon)}{\ln(\frac{1}{\epsilon})} \tag{2}$$

where $M(\epsilon)$ is the number of hypercubes (a hypercube is a generalization which includes line segments, squares, cubes, etc.) with sides of length ϵ that are needed to cover the set in the space.

We will attempt to clarify this using the simple example of a line segment. Consider a line segment of unit length which is the set of points on the interval $[0, 1]$ in R', and measures of length $\epsilon = \frac{1}{2^n}$ where $n = 1, 2, ...$ The number of measures, $M(\epsilon)$ needed to cover the unit interval is 2^n. Substituting these values into our definition we arrive at

$$D_H = \lim_{\epsilon \to 0} \frac{\ln 2^n}{\ln 2^n} \quad . \tag{3}$$

This gives us our expected value for the dimension of a line segment, 1.

We can also show that the Hausdorff dimension of a square is 2 as we would expect. Consider the unit square $[0,1] \times [0,1]$ in the plane, and measuring squares with sides of length $\epsilon = \frac{1}{2^n}$. To cover the unit square, we need $M(\epsilon) = 2^{2n}$ measuring squares.

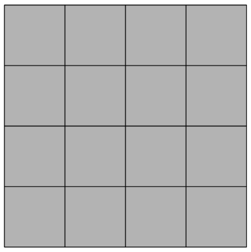


Figure 16: A unit square covered by 16 squares with sides of length 1/4

This leaves us with

$$D_H = \lim_{\epsilon \to 0} \frac{\ln 2^{2n}}{\ln 2^n} \quad . \tag{4}$$

Since as $\epsilon \to 0$ $\quad n \to \infty$, we can rewrite the equation as follows

$$D_H = \lim_{n \to \infty} \frac{\ln 2^{2n}}{\ln 2^n} = 2 \quad . \tag{5}$$

Taking this limit,

$$D_H = \frac{2n \ln 2}{n \ln 2} = 2 \tag{6}$$

Which leaves us with a dimension of 2 for our unit square.

Figure 16 shows a unit square covered by $M = 16$ squares with sides of length $\epsilon = 1/4$. For this example we can observe that $n = 2$. Substituting into Equation 2 easily yields $D_H = 2$.

This method can be used to compute the dimensions of objects with more degrees of freedom such as cubes and hypercubes. But so far we have only shown that the Hausdorff dimension gives us the values that we would expect to see for familiar geometric objects. We will now show that the fractal dimension can be useful in characterizing some less common mathematical objects.

7.1.1 The Koch Curve

The Koch Curve was introduced as part of the discussion of L-Systems in Section 6.3 on page 37. It is an example of a fractal structure that is strictly self-similar. It is composed of an infinite number of copies of itself. Figure 17 shows the Koch curve. In order to construct it we begin with a line segment. This segment is then divided into three pieces of equal length. The middle piece is replaced with the top two legs of an equilateral triangle and the base of this triangle is removed. The result is a curve consisting of 4 line segments each $\frac{1}{3}$ the length of the original line. This procedure is repeatedly applied to the remaining line segments. The limiting object is a self-similar structure called the Koch curve.

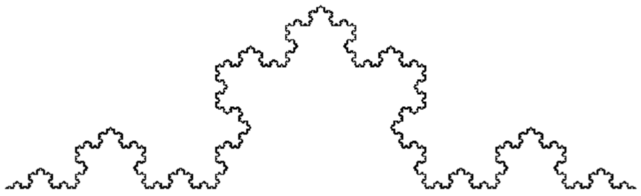


Figure 17: The Koch Curve

To compute the dimension of the Koch curve we use an approach similar to that used with the line segment. Assume that the original construction step is a line segment of unit length. After n steps the number of segments is $M(\epsilon) = 4^n$ each with length $\epsilon = \frac{1}{3^n}$. Substituting into Equation 2 we obtain a value for the fractal dimension of the Koch curve to be

$$D_H = \frac{\ln 4}{\ln 3} \approx 1.26 \tag{7}$$

7.2 Box Dimension

A convenient way of estimating the fractal dimension of nearly any arbitrary structure is the box-counting method. In fact, not only can this method be used to estimate the dimension of classical geometric shapes and all of the fractals discussed so far, but it can be used to find dimension values for all kinds of data sets, not just mathematically defined objects. It is possible to compute the fractal dimension of a mountain or tree given a digital image of the object. For structures in the plane, the method of computing the box-counting dimension is to cover the object with a grid of square cells with sides of length r. Next, count the number of squares $N(r)$ that contain at least part of the structure. Repeat this for varying lengths r. We can then plot on double logarithmic axes the number N cells versus the box-size r. Using common regression techniques, we can fit a straight line to our data points, which corresponds to the relation,

$$N(r) \propto r^{-D_B} \tag{8}$$

where D_B is the box-counting dimension. This assumes that as the box-size r varies, the number of cells N needed to cover the image varies proportional to r^{-D_B}. Since we are plotting this relationship on a *log*/*log* plot, D_B is simply the negative value of the slope of our plot line.

The data points used to generate the following plots were generated by a box-counting program written for this project. The program divides the image into a number of squares of a certain size and then counts the number of squares that contain part of the image (a black pixel). This process is repeated for varying square sizes. Next, the *log* of each count value and square size is output in a text format that is readable by Microsoft Excel. At this point Excel is used to plot the *log*/*log* graph, and linear regression is used to fit a line to the data points. The negative slope of this line is taken as the box counting dimension D_B. The program runs on Microsoft Windows systems and accepts 512 X 512 pixel, monochromatic images in the Device Independent Bitmap (DIB) format. It could, however be easily modified to accept images of various dimensions. For more information please refer to the source code located in Section A.2 on page 84.

7.2.1 Example: The Sierpinski Gasket

Figure 18 contains an image of the Sierpinski Gasket which was discussed in Section 5.3 on page 29. The fractal dimension of the gasket can be computed analytically. It consists of three pieces, each identical to the whole, except scaled down by a factor of 1/2. This means that $M(\epsilon) = 3$ and $\epsilon = 1/2$. By substituting into Equation 2 we can see that the Hausdorff dimension for this object is $D_H = \frac{\ln 3}{\ln 2} \approx 1.58$. Table 4 shows the data produced by running our box-counting program on the image, and figure 19 shows a plot of these data points. As you can see, this technique estimates the dimension as 1.67.

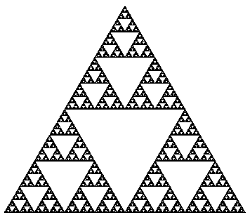


Figure 18: The Sierpinski Gasket

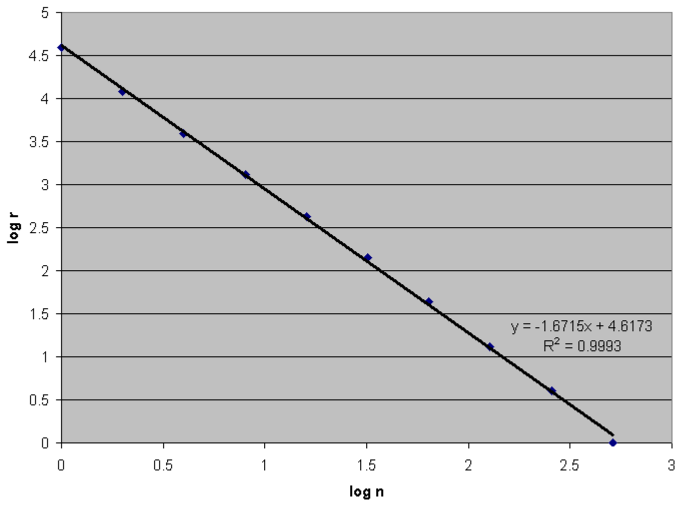


Figure 19: Plot of box-counting measurements for the Sierpinski Gasket. This graph shows the box size r as a function of the number of boxes n needed to cover the image. The slope is shown as -1.6715

$log(r)$	$log(n)$
0	4.59405
0.30103	4.081311
0.60206	3.596487
0.90309	3.118926
1.20412	2.631444
1.50515	2.149219
1.80618	1.643453
2.10721	1.113943
2.40824	0.60206
2.70927	0

Table 4: This table contains the data points generated using our box-counting program on the Sierpinski Gasket.

7.2.2 Example: The Koch Curve

Figure 20 shows the *log*/*log* plot of box-counting data for the Koch Curve. The same program used for the Sierpinski Gasket was used to analyze this image. The slope of the curve is -1.2806. This gives us a box-counting dimension which is very close to the fractal dimension determined analytically earlier.

$$D_H = \frac{\ln 4}{\ln 3} \approx 1.26 \tag{9}$$

For many self-similar structures, such as natural objects and fractal art, it may not be possible to analytically determine dimension. However the box dimension can be used as an appropriate estimator for the Hausdorff dimension. This gives us the ability to apply fractal analysis techniques to a wide range of natural phenomena.

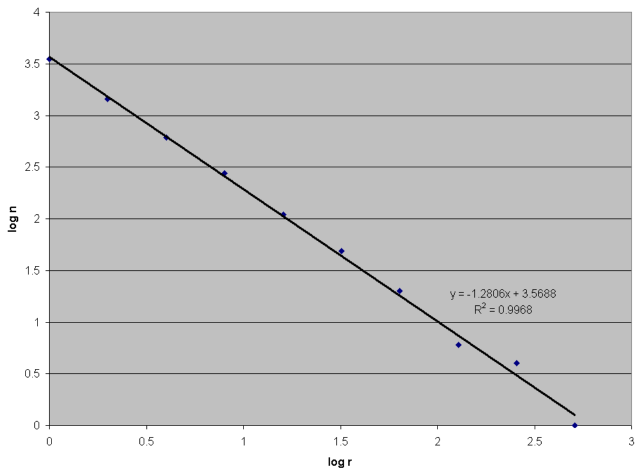


Figure 20: Plot of box counting measurements for the Koch Curve. The box-counting dimension is $D_B = 1.2806$

8 Jackson Pollock's Fractal Paintings

Jackson Pollock began his work on drip paintings in 1942. Today he is considered by many to be the most famous of abstract artists. He created many of his works by dripping paint onto large canvases. His technique is recognized by many as an important advancement in the development of modern art; however, the artistic value of the resulting works is a topic for debate. To the casual observer it may appear that Pollock was simply applying paint in random dribbles,but researchers have recently examined his paintings and discovered the fractal property of self-similarity rather than unorganized, random patterns. One researcher in particular, Richard Taylor, has spent a lot of time exploring the fractal nature of Pollock's work. Taylor is a professor of physics at the University of Oregon. He also has a master's degree in art theory from the University of New South Wales with a focus on Jackson Pollock's work. Two main observations come out of Taylor's work: Pollock's patterns are fractal in a way that reflects nature, and the fractal dimensions of Pollock's paintings increased during his career.

Both images used in this section are from Professor Taylor's web page. [1] In an effort to quantify the fractal nature of Pollock's drip paintings, Taylor and his team used the box-counting method to determine the fractal dimensions of the works. This is the same box-counting method described in Section 7.2 on page 47. They began by overlaying a computer-generated grid on top of the painting and counting the number of squares that contain part of the painted image. This process is repeated with various grid sizes. Next a plot of the number of squares versus the grid size on a log/log axis provides a graphical representation of the fractal dimension. Taylor discovered that, for much of Pollock's work, this graph contained two distinct linear regions. This equates to two separate values for the fractal dimension, dependent on the resolution of the computer generated grid. For square size, r, in the range $1mm < r < 5cm$, one value was found, and for $5cm < r < 2.5m$ another was discovered. Figure 22 shows a box-counting plot of one of Pollock's works. The two different slopes of the line indicate that there are two different fractal dimensions in the piece; one at a large scale and another at a small scale.

The fractal dimension computed for the smaller grid size range is attributed to the dripping process itself. Factors such as paint viscosity, paint absorption into the canvas, etc., are attributed to this chaotic behavior. The

[1] http://materialscience.uoregon.edu/taylor/art/fractal_taylor.html

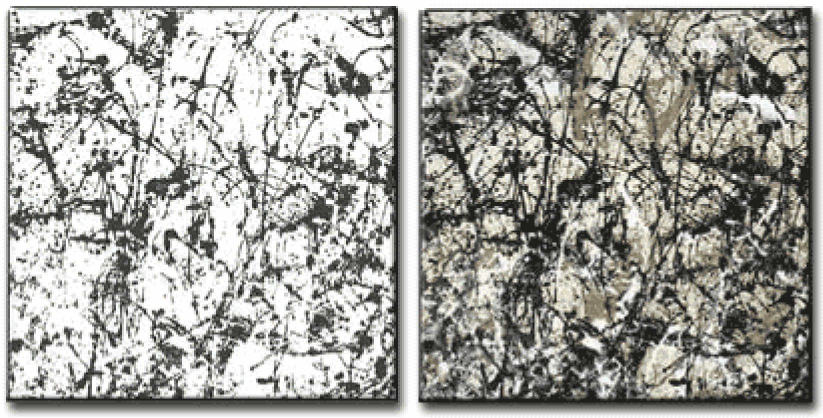


Figure 21: Autumn Rhythm. An example of Pollock's work. The right side shows the whole work while the left side shows just the black anchor layer.

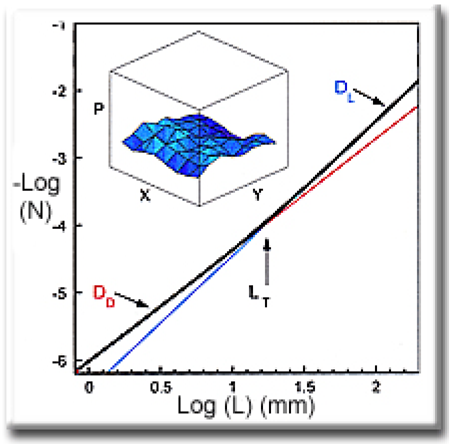


Fig. 3

Figure 22: Box-counting analysis of Pollock's work showing the presence of two different fractal dimensions in one painting.

fractal dimension of the larger square size range is thought to be a direct product of the painter's motion. In other words, even though Pollock did not have any technical knowledge of chaos theory or fractals (these fields of study did not even exist when he was painting), it was still his intent to create fractal patterns. This is supported by Pollock saying that his concerns were with "the rhythms of nature". In Taylor's words, "He described Nature directly. Rather than mimicking Nature, he adopted its language - fractals - to build his own patterns."[Taylor 1999]

When Pollock began making drip paintings in 1943 they consisted of a single layer of paint occupying only about 20% of the canvas area. His later paintings, in contrast, contained multiple layers of paint trajectories that covered over 90% of the canvas. The fractal dimension also increased as he continued his work, starting at values close to 1 and getting as high as 1.72 in 1952. This change in fractal dimension is a reflection of a dramatic change in visual character. Taylor believes that since the fractal dimension increases distinctly with time, fractal analysis could provide a quantitative way to date Pollock's paintings, and to detect forgeries.

It seems that fractal analysis techniques can be valuable tools for the objective and quantitative characterization of abstract visual art. It is interesting to compare the fractal properties of popular art work with the same properties measured from natural objects. Is there a component shared between the artwork of Jackson Pollock and a beautiful mountain range? Further study using fractal techniques could help to quantify beauty.

9 Fractal Music

How are fractals related to music? Can a relation be found at all? Many people who are interested in science are also interested in music due to its mathematical properties. Quite a few people have wondered if they can apply their knowledge of music to the growing field of fractals.

Before we explore this topic, it is important to address exactly what "music" is. Many people have preconceived notions of what music should be. They think of Mozart and Beethoven, with complex interweaving melodies and harmonies. Unfortunately, it has not yet been discovered how to write a program that performs some magic with numbers and fractals, and produces a fully scored piece for a symphony. When exploring how to make music out of fractals, it is important to relax your view of what "music" is.

Since fractals have a strong basis in chaos, fractal music will often sound chaotic and lacking in rhythm, harmony, or discernable melody. This will often clash with a person's preconceived notion of what "music" is, and so when they hear fractal-generated music, they will think, "This isn't music." However, new-age composers are attempting to open up the field of atonal, arrhythmic compositions, and it is in this category that fractal music fits best.

Fractal music has some properties in common with the aleatoric compositions of John Cage. Cage, who is most famous for his piece entitled 4'33 (4 minutes and 33 seconds of silence, in 3 movements), tried to open our ears to alternative forms of music. His compositions were usually formed by utilizing chance events, such as flipping a coin, or by giving vague direction to performers and suggesting they "do what they feel like". Cage claimed to be trying to open our ears to other forms of music that are all around us, such as the nervous coughing during a performance of 4'33. Fractal music is a similar attempt to open our ears to a new form of music, and in this case, that music is derived from mathematical manipulations of fractal systems. However, human involvement is still necessary to guide fractal music into a work of art.

The simplest form of fractal music is demonstrated at the Fractal Music Lab web site [Strohbeen 2000]. The music generation programs found on this site make use of the MIDI protocol, a well established method of designing computer generated music. MIDI music is defined by a collection of numbers, each describing a different component of the music. For example, you can specify the pitch, duration, volume, musical instrument, and many other musical parameters, each with just a number. This is a completely natural playground for fractal-generated music. One can simply map numbers found in a fractal to the different MIDI numerical components, and see what kind of music comes out.

One method of mapping fractals to MIDI musical data is through the use of an iterated chaos function, a concept at the heart of fractals. First, a function and a starting value are chosen. The function is applied to the starting value, and a new value is produced. This new value is mapped to some component of the MIDI data, such as pitch. Then the function is applied to the new value, and another value results. This is repeated until the programmer decides the composition is finished. At each step, the result of the function is used to specify a MIDI parameter, and also to generate a new value.

Another method to generate MIDI data from fractals involves manipulations of the graphical rendering of a fractal. For example, there are programs written that will generate music out of the Mandelbrot set chosen by the user. These programs rely on how computers represent images internally. Recall that computers represent each pixel's color as three numbers indicating how much red, green, and blue light is mixed together. A program will choose a set of points (usually by picking points on a line or on a user-defined path), and use the color values as MIDI parameters.

Other types of programs exist, and there are probably many other fractal music techniques just waiting to be discovered. The concept is simple: just define an algorithm to take numbers generated from a fractal and map them to the parameters of a MIDI music composition.

Those readers that program would be interested to know that the standard pseudo-random number generator software included with almost every computer is a form of iterated chaos function, which is a type of fractal. The random seed that is specified at the start of the program is simply the starting value used in the iteration. So, generating fractal music can be as simple as directing a computer to play "random" notes using MIDI.

9.1 Example 1: Random note generator

In Appendix B on page 85, you can find a listing for the program FractalNoteGenerator. This is actually a pair of classes written in Java™, FractalNoteGenerator and RandomFunction. Those readers with programming experience in Java™or other object oriented languages may be interested in reading the code.

FractalNoteGenerator is a simple program that illustrates the most basic form of fractal-to-MIDI mapping. The FractalNoteGenerator component implements the details necessary to construct and play notes every half second, for 20 seconds. The notes it chooses to play are dictated by the numbers produced by an instance of class RandomFunction. This object uses the built-in chaos function mentioned above, found in Java^{TM}'s Random class, which is an implementation of a pseudo-random number generator. FractalNoteGenerator is designed so that any kind of chaos function can be used, but in this example, the pseudo-random number generator works quite well.

Instructions for running the program are included in the appendix. The results of execution are a chaotic selection of notes that don't sound much like music, but perhaps a composer might hear a sequence of notes that he likes.

9.2 Example 2: Random multiple note generator

In Appendix B on page 86, you can find a listing for the program FractalMultipleNoteGenerator. This builds on the FractalNoteGenerator, playing multiple notes at a time. Instructions for running the program are included in the appendix.

This program attempts to make fractal-generated "chords" by playing up to three notes at a time. The resulting "music" is dissonant, and very difficult to listen to. Whereas a composer might be able to pick out a melody he likes in the previous program, this example is too hopeless to work with and illustrates some of the difficulties in writing fractal music programs.

9.3 Example 3: Random chord generator

In Appendix B on page 88, you can find a listing for the program FractalChordGenerator. This program extends the idea of the FractalMultipleNoteGenerator, but it follows basic chord construction rules so that the notes it plays together aren't dissonant. However, it still makes no attempt to place chords together in such a way that a coherent melody is formed, or so that well-known taboos of music theory aren't broken.

The music that comes out of this third example is a little less painful to listen to than the FractalMultipleNoteGenerator's music. While it is still seemingly random in nature, it sounds a little more full than the FractalNoteGenerator's music. More sophisticated techniques could be used, such as amplifying the soprano line. This would mimic traditional classical music, in which the melody is found in the highest pitched voice. Effort might also be made to try to organize chords in a slightly more coherent manner in line with traditional chord progressions.

9.4 Adding human talent

Usually, the first two examples above generate music that sounds very chaotic and unorganized. It can be interesting on its own, when you consider that you're "hearing" a fractal almost directly, but, as the second example shows, raw fractal notes don't exactly produce pretty music. Instead, some artists use the sounds generated by techniques such as those above as a starting point in their compositions. Usually, one of the most difficult parts about writing music is coming up with an idea for a major theme. Given a theme, a practiced composer can fill out many voices with harmonies and variations on the theme. Using common chord progression practices, the harmonization tends to fall into place after the initial creative inspiration of the melody. The composer knows all of the rules of thumb of harmony and voice leading that make a piece of music sound like a coherent unit, as opposed to random "chords" like in the second example. This is why the harmonies in the third example sound a little more like real music instead of randomly generated notes played together.

Using fractal music techniques such as those described above can help a music writer get over "composer's block," giving him samples of music to work with. Fractal music may offer him full-fledged themes, or it may give him a base to start on. However, due to its chaotic nature, fractal music tends not to have harmonies and rhythms that appeal to our subconsciously conditioned sense of music. In musical composition, only certain notes sound good together ("harmonious"), while others will clash against each other ("dissonant"). Furthermore, certain sets of notes ("chords") may work well together, but they won't sound right when played soon after other chords. This is why it is not generally feasible to use fractals to generate fully scored music akin to Beethoven, because there is a very careful distinction between what sounds good and what doesn't.

A fractal music composer, however, can help guide the raw chaos of fractal-generated MIDI data into a musical composition that makes sense to our ears. A painter does more than choose his paints carefully and then splatter them at random on the canvas; even Jackson Pollock had a careful method. A fractal music composer uses the themes he hears in fractals and builds them up into full-fledged compositions with harmonies that make sense. The resulting music is chaotic and at the same time more structured than the music generated by a program, and is much more interesting to listen to by people whose sense of “music” has been influenced by years of human-created compositions.

10 Art as a Reflection of Society

Art is a reflection of society in many ways, and as society grows more complex, so does our art. Fractal art, due to its complexity, is an example of this. We can easily see many examples throughout history of how art is a reflection of society. Cave drawings found throughout the world tell the story of hunters killing game. Images found in archeological digs in Egypt contain scenes from the religious mythologies of the region as well as historical information. Religious art of the Middle Ages depicts the beliefs of Christians of the time.

The connection between art and society in these examples is obvious. Art serves as a way to express religious beliefs or preserve history, so of course it is a direct reflection of society. It has, however, been proposed that some of the more general features of art, such as style, may also tell us something about the society from which it comes.

One example of art reflecting society in a more general way is the relationship of the song styles of the Mbuti Pygmies and Bushman and their cultures. The musical culture of the pygmies revolves around group singing. Their culture contains virtually no solo song except for lullabies. The vocal lead of their songs switches between members with differing musical talents yet all receive equal vocal support from the group. The pygmies use a yodeling tone in much of their music. This is known to be physically one of the most relaxed vocalizations and is thought to symbolize openness in communication.

This vocal style seems to be matched by the style of the Pygmy culture which is characterized by cooperation. The Pygmy men hunt in groups while the women gather in groups. Food is divided among the people to ensure that everyone gets enough to eat. Law enforcement does not exist among their people. Even the most severe crimes such as murder and theft do not have specific penalties.

While the Pygmies live in the jungle, another society, the Bushman live in the desert. The two groups are separated by 3,000 miles yet surprising similarities have been found between their cultures. Despite environmental, racial, and linguistic differences, both the social structure and song structure of these two groups are quite similar. This is a fact that supports the hypothesis that society affects the style of its art.

In an article entitled Art Styles as Cultural Cognitive Maps, J. L. Fischer theorizes about the relationship of art style to social conditions. Fischer argues that the form the arts take in a specific culture are partially determined by social fantasy. The artist's fantasies about social situations are expressed in the form of the art he produces. In these cases the introduction of personal fantasy into an artist's work is usually not a conscious decision and is not reflected in the subject of the art but rather in the style of the art. As an artist's individual relationship with society is reflected in the style of his art, entire schools of artistic style can be seen to correlate to the cultures from which they come.

Fischer compares egalitarian societies, where people are thought of as being equal and the individual is valued, to hierarchical societies, where social order and rule structures are important. He provides a list of four hypotheses relating artistic style to these two contrasting forms of society and provides data to support these theories. He also believes that complexity of design can be associated with societies that are hierarchical. The past two hundred years have been a time of extreme technological growth. The ability of people to make products, and communicate has increased greatly. Our increased efficiency however has led to increased complexity in our lives. Perhaps this rise in social complexity is reflected in our art. Could this be responsible for the recent embrace of complex fractal patterns in art?

10.1 Art as a Reflection of the Mind

Many psychologists believe that art is a reflection of the inner make-up of the artist. It reflects the desires, fantasies and wishes of the artist, and is an expression of the unconscious mind. Analysis of this expression can lead to a deeper understanding of the individual.

The famous psychotherapist, Carl G. Jung, was one of the primary researchers in the field of psychoanalysis. It was his belief that works of art which were created as part of psychotherapy could be used as a tool in the diagnosis and treatment of various forms of psychosis.

The principal method of using artistic expression in the treatment of mental illness is the technique of active imagination. Figures 23 through 26 show paintings by one of Jung's patients over the course of her ten-year treatment. The paintings represent her progress, showing the growing unity of conscious and unconscious mind. The idea behind active imagination is that the subject allows himself to enter an altered state of consciousness. In this state the ego (reasoning, choice-making part of the mind) becomes more receptive to unconscious thoughts. This altered mental state is similar to

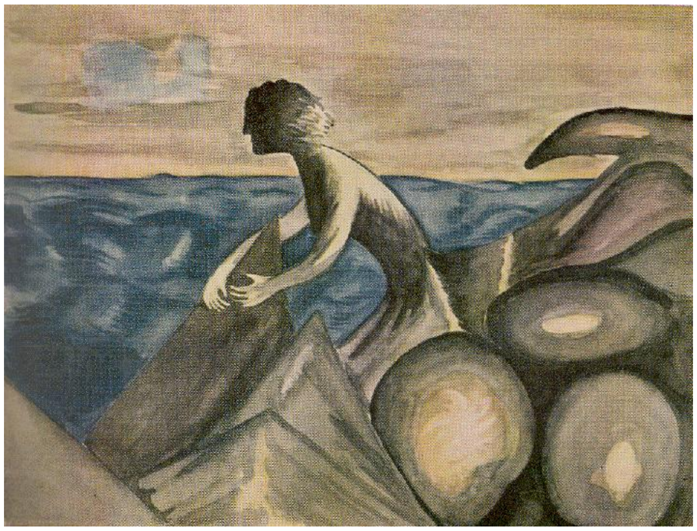


Figure 23: The first painting in a series discussed by Jung, by a woman referred to as Miss X

light hypnosis or meditation. It is important to note, however, that it is not the same as guided imagery where the practitioner is lead by external stimuli or a particular pre-chosen image on which to focus. In active imagination it is important that the patient experiences only inner images and that external images do not become the focus of introspection.

Once in this state the practitioner simply allows the images and thoughts to come out. Irrational thoughts may occur and perception of time and space may change. This is because the ego is allowing the unconscious mind to have more control. The main role of the ego is as a recorder, noting the images that occur. The next phase is for the patient to represent concretely the thoughts he experienced. This can be accomplished through painting, sculpting, writing or any one of a number of media.

To complete the process, the ego observes the result, reacts, and draws conclusions. These conclusions can then be taken advantage of in life, leading to self-improvement.

10.2 Motivation for Art

Throughout history, art has been created for a number of reasons. Two of the most ancient of these motivations appear to be record-keeping and

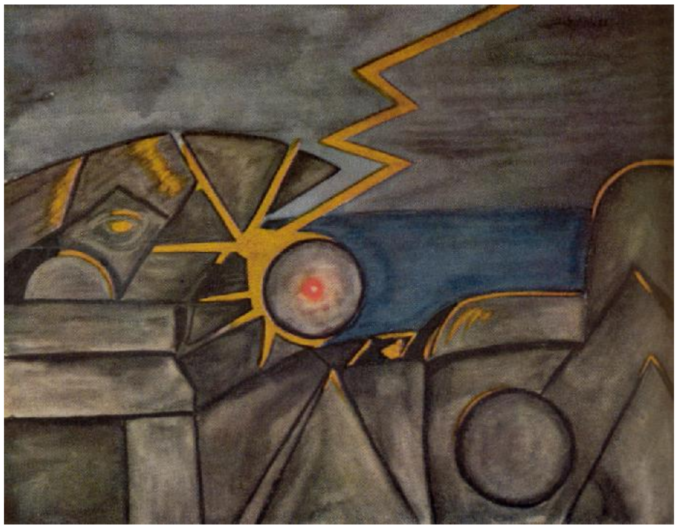


Figure 24: Picture 2 from Miss X

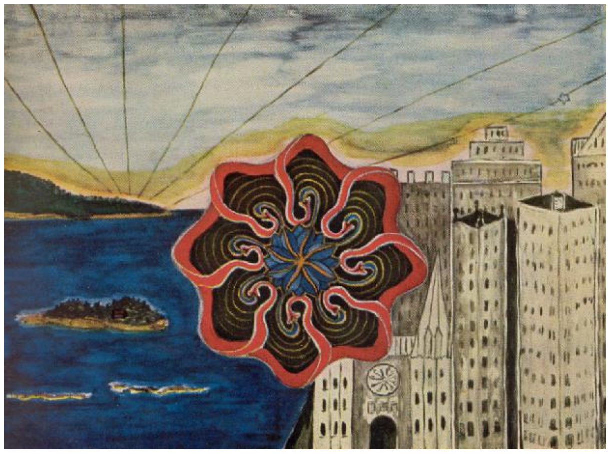


Figure 25: Picture 15 from Miss X

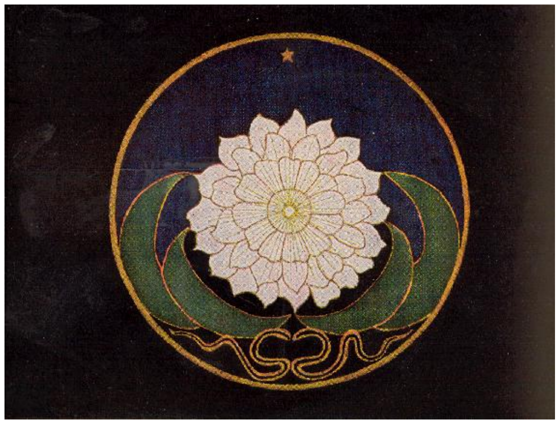


Figure 26: Final image presented by Jung from Miss X (picture 24)

worship. Almost every ancient culture has art that is believed to be religious in nature. The ancient Romans and Greeks made statues and built large temples to honor their gods. The Egyptians built pyramids for their kings who were believed to be gods. From Christianity to tribal religions of Native Americans, art was created for the purpose of worship. Artists have also created art for entertainment and pure aesthetic value. Today, pop music and television provide many of us with the well deserved distraction that we crave after a hard day at the office. Other art and music is created simply for its own sake. Sometimes a painting is just a painting and can be appreciated as such.

Yet another motivation for art is the realization of social goals. Patriotic songs are crafted to give us a sense of unity with our neighbors and to cause us to put faith in our government. Artistic propaganda was used during the cold war to convince people of the evils of communism. Jonathan Swift wrote to criticize society. In Gulliver's Travels he pokes fun at politicians and the upper class of the day. Upton Sinclair used art as a means of changing society. His book, "The Jungle" showcases the working conditions in a Chicago meat processing plant around 1900. His book caused many people to stop consuming meat, and led to the eventual improvement of working conditions.

11 Making Art More Accessible

As we have shown so far, fractal algorithms are fairly easy to understand with a college level background in mathematics. Because they are so interesting to explore and manipulate, fractals are a favorite area of exploration for programmers. So naturally, there are many programs available, for free and for a price, that will help a user explore the world of fractals. These programs are easy to understand even without a background in mathematics, and they continue to attract the interest of many users. These programs, coupled with the Internet, have fueled the formation of online communities of fractal creators who call themselves artists. Fractal artists display and even sell their works on their web sites, and link their pages together to form communities of fractal artists.

With the abundance of web pages written by fractal artists and enthusiasts come many tips, tricks and tutorials for creating fractal art. A few examples of these can be found in Section E on page 100. The beginning fractal enthusiast will find an abundance of advice on how to get started and how to find good-looking fractals, although the advice is, of course, biased toward the view of the authors.

The abundance of fractal programs, along with the Internet as a showcase and learning tool, have attracted many people to try their hands at this new form of art. They feel that it's a lot easier to create art using fractals than with more traditional methods such as paint or pencil. This is especially true for people who feel they have no skill for drawing.

11.1 The Art Question

It's the very fact that fractal "art" is more accessible that brings up the question addressed earlier, of whether fractal art can, indeed, be considered a true form of art. After all, if this art can be created by experimentation with random parameters plugged into a certain fractal formula, is it art? If a computer program happens to choose parameters for a formula that result in a beautiful image, is the program an artist? And if, as suggested above, it's very easy to get started as a fractal artist, with easily available fractal programs and tutorials, can the resulting images really be considered masterpieces of fine art?

Charles Vassallo attempts to address some of these questions in his remarks about fractals and art [Vassallo 1999]. Vassallo is physicist by trade, though he has been experimenting with Markus-Lyupanov fractals since the early 1990s. His fractal images are vivid, swirling objects with a decided three-dimensional quality. Along with showcasing his art, his web site holds his writings on questions related to the subject of fractals and art.

He advocates that pictures of the Mandelbrot set are of dubious artistic value, because they are discovered rather than developed through conscious manipulation. He calls the Mandelbrot set a "virtual amusement park", in which the explorer searches around to find new and exciting images. It seems that he is more interested in fractal art forms that require conscious development, such as his Markus-Lyupanov fractals. He makes the point that, when the explorer begins to craft his own formulas and deliberately form his images, then he is becoming more like an artist.

Janet Parke also has some opinions related to the questions above. Parke is the winner of the Fractal Art contest hosted by Fractalus.com, and was also the winner of an Award of Excellence in Toray's Digital Creation Awards contest. Her art is primarily created through exploration of the Mandelbrot set, with careful use of the extensive features of Ultra Fractal. She creates intriguing works of art using the layering, fading, and coloring capabilities of Ultra Fractal.

In her essay, "Fractal Art – A Deliberate Approach", Janet explores questions about the validity of using technology to create art. She stresses that art is the creation of beautiful products through intentional, conscious work. This purposeful creation is not present when an amateur enters random parameters into a fractal program. She then outlines her unique method for creating her fractal art, which involves the careful manipulation and blending of as many as 12 different layers to create her fractal art.

So, while fractal art may attract many amateurs because they find it easy to get started, this does not mean that just any person can create fractal art. This is obvious when one looks among some of the galleries posted by fractal artists on the Internet. Quite a few galleries are wholly uninspiring, with images that show different levels of artistic skill and expression [Parke 2000]. But among some artists, such as Vassallo and Parke, one can see a distinct style and similarity to all of their pieces, which seems to suggest that they are true artists.

11.2 A New Artistic Paradigm

This seems to suggest that fractal artists share qualities with artists of more traditional media. Not everyone has the skill to be a fractal artist, even though fractals may provide an easier medium for artistic expression than a canvas. By the same token, some artists may find that fractals provide an ideal stage for their talents. In the same way, photography opened the door to an entirely new kind of art. Like photography, fractals provide a new entry point into the world of art that requires different skills than traditional artistic media. It provides a different road that artists can follow to reach the same destination: the creation of beautiful works of art.

Janet Parke shed some light on this subject in an interview she gave after winning the Toray Award of Excellence for her work. She said that, before she began to explore the world of fractals, she didn't consider herself an artist at all. She didn't have the dexterity to create more physical forms of art through painting or drawing, although she was exploring her aesthetic ability by choreographing ballet. When describing her style of art, she said, "In essence, the formulas become the artist's paint and brush, and, like the way a photographer needs a camera to capture his scene, the fractal artist needs a computer to perform the millions of calculations required to render the image."

Fractals, like photography, painting, and drawing, are just another tool that can be used to create art. In the hands of an artist, fractals can be used to create beautiful artistic pictures. They provide a new avenue into the world of art, making it accessible to artists who might otherwise never discover their artistic ability.

12 Psychology of Fractal Art

What is it about fractals that intrigues us? Why are they aesthetically pleasing, and what can that say about our sense of aesthetics? At this point, it will be helpful to call in the opinions of an expert on the subject, Sir Ernst Gombrich. In his book, The Sense of Order, Gombrich explores what makes us perceive beauty in patterned art, and what kind of psychological effects patterns can evoke. His writing makes no specific mention of fractals, but it is interesting to extend his ideas into the field of fractal art.

Gombrich makes an important point, that we cannot easily choose certain features of a figure as the reason behind its beauty. The perception of beauty is largely dependent on who is viewing it, and in what context – beauty is in the eye of the beholder. We can, however, make some general observations about what kinds of things people find beautiful and explore the psychological reasons and ramifications.

One aesthetic quality that Gombrich describes is the concept of restlessness and repose. A completely chaotic picture will tend to draw the eye in all directions at once, and leave the observer confused and uncertain. On the other hand, a completely uniform and ordered design will leave the observer bored. Humans appreciate order, and seek it in the environment around them. Our attention is naturally drawn to places where a pattern is broken. Fractals run dangerously close to being too chaotic to handle, but this danger is averted by self-similarity and patterns. In this way, our eye is drawn to the repeating patterns to explore what makes each repetition different, such as rotation and scaling.

Next, Gombrich describes the sense of symmetry. People tend to notice and explore symmetry, and art is often designed to take advantage of this. Small departure from symmetry is unnoticed, but large departures are distracting. Fractal images occasionally exhibit symmetry, and it is often beneficial to the appeal of the resulting image. However, since fractals have mathematics at their foundations, they run dangerously close to becoming boring through perfect symmetry. Fractal artists must be careful not to bore their audience with too much self-similarity and redundancy.

The metaphorical concept of movement in an image is another important psychological factor that Gombrich discusses. Often, artists attempt to draw our attention and interest by designing figures that imply movement, such as pinwheels and spirals. Fractals often contain spirals and whirls that naturally draw the eye along their path to the center. This is enhanced because the spirals are often made up of self-similar copies of the main image, so the eye is first drawn by the similarity, then by the movement. Spirals are often seen in the Mandelbrot and Julia sets, such as the center image of Figure 4 on page 18.

Another interesting aesthetic quality often found in fractal art is the feeling of depth. By overlapping figures using positioning and shading techniques, the artist can strongly imply that one part of an image is behind another, invoking a three dimensional feel. Examples of this can be found in the images of Charles Vassallo, one of which is shown in Figure 27. His images contain strongly three dimensional objects that overlap and intersect each other, giving an overpowering feeling of depth.

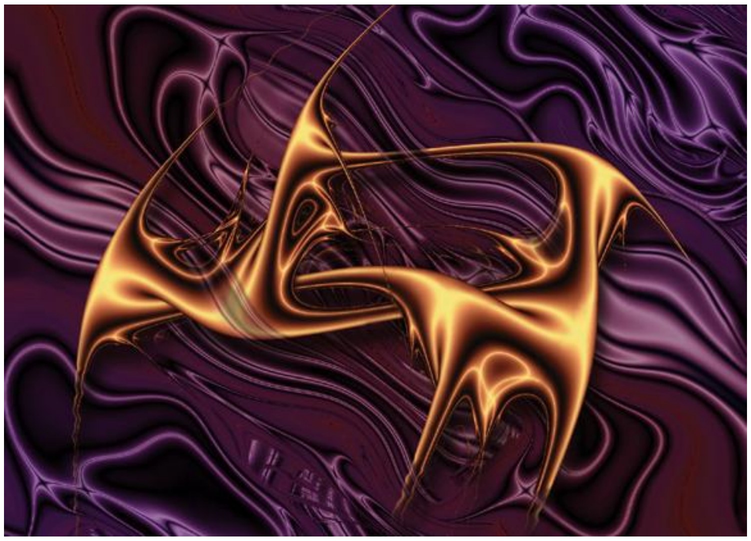


Figure 27: One of Charles Vassallo's images of Markus-Lyapunov fractals.

A very important factor contributing to aesthetic appeal is the use of color. Gombrich argues that, without color, the effects of an image are severely limited. Color is a key part of the fractal rendering process, manipulated carefully by every fractal artist. Artists must ensure that the use of color is not chaotic and confusing as in some of our pictures of the Mandelbrot and Julia sets with a rainbow palette. Color can be carefully manipulated to provide depth and meaning to a piece. It is also a very important factor in separating ground from figure. Fractal artists manipulate their color palettes to emphasize the parts of the image they find most interesting, while muting the rest, so that the important object stands out. Without this, the viewer would have little to focus on, and the image would seem chaotic and meaningless.

This leads to the final important aesthetic tool used by fractal artists: object recognition. Gombrich explains that humans naturally try to find objects they relate to and understand, and this is evident in the process of many fractal artists. They often describe the process of searching through the Mandelbrot set, identifying recognizable constructs such as seahorses and spirals. Fractal artists often try to find recognizable features and modify their rendering method to make an object more familiar. They will occasionally go so far as to give suggestive names to their works, such as Janet Parke's "Iron Gate". It seems as if fractal enthusiasts find some pleasure in finding a recognizable object in the purely mathematical world of fractals.

Fractal artists are careful to consider these aesthetic properties when they create their art. These are the kinds of qualities that separate the work of a true artist from an amateur fractal hunter. Often, an amateur's work will have very few of the above qualities and may even accidentally break some of the "rules" and create something that is actively displeasing or disconcerting. A fractal artist is careful to consider the effects of his work on its audience.

12.1 Vassallo's art

Looking back at Figure 27, what can we say about its artistic characteristics? Charles Vassallo's work has been featured in an exhibition in Lannion, France, and has been submitted to several art competitions. His images seem to depart from the purely mathematical and encompass more of what we might call traditional art. What might Gombrich have to say about this example image?

This image presents a smooth, unbroken object that instantly stands out to the viewer. The eye is drawn across its convoluted contours and to the intersection points, because this is where the continuity of the object is broken. Our eyes are naturally drawn to examine discontinuity, so we will tend to take the long, smooth areas of the object for granted, and examine the intersections and overlaps more closely. In this way, Vassallo carefully manipulates our discontinuity sense to draw us to explore the three-dimensional qualities of the object. There is just enough continuity to keep the image from feeling restless, and there is a balancing discontinuity to keep our eyes moving.

The mathematical idea of symmetry does not apply to this figure. Nevertheless, we can still apply some of the opinions of Gombrich to this image. He suggests that symmetry can have both positive and negative sides. While Vassallo's image does not exhibit symmetry, it also does not disrupt our sense of symmetry. For example, a picture of a roughly symmetric object might distract our attention if it exhibits key asymmetry that is bothersome to our sense of order. Vassallo's object is abstract enough that the mind doesn't even try to find symmetry.

When speaking of movement in a piece of decorative art, Gombrich used the examples of waves, pinwheels, and spirals as objects which imply movement to us. In the background of this example image, we see a wavy purple surface reminiscent of the ocean, giving a strong suggestion of a viscous wavy liquid. The foreground object also gives an impression of twisting and contorting on itself, because of its convoluted structure and strong shading. It is likely that the artist was well aware of the sense of movement that he was creating, and that he intended to hold the viewer's fascination for as long as possible.

This image, among all of Vassallo's available works, was selected based on its extraordinary sense of depth. The artist has carefully shaded the central object to give the impression of a light source, with shadows at overlapping points to suggest that the it folds behind itself. The object itself is abstract, but its depiction of depth and solidity is so concrete that the viewer can easily imagine himself holding it. Our visual system would easily notice if, for example, the light spots could not have resulted from a single light source, and this would likely have given us a feeling of unease or confusion. The artist clearly had to carefully develop the depth of the image, so that the light areas and shadows do not conflict with eachother.

To achieve the effects of movement and depth, Vassallo has made careful use of color. First, he has separated figure from ground by choosing two distinct colors, purple and yellow. The viewer's eyes are instantly drawn to the central yellow object, which is accentuated by the visual cue of the shadow cast on the purple background. Further careful color use creates the sense of depth and solidity in the object. It is no mistake that only two major colors are used in the image. Any more might cause the image to lose its continuity, making it feel more restless.

Finally, how does object recognition apply? It is easy to see that Vassallo is trying to imply some kind of solid object with his image. But this object is completely fictitious, and might possibly play games with space, like the work of M. C. Escher. In a way, it is somewhat distracting that the image depicts an object that the viewer cannot identify. However, the image does not seem to claim that it depicts reality at all; rather, it tries to suggest some kind of mathematical fantasy-world. So, the fact that the viewer cannot identify with the object is not in any way distracting. It keeps the viewer interested, but not disturbed.

Vassallo's work seems to share many qualities with established works of art. In analyzing the qualities of this image, it is clear that the artist was very much aware of how his image would affect its audience. He carefully expressed an idea through his work, and it tends to create emotions of intrigue and fascination. It is the authors' opinion that this image is indeed a work of art.

12.2 The John IFS

Recall the fractal created by one of the authors shown in Figure 10 on page 34. This was created primarily to explore the mechanics of Iterated Function Systems and as a whimsical toy picture. A major focus of this project is to try to decide whether fractal images can be considered works of art. We've shown by example and argument that the answer is not a definite "no", but perhaps an analysis of the John fractal can show that not *all* fractals are works of art.

The John fractal is a purely mathematically defined object, completely described by 91 numbers. It is no surprise that it exhibits a sense of repose so strong as to be boring. People have plenty of experience reading words, even sideways and mirror-imaged. After the initial reading of the image, there is almost nothing left for the eye to explore; the image just sits there. There are no breaks in the pattern to draw our interest, because of its nature as an IFS.

The image exhibits a form of symmetry in its self-similarity. Every occurrence of the name John is made up of smaller versions of the whole image. This is intriguing when the initial concept of infinity sets in, but the feeling of fascination doesn't last for very long. Quickly, the mind becomes bored, because the self-similarity is too extreme. Gombrich points out that symmetry should be tempered with a certain amount of deviation to avoid boredom. For a second time, we have found a reason why the John fractal is likely to bore its audience.

The fractal also fails completely in the area of movement and action.

The fractal is completely devoid of any kind of movement. There are no suggestive shapes, not even a spiral down to infinity that is often a product of the definition of an IFS. This is missing in part because only three levels of self-similarity are distinguishable at the level of detail rendered.

Depth is yet another area in which the John fractal is lacking. Especially with its haphazard coloring and white background, there is nothing to suggest that the fractal is in any way three dimensional, such as overlapping, shadowing, or intersection.

The coloring in this fractal is also a product of its minimalistic mathematical definition. The color of each pixel is chosen by the affine transformation used during each iteration. It is no surprise that this creates a color palette that adds nothing to a feeling of movement and depth in the image. Color is used solely to separate figure from ground, and no care is taken to avoid the appearance of randomly chosen colors.

About the only thing the John fractal has on its side is object recognition. The name "John" is instantly readable, and the concept of infinity is fairly clear after a few seconds of consideration. The author was careful to position each word so that it is easy to read, avoiding upside down and mirrored text when possible. The letter "O" was carefully crafted to be symmetric. These choices were made because inconsistencies in the direction of the text in earlier versions tended to draw attention to certain areas, distracting the viewer.

It is fairly clear that the John fractal is not a work of art. It can best be considered a mathematical joke, worth a few seconds of interest. Unlike the previous example, it does not draw any emotion, and it almost actively bores its audience. There is nothing to keep the viewer interested in the image beyond the first glance.

12.3 Janet Parke's "Then I Saw a New Heaven"

For several years, Fractalus.com hosted a fractal art contest. The 1999 contest drew entries from over 130 fractal artists and selected winners in 13 different categories, such as Emotion, Best Color, Organic, and Mechanical, among others. Janet Parke won several awards for two of her three submissions. We will examine her entry, "Then I Saw a New Heaven", shown in Figure 28, the first place winner of the Quote Interpretation section of the contest. The image is an interpretation of Revelation 21:1,4 from the Bible:

Then I saw a new heaven and a new earth; for the first heaven and the first earth had passed away... and death shall be no more, neither shall there be mourning nor crying nor pain any more, for the former things have passed away.

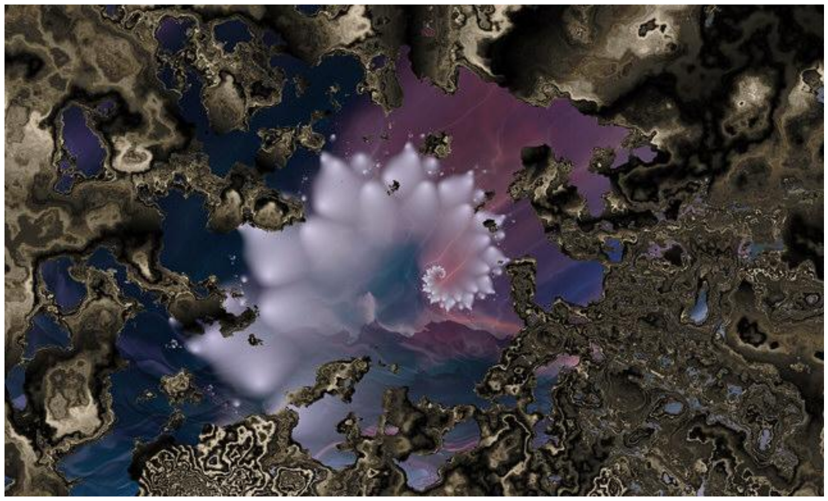


Figure 28: Janet Parke's "Then I Saw a New Heaven"

Her image depicts heaven as a cloudy white fractal figure in the center, surrounded by serenely colored skies. This is framed by a silvery substance reminiscent of mercury, which appears to be opening up to expose Heaven. Another interpretation might be made, but the title and Bible passage tend to guide the mind into perceiving the image in a specific way.

The colors and textures clearly create an image that is not as boringly static as the John fractal. There is a degree of restlessness, created by the rippling structure of the metallic liquid. However, this is not so powerful as to distract the eye from the image of Heaven. Instead, it gives the feeling that the liquid is flowing away to expose Heaven in the center.

Symmetry does not apply to the outer frame, but it can be found in the white clouds in Heaven. The clouds spiral into the center of the image, providing self-similarity. It appears that the similarity is mathematically perfect, and that this is a fractal object processed and colored to look like clouds. Though the similarity has no breaks, it does not fall into the trap of being boring, because it does not try to be the main focus of attention in the image.

The image, with its suggestive title, invites a feeling of movement. It seems to imply that the viewer is swimming through some kind of obstruction. With the context provided by the title and Bible passage, the suggestion is that viewer is moving forward toward Heaven, leaving behind the oily substance of death, mourning, and pain. This is accentuated by the carefully manipulated structure of the foreground matter, which is made to look like a liquid that is sliding away from the center of the image.

The layering of the picture also gives a sense of depth. It is clear that the dark, negative foreground liquid is close to the viewer and that Heaven is further away but close enough to reach for. The artist made careful use of transparency so that Heaven is viewable between all of the gaps in the metallic liquid, making it seem as if the darkness is obscuring Heaven.

Color is used in this image primarily to express emotion. The negative emotions of death and pain are depicted as a filthy oily substance obscuring the view. Heaven, on the other hand, is represented using soft, tranquil colors, giving a sense of peace and happiness. Color is also used to define objects that the viewer can recognize, such as the fractal structure in the center of heaven that is colored to look like a cloud.

It is clear that Janet Parke kept in mind how her audience would react to her image. While the "JOHN" image is merely a fractal rendering, "Then I Saw A New Heaven" is a carefully created work of art, with fractals as the paintbrush, and layering techniques as the canvas.

12.4 Dots

In an attempt to create some original art work, Steve Conte developed the image "Dots", shown in Figure 29, using Ultra Fractal. The authors have little experience creating art, and do not consider themselves artists. Nevertheless, "Dots" is somewhat interesting to look at. In order to fairly assess the artistic merit of this image, it will now be analyzed by John Waymouth. He has been given no knowledge of what process was used to create the fractal so that his opinions are not swayed in any way.

At first glimpse, this image seems as if it suffers a similar problem as the John fractal: its sense of order and repose is dangerously close to boring. However, one's eye is soon drawn to the dot field in the lower right, and the eye starts to try to make some order out of the pattern. Phantom arcs seem

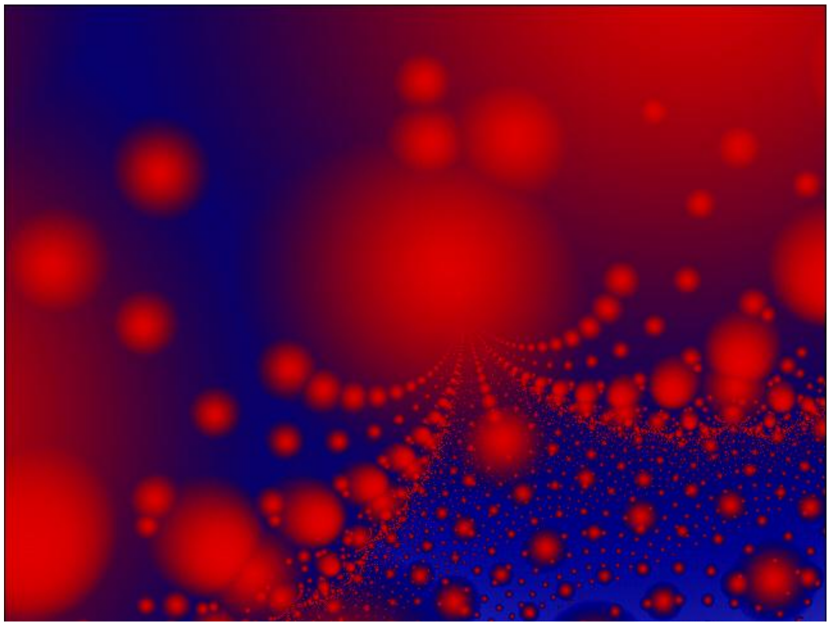


Figure 29: “Dots”, by Steve Conte

to connect the dots in rivalling orientations, and the eye is also drawn to explore the clusters of dots. But after this initial surprise, the image settles down again, and patterns tend not to have much more interesting information to divulge.

Though it may not be obvious at first glance, this image has reflective symmetry along an axis approximately 20° counterclockwise from vertical, which is more noticeable if the viewer tilts his head to match it. Any benefit that might be gained from symmetry is lost because it is not readily apparent, due to its non-standard orientation. On the other hand, this symmetry is mathematically precise, and Gombrich warns against this kind of symmetry because it is boring. Perhaps the mind unconsciously notices the symmetry, and this contributes to the overall emotional feeling created by the image, but I did not consciously notice it until I specifically looked for it. It seems likely that this symmetry is an accident of the rendering technique used, and not a decision of the artist.

This image has a definite sense of depth, created by the use of dots of varying size. As the viewer’s eyes follow the movement of the nearest set of dots, he might feel the need to move into the picture and follow the path. The nearest dots obscure the view, and appear hazy and indistinct, as if the eye is unable to focus on an object so close.

The coloring seems to be as accidental as the symmetry. Red is used for the repeated visual element of the dots, and a rather plain blue is used for the background. In the works above by Parke and Vassallo, the background always had some interesting detail, such as gradients or waves. In "Dots", the background is monochrome, and this lack of detail seems to paradoxically attract attention. When the eye moves to examine the sea of blue more closely, it is not rewarded with any new detail, and this detracts from the emotional experience and leaves a feeling of shallowness. Perhaps black or white would have been better choices, because they are colors more traditionally used as plain, undecorated backgrounds.

"Dots" has no appeal for the object recognition sense that is part of the human visual system. The only structural object in the image is the dot, repeated hundreds of times at varying distances. Its uniformly hazy, circular appearance has no detail to interest the eye, and seems to exist only to define some three-dimensional starfield. There are no objects to engage our interest, except those created by the juxtaposition of groups of dots. The only interesting patterns are in the arcs described above, and the five central cables that extend off to the horizon. Since the termination of these objects is obscured by one of the dots, the eye is not rewarded by its natural tendency to follow the "dotted lines" to their conclusion.

Steve Conte's "Dots" and the John fractal seem to be lacking some underlying quality that is present in the previous examples by Vassallo and Parke. In both of these images, there is visual information that keeps the viewer interested beyond the first cursory inspection. It is possible to catch attention with simple fractal patterns, but it seems to take a true artist to hold our interest and make us want to explore an image more fully. It is clear that fractal artists must carefully consider how their work will be perceived. They must make careful use of the psychological impact of patterns, symmetry, movement, depth, color, and object recognition to stimulate the interest of their audience.

This is why "Dots" and the John fractal are not nearly as artistically interesting as the works by Parke and Vassallo. Since we are not talented and experienced artists, we did not know how to design our images to benefit from the psychological impact that various visual elements have on the audience. When examined as works of art, our fractal images appear just as amateur as any of our artistic undertakings in other media, because we are not accomplished artists.

13 Conclusions

The WPI Projects and Registrar's Office states that the objective of the IQP is to enable WPI graduates to understand how their careers will affect the larger society of which they are a part. It should challenge students to identify, investigate, and report on a topic examining how science or technology interacts with societal structures and values.

This project explored the interaction between the mathematics of fractals and the institution of art. Advances in computing technology have led to the rapid growth of fractals as a research topic, and to the inclusion of fractal images and methods in art. We investigated how fractals are changing art and how fractal art reflects society today. Studying these relationships has made us more aware of the possible implications of future technological developments to the society in which we live. This will make us more aware of the societal implications of the work we do throughout our careers.

In working on this project, we came to an important conclusion: it takes an artist to create fractal art. Although computers are very good at performing the iterative mathematical calculations required to generate a fractal image, they do not have a sense for what makes an image a true piece of art. Images generated solely by a computer, although interesting, can seldom be considered art. These images can, however, provide a good starting point for the artist. Just as a camera is an invaluable tool for the photographer, the computer is an invaluable tool for the fractal artist. In addition to being tools in the development of art, computers and fractals allow more people to participate in the creation of art. Many people who do not have the dexterity of hand required for painting or sketching find that they can express themselves artistically through the creation of fractal art using computers. In this way scientific and technological advancement has opened the door for many people to participate in a rewarding social institution. We feel that fractals have introduced a new paradigm to the world of art.

Our impression of this project has been positive. Through our research we have learned a lot about the mathematical basis for fractals and the many different ways of using fractals to generate art. Additionally, we have observed the way in which a new scientific discovery can affect unexpected areas of society. By studying how mathematical advancements in fractals have affected the world of art, we were made more aware of how technology and science can affect our world. As we go into the work force we will bring this awareness with us, hopefully allowing us to do work that is not only scientifically sound, but also beneficial to the society in which we live.

A Source Code Listings

This appendix contains several programs written by the authors to demonstrate various algorithms related to this report. In order to fit the source code in the space available, it must be rendered in a small font. On some computers, this font can appear as grey lines. If this is the case, please zoom in until text appears.

A.1 Mandelbrot and Julia Set Generator

In section 4.2 on page 18, we described our Mandelbrot and Julia set rendering program. Below is a listing of the source code for this program.

```
/*
 * mandelbrot.cpp
 *
 * Written by John Waymouth as part of the Fractals and Art IQP by John
 * Waymouth and Steve Conte.
 *
 * Fractals and Art: The Mandelbrot Set
 *
 * This C++ program is a simple utility to generate images of the mandelbrot and
 * julia sets in PNG format. The GD library, version 2, is required.  An
 * example of how to compile with GCC in a UNIX environment:
 *
 *   g++ -o mandelbrot mandelbrot.cpp -lgd
 *
 * Usage information can be found by running the program with no arguments.
 *
 * This program is designed for a UNIX environment, but will probably work in
 * any environment that has the GD library, such as CYGWIN for windows
 * (untested).
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <gd.h>

/* Declare a structure to hold a complex number */
typedef struct {
  double real;
  double imag;
} complex;

/* Prototypes for functions to do various complex number operations */
complex mulComplex(complex a, complex b);
complex addComplex(complex a, complex b);
double magSquared(complex a);
complex mulComplex(complex a, double b);

int main(int argc, char **argv) {
  if (argc < 13) {
    printf("Usage: %s <maxIterations> <criticalMagnitude> <xmin> <xmax> \n"
           "      <ymin> <ymax> <width> <height> <exponent> <type> <param.real> \n"
           "      <param.imag>\n\n",argv[0]);
    printf("<type>: specify \"mandelbrot\" or \"julia\"\n"
           "<param.real>, <param.imag>: for mandelbrot, this is the starting z value. For \n"
           "      julia, this is the value of c\n\n");
    printf("The image is output in PNG format on standard output. The column number is \n"
           "printed on standard error after each column is calculated.  Examples:\n\n"
           "mandelbrot 100 10 -2 1.0 -1.5 1.5 500 500 2 mandelbrot 0 0 > mandelbrot.png\n"
           "mandelbrot 100 10 -1.5 1.5 -1.5 1.5 500 500 2 julia 0.375 0.375 > julia.png\n\n"
           );
    return -1;
  }
```

```
/* Initialize variables based on input */

int maxIterations=atoi(argv[1]);
int criticalMagnitude=atoi(argv[2]);
double xmin=strtod(argv[3],NULL);
double xmax=strtod(argv[4],NULL);
double ymin=strtod(argv[5],NULL);
double ymax=strtod(argv[6],NULL);
int width=atoi(argv[7]);
int height=atoi(argv[8]);
int exponent=atoi(argv[9]);

/* Decide if this is a mandelbrot or julia set based on the type argument */
bool mandelbrot=false;
if (strncasecmp(argv[10],"mandelbrot",strlen(argv[10])) == 0) {
  mandelbrot=true;
} else if (strncasecmp(argv[10],"julia",strlen(argv[10])) == 0) {
  mandelbrot=false;
} else {
  printf("<type>: must specify mandelbrot or julia\n");
  return -1;
}

complex param={strtod(argv[11],NULL),strtod(argv[12],NULL)};

/* working data */
complex newz, c;
complex z={0.0,0.0};
int x, y, e;
double xch=(xmax-xmin)/width; /* xch is the width of each pixel */
double ych=(ymax-ymin)/height; /* ych is the height of each pixel */
/* If xch and ych are not equal, the image will be distorted. Therefore, the
 * following equation should hold:
 *     xmax - xmin    ymax - ymin
 *    ------------- = -------------
 *        width          height
 */
int criticalMagSquared=criticalMagnitude*criticalMagnitude;

/* A note about pixel positions: In complex numbers, a greater imaginary part
 * means a higher point on the graph of the complex plane. However, in
 * graphics programming, the highest position is 0, and higher y values mean
 * lower points on the screen. Therefore, when plotting in the loop below,
 * the points must be graphed using (x, height - y).
 */
gdImagePtr im=gdImageCreate(width,height);

/* Generate an interesting palette.
 *
 * The following procedure generates a rainbow palette. If a pixel reaches
 * the critical magnitude in n iterations, then its color is chosen by using
 * the palette entry stored in colors[n].
 *
 * For a detailed description of the algorithm used to generate these colors,
 * please refer to the section entitled "Rainbow color palette".
 */
int inc=32;
int colors[maxIterations + 256*6/inc];
int background=gdImageColorAllocate(im,0,0,0);
int i=0;

/* Start at red */
int green=0;
int red=255;
int blue=0;

while(i+1 < maxIterations) {
  /* transition through orange to yellow */
  for (; green <= 255;green += inc)
    colors[i++]=gdImageColorAllocate(im,red,green,blue);
  green=255; /* would be 256 otherwise, too high */
  /* (255, 255, 0) */

  /* transition to green */
  for (; red >= 0;red -= inc)
    colors[i++]=gdImageColorAllocate(im,red,green,blue);
```

```
      red=0; /* would be −1 otherwise */
      /* (0, 255, 0)

      /* transition through turquoise to cyan */
      for (; blue <= 255;blue += inc)
        colors [i++]=gdImageColorAllocate(im,red,green,blue);
      blue=255;
      /* (0, 255, 255) */

      /* transition to blue */
      for (; green >= 0;green −= inc)
        colors [i++]=gdImageColorAllocate(im,red,green,blue);
      green=0;
      /* (0, 0, 255) */

      /* transition through violet to magenta */
      for (; red <= 255;red += inc)
        colors [i++]=gdImageColorAllocate(im,red,green,blue);
      red=255;
      /* (255, 0, 255) */

      /* transition back to red again */
      for (; blue >= 0;blue −= inc)
        colors [i++]=gdImageColorAllocate(im,red,green,blue);
      blue=0;
      /* (255, 0, 0) */

      /* we've come full circle , repeat again if necessary */
    }

    for (x=0;x<width;x++) {
      for (y=0;y<height;y++) {
        /* choose values for z and c depending on the type of fractal */
        if (mandelbrot) {
          c=(complex){x*xch+xmin,y*ych+ymin}; /* c = pixel position */
          z=param;                             /* initial z is a parameter */
        } else {
          c=param;                             /* c is a parameter */
          z=(complex){x*xch+xmin,y*ych+ymin}; /* z = pixel position */
        }

        for ( i=0;i<maxIterations;i++) {
          newz=z;

          /* z_new = z^exponent + c */
          for (e=1;e<exponent;e++)
            newz=mulComplex(newz,z);
          newz=addComplex(newz,c);

          /* prepare z for the next iteration */
          z=newz;

          if (magSquared(z) > criticalMagSquared) {
            gdImageSetPixel(im,x,height−y,colors[i]);
            break;
          }
        }

        /* If the inner loop hasn't broken before reaching the maximum number of
         * iterations , then the point is in the set , so mark it black
         */

        if ( i == maxIterations)
          gdImageSetPixel(im,x,height−y,background);
      }
      fprintf (stderr ,"col %d\n",x);
    }

    /* print the image to standard output */
    gdImagePng(im,stdout);

    return 0;
}

/* Multiply two complex numbers.
 */
complex mulComplex(complex a, complex b) {
    complex ret;
    ret.real=a.real*b.real - a.imag*b.imag;
    ret.imag=a.imag*b.real + a.real*b.imag;
    return ret;
}

/* Add two complex numbers */
complex addComplex(complex a, complex b) {
    complex ret;
    ret.real=a.real+b.real;
    ret.imag=a.imag+b.imag;
    return ret;
}

/* Compute the square of the magnitude of a complex number. This avoids the
 * necessity of using the sqrt function, for increased efficiency.  Recall that:
 *
 * magnitude(a) = sqrt(a.real*a.real + a.imag*a.imag);
 */

double magSquared(complex a) {
    return a.real*a.real + a.imag*a.imag;
}
```

A.2 Box-Counting Dimension Calculator

Section 7.2 on page 47 described an algorithm for calculating the fractal dimension of an image file. Below is the source code for our box counting fractal dimension calculator.

```
// boxcount.cpp
// Author: Steve Conte
// A windows console application to generate the box-counting dimension of a
// monochromatic bitmap
//
// Note: This program was thrown together and is not horribly user friendly
// Currently assumes the image is a black and white, 512x512 pixel, device
// independent bitmap (.bmp)
// Black pixels are counted as part of the image, white are not.

#include "stdafx.h"
#include "windows.h"
#include <math.h>

int CountBoxes(HDC theDC, int r, int height, int width);
bool CheckBox(HDC theDC, int x_start, int y_start, int r);

#define HEIGHT 512
#define WIDTH 512
#define R_MAX 512

int main(int argc, char* argv[])
{
  if(argc <2)
    argv[1] = "c:\\stevetest.bmp";

  int i, j;
  int r = 1;

  HBITMAP mybitmaph;
  HDC myDC;
  HDC DontUse;
  int blackpixelcount = 0;
  char buff[100];

  DontUse = CreateDC("Display", NULL, NULL, NULL);
  myDC = CreateCompatibleDC(DontUse);
  // Make sure the image is Blank white
  BitBlt(myDC, 0, 0, WIDTH, HEIGHT, NULL, 0, 0, WHITENESS);
  mybitmaph = (struct HBITMAP__ *)LoadImage(0, argv[1], IMAGE_BITMAP,
                                            0,0, LR_LOADFROMFILE);
  SelectObject(myDC, mybitmaph);

  printf("Box-counting for image: %s\nAssuming Image size: %d X %d\n",
         argv[1], HEIGHT, WIDTH);
  printf("Format: r (box size), N (number of boxes containing image), "
         "log(r), log(n)\n");
  for(r=1; r<R_MAX+1; r = r*2){
    blackpixelcount = CountBoxes(myDC, r, WIDTH, HEIGHT);

    printf("%d,%d,%f,%f\n",r, blackpixelcount, log10(r), log10(blackpixelcount));
  }

  return 0;
}

int CountBoxes(HDC theDC, int r, int height, int width)
{
  int boxcount = 0;
  int i, j;

  for(i=0; i<width; i=i+r){
    for(j=0; j<height; j=j+r){
      if(CheckBox(theDC, i, j, r))
        boxcount++;
    }
  }
  return boxcount;
}

bool CheckBox(HDC theDC, int x_start, int y_start, int r)
{
  bool contains = false;
  int i,j;

  for(i=x_start; i<x_start+r; i++){
    for(j=y_start; j<y_start+r; j++){
      if(GetPixel(theDC, i, j) == 0x000000)
        contains = true;
    }
  }

  return contains;
}
```

A.3 Fractal Music Generator

Sections 9.1 through 9.3 starting on page 56 described three example algorithms for generating fractal music. The listings below are a set of Java™ files that implement these three examples. To run the examples, load *fractalmusic.html* in a web browser with Java™ Applet support enabled.

```
/**
 * FractalNoteGenerator.java
 *
 * Written by John Waymouth as part of the Fractals and Art IQP by John
 * Waymouth and Steve Conte.
 *
 * Fractals and Art: Music and Fractals
 *
 * This java class is an example of a program that generates MIDI notes using a
 * chaos function.  Chaos functions are implemented through the IteratedFunction
 * interface, so that they can be interchanged.
 *
 */

import java.util.*;
import javax.sound.midi.*;

public class FractalNoteGenerator {
    private int minNote;
    private int spread;
    private long duration;
    private IteratedFunction func;

    /**
     * @param minNote the lowest MIDI note value to generate
     * @param maxNote the highest MIDI note value to generate
     * @param duration the duration of each note, in milliseconds
     * @param func the source of fractal data
     */
    public FractalNoteGenerator(int minNote, int maxNote, long duration,
                                IteratedFunction func) {
        this.minNote=minNote;
        this.spread=maxNote−minNote;
        this.duration=duration;
        this.func=func;
    }

    public static void main(String[] args) {
        IteratedFunction func=new RandomFunction(System.currentTimeMillis());
        FractalNoteGenerator fng=new FractalNoteGenerator(48,72,500,func);

        fng.generate(20000);
        System.exit(0);
    }

    /**
     * Generate notes using the fractal number source in func. Notes are played
     * using the computer's speakers and MIDI system.
     *
     * @param length how long to play notes for, in milliseconds
     */
    public void generate(long length) {
        try {

            long endTime=System.currentTimeMillis() + length;
            long startTime=System.currentTimeMillis();
            Synthesizer synth=MidiSystem.getSynthesizer();
            synth.open();
            Receiver synthReceiver=synth.getReceiver();

            ShortMessage cmd=new ShortMessage();
            while(System.currentTimeMillis() < endTime) {
                /* use the chaos function to generate the next note value, which is
                 * a number between minNote and minNote+spread */
                int noteValue=func.nextValue()%(spread+1) + minNote;

                /* turn this note on */
                cmd.setMessage(cmd.NOTE_ON,noteValue,64);
                synthReceiver.send(cmd,−1);

                /* wait to send the note off */
                Thread.sleep(Math.min(endTime−System.currentTimeMillis(),duration));

                /* turn the note off */
                cmd.setMessage(cmd.NOTE_OFF,noteValue,64);
                synthReceiver.send(cmd,−1);
            }

            synth.close ();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

```
/**
 * FractalMultipleNoteGenerator.java
 *
 * Written by John Waymouth as part of the Fractals and Art IQP by John
 * Waymouth and Steve Conte.
 *
 * Fractals and Art: Music and Fractals
 *
 * This class is like the FractalNoteGenerator except that it plays 3 notes
 * at once, generating a "chord". This demonstrates that fractals tend not to
 * be good for generating nice−sounding harmonies.
 *
 */

import java.util.*;
import javax.sound.midi.*;

public class FractalMultipleNoteGenerator {
    private int minNote;
    private int spread;
    private long duration;
    private IteratedFunction func;

    /**
     * @param minNote the lowest MIDI note value to generate
     * @param maxNote the highest MIDI note value to generate
     * @param duration the duration of each note, in milliseconds
     * @param func the source of fractal data
     */
    public FractalMultipleNoteGenerator(int minNote, int maxNote, long duration,
                                        IteratedFunction func) {
        this.minNote=minNote;
        this.spread=maxNote−minNote;
        this.duration=duration;
        this.func=func;
    }

    public static void main(String[] args) {
        IteratedFunction func=new RandomFunction(System.currentTimeMillis());
        /* generate all notes in the octave starting at middle C */
        FractalMultipleNoteGenerator fmng=
            new FractalMultipleNoteGenerator(60,72,500,func);

        fmng.generate(20000);
        System.exit(0);
    }

    /**
     * Generate notes using the fractal number source in func. Notes are played
     * using the computer's speakers and MIDI system.
     *
     * @param length how long to play notes for, in milliseconds
     */
    public void generate(long length) {
        try {

            long endTime=System.currentTimeMillis() + length;
            long startTime=System.currentTimeMillis();
            Synthesizer synth=MidiSystem.getSynthesizer();
            synth.open();
            Receiver synthReceiver=synth.getReceiver();

            ShortMessage cmd=new ShortMessage();
            int noteValues[]=new int[3];

            while(System.currentTimeMillis() < endTime) {
                /* use the chaos function to generate the next note values,
                 * which are numbers between minNote and minNote+spread */
                for (int i=0;i<3;i++)
                    noteValues[i]=func.nextValue()%(spread+1) + minNote;

                /* turn all notes on */
                for (int i=0;i<3;i++) {
                    cmd.setMessage(cmd.NOTE_ON,noteValues[i],64);
                    synthReceiver.send(cmd,−1);
                }

                /* wait to send the note off */
                Thread.sleep(Math.min(endTime−System.currentTimeMillis(),duration));

                /* turn the note off */
                for (int i=0;i<3;i++) {
                    cmd.setMessage(cmd.NOTE_OFF,noteValues[i],64);
                    synthReceiver.send(cmd,−1);
                }
            }

            synth.close();
        } catch (Exception e) {
            System.out.println("Exception!");
            e.printStackTrace();
            System.exit(-1);
        }
    }
}
```

```
/**
 * FractalNoteGenerator.java
 *
 * Written by John Waymouth as part of the Fractals and Art IQP by John
 * Waymouth and Steve Conte.
 *
 * Fractals and Art: Music and Fractals
 *
 *
 * This class is like the FractalMultipleNoteGenerator, except that it generates
 * 4-voice chords that are all in the key of C Major. This means that the chords
 * will not sound as dissonant, and that the brain may try to follow individual
 * voices.
 *
 */

import java.util.*;
import javax.sound.midi.*;

public class FractalChordGenerator {
    private int key;
    private long duration;
    private IteratedFunction func;

    /**
     * @param key the first note in the desired key
     * @param duration the duration of each note, in milliseconds
     * @param func the source of fractal data
     */
    public FractalChordGenerator(int key, long duration,
                                 IteratedFunction func) {
        this.key=key;
        this.duration=duration;
        this.func=func;
    }

    public static void main(String[] args) {
        IteratedFunction func=new RandomFunction(System.currentTimeMillis());
        FractalChordGenerator fng=new FractalChordGenerator(48,500,func);

        fng.generate(20000);
        System.exit(0);
    }

    /**
     * Generate notes using the fractal number source in func. Notes are played
     * using the computer's speakers and MIDI system.
     *
     * @param length how long to play notes for, in milliseconds
     */
    public void generate(long length) {
        /* Create lists of chords. Each chord is specified by listing the offset
         * above the bass note, in half-steps. These chords are chosen from
         * various possible positionings of the four voices. Note that the last
         * minor chord is in first inversion for added fun. */
        int majorChords[][]=new int[][]{{0,16,19,24},{0,7,16,19},{0,12,19,28}};
        int minorChords[][]=new int [][]{{0,15,19,24},{0,7,19,24},{0,12,19,27},{0,4,12,21}};

        /* This array tells whether a chord starting at this step should be
         * major or minor (true means major) */
        boolean major[]=new boolean[]{true,false,false,true,true,false,false,true};

        /* This array gives the offsets in half-steps of each note of a major
         * scale */
        int steps[]=new int []{0,2,4,5,7,9,11,12};

        try {

            long endTime=System.currentTimeMillis() + length;
            long startTime=System.currentTimeMillis();
            Synthesizer synth=MidiSystem.getSynthesizer();
            synth.open();
            Receiver synthReceiver=synth.getReceiver();

            ShortMessage cmd=new ShortMessage();
            while(System.currentTimeMillis() < endTime) {
                /* use the chaos function to generate the bass note, by choosing
                 * a step of the scale, finding its half-step offset from steps [],
                 * and using that to calculate the MIDI note value */
                int step=func.nextValue()%8;
                int bassNote=key + steps[step];
                int [] chord;

                /* choose the chord.  Make the decision by using more data from
                 * the chaos function */
                if (major[step])
                    chord=majorChords[func.nextValue() % majorChords.length];
                else
                    chord=minorChords[func.nextValue() % minorChords.length];

                /* turn on all notes of the chord */
                for (int i=0;i<chord.length;i++) {
                    cmd.setMessage(cmd.NOTE_ON,bassNote+chord[i],64);
                    synthReceiver.send(cmd,-1);
                }

                /* wait to send the note off */
                Thread.sleep(Math.min(endTime-System.currentTimeMillis(),duration));

                /* turn off all notes of the chord */
                for (int i=0;i<chord.length;i++) {
                    cmd.setMessage(cmd.NOTE_OFF,bassNote+chord[i],64);
                    synthReceiver.send(cmd,-1);
                }
            }

            synth.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

```
/**
 * IteratedFunction.java
 *
 * Written by John Waymouth as part of the Fractals and Art IQP by John
 * Waymouth and Steve Conte.
 *
 * Fractals and Art: Music and Fractals
 *
 * This interface is for objects that provide numbers through some kind of
 * chaos function.  The nextValue() method mimics the behavior of f(x):
 *
 * x_new=f(x_old)
 *
 * The implementing object rememebers x_old. The initial x value should be
 * passed in through the constructor or by other means.
 */

public interface IteratedFunction {
    /* Return the next number from the chaos function. Return value should be
     * an integer between 0 and 255
     */
    public int nextValue();
}
```

```
/**
 * FractalNoteGenerator.java
 *
 * Written by John Waymouth as part of the Fractals and Art IQP by John
 * Waymouth and Steve Conte.
 *
 * Fractals and Art: Music and Fractals
 *
 * This iterated function class provides fractal data from the java's
 * pseudorandom number generator Random, which is a type of chaos function.
 *
 */

import java.util.Random;

public class RandomFunction implements IteratedFunction {
    private Random random;

    public RandomFunction (long startingX) {
        random=new Random(startingX);
    }

    public int nextValue() {
        return random.nextInt(256);
    }
}
```

```
/*
 * FractalMusicApplet.java
 *
 * Written by John Waymouth as part of the Fractals and Art IQP by John
 * Waymouth and Steve Conte.
 *
 * Fractals and Art: Music and Fractals
 *
 * This class is a simple interface to the three fractal music examples, which
 * may be run as an applet from a web browser. This code just implements a
 * minimal interface, and is not pertinent to the study of fractals and music,
 * so readers may skip this file .
 *
 */

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class FractalMusicApplet extends JApplet implements ActionListener {
    private JTextField durationBtn;
    private JButton noteGeneratorBtn,multiNoteGeneratorBtn,chordGeneratorBtn;
    private IteratedFunction func=new RandomFunction(System.currentTimeMillis());
    private FractalChordGenerator fcg=new FractalChordGenerator(48,500,func);
    private FractalNoteGenerator fng=new FractalNoteGenerator(48,72,500,func);
    private FractalMultipleNoteGenerator fmng=
            new FractalMultipleNoteGenerator(60,72,500,func);



    public void init() {
        Container contentPane=getContentPane();
        contentPane.setLayout(new FlowLayout());

        JLabel durationLabel=new JLabel("Enter duration in seconds:");
        durationBtn=new JTextField("20",10);
        contentPane.add(durationLabel);
        contentPane.add(durationBtn);

        noteGeneratorBtn=new JButton("FractalNoteGenerator");
        noteGeneratorBtn.addActionListener(this);
        contentPane.add(noteGeneratorBtn);

        multiNoteGeneratorBtn=new JButton("FractalMultipleNoteGenerator");
        multiNoteGeneratorBtn.addActionListener(this);
        contentPane.add(multiNoteGeneratorBtn);

        chordGeneratorBtn=new JButton("FractalChordGenerator");
        chordGeneratorBtn.addActionListener(this);
        contentPane.add(chordGeneratorBtn);
    }

    public void actionPerformed(ActionEvent e) {
        if (e.getActionCommand().equals("FractalNoteGenerator")) {
            enableButtons(false);
            fng.generate(Integer.parseInt(durationBtn.getText())*1000);
            enableButtons(true);
        } else if (e.getActionCommand().equals("FractalMultipleNoteGenerator")) {
            enableButtons(false);
            fmng.generate(Integer.parseInt(durationBtn.getText())*1000);
            enableButtons(true);
        } else if (e.getActionCommand().equals("FractalChordGenerator")) {
            enableButtons(false);
            fcg.generate(Integer.parseInt(durationBtn.getText())*1000);
            enableButtons(true);
        }
    }

    public void enableButtons(boolean enable) {
        noteGeneratorBtn.setEnabled(enable);
        multiNoteGeneratorBtn.setEnabled(enable);
        chordGeneratorBtn.setEnabled(enable);
    }
}
```

```
<html>
<body>
<h1>Fractal Music Applet</h1>
This is an interface to three examples of fractal "music". By clicking
any of the three buttons in the applet below, you will start the
generation of notes chosen by a chaos function. In essence, you are
 listening to a fractal.
<br><br>
<i>notes and chords should be played at even time intervals.
 However, occasionally they may sound slightly off, due to a
bug in Java's MIDI software.</i><br><br>
<applet code="FractalMusicApplet.class" width=300 height=200></applet>
<br><br>
To download the code: <a href="fractalmusic.tar.gz">Unix
.tar.gz</a>&nbsp;&nbsp;&nbsp;<a href="fractalmusic.tar.gz">.ZIP
format</a>
</body>
</html>
```

B Programs Used

B.1 Fractint

Authors	Bert Tyler, Tim Wegner, Jonathan Osuch, Wes Loewer, George Martin, Robin Bussell, and various other contributors
Where to find	http://spanky.triumf.ca/www/fractint/fractint.html
Platforms	Written for DOS. Ported to Windows NT, Macintosh, Linux, BeOS.
Distribution	Free to download; source code available

Fractint is an ongoing project that was started in 1988, as an effort to make a Mandelbrot set rendering program that was faster than any program in existence. The way they did this was to use integer calculations to emulate floating point mathematics, hence the "int" in Fractint. This made the program much faster than others that used floating point operations, because integer operations are much easier to calculate. Also, this meant that Fractint could be used on computers without floating point coprocessors. Technology has progressed a lot since then, and floating point calculations can now be computed fairly quickly. But because Fractint had this edge over other programs in the late 1980's, the program has stayed under continuous development and use and has grown to be a program full of useful features.

Fractint is distributed as a DOS program, although we were able to run it in Microsoft Windows XP Home Edition without any trouble. The Unix port didn't work so well for us. The DOS version supports a great variety of video modes and also has support for standard mice. The interface isn't very impressive, but it's clean and simple, and it gets the job done.

Included in the standard distribution is support for over 100 types of fractals, including Mandelbrot and Julia sets, Iterated Function Systems, L-Systems, and many others. Also included is a user-defined formula interpretation module, which allows users to craft their own iteration formulas. For example, one could take the Mandelbrot formula, $z_{n+1} = z_n^2 + c$, and experiment with minor modifications, such as $z_{n+1} = z_n^2 + z_n + c$. Many other variations are possible, such as referring to z_{n-2} and earlier values, adding more parameters, using functions such as sine and cosine, etc. Fractint includes hundreds of premade user defined formulas that run in this module. This means that, as distributed, Fractint comes with built-in support for over

283 different fractals. By searching on the Fractint web site or elsewhere on the Internet, one can find thousands of additional Fractint formulas.

One of Fractint's best features is its implementation of arbitrary-precision floating point numbers. Computer floating point operations are inherently inexact, because it is not possible to store an infinitely repeating decimal number in a computer. This means that most fractal rendering programs cannot zoom in on a fractal indefinitely, because eventually, the computer's inaccuracies are so great that it is impossible to calculate the difference between adjacent pixels with enough precision. Where a modern computer can calculate floating point numbers to 15 decimal digits of precision, Fractint's software emulates a computer that can calculate up to 1600 digits after the decimal point. This allows exploration to lower depths in the fractal, as outlined in section 4.5.2 on page 23. This doesn't come without a price, though. Calculations at very deep levels can take days, weeks, or even months to calculate, even on modern computers.

Among its other features Fractint supports minimal palette editing and loading, though it supports a maximum of 256 colors. Fractint can also save a screen image to disk, including with it the parameters necessary to resume computation. This means that the user can save a screen picture to disk and load it at a later time, picking up right where he left off. This is extremely valuable, because it is highly improbable that anyone can find the same place in a fractal twice without knowing the parameters.

Fractint was used extensively in this project to render Mandelbrot sets, Julia sets, IFS's, and L-Systems. Fractint allows the user to enter the parameters of a fractal image (such as fractal type, zoom level, position, and parameters to the calculation) in a file and load these files for rendering. The reader can find parameter files for many of the figures in this report on the accompanying CD-ROM.

B.2 XaoS

Authors	Jan Hubicka, Thomas Marsh, other contributors
Where to find	http://www.ucw.cz/~hubicka/XaoS/
Platforms	Written for Linux/X-Windows, ported to Windows, DOS, OS/2, BeOS, Amiga, Macintosh
Distribution	released under the GPL (source code freely available)

XaoS is another program that renders Mandelbrot and Julia sets, and a few other kinds of fractals. Its authors don't offer a suggestion of how to pronounce "XaoS", so we suggest calling it "Chaos". The interesting part is that XaoS is designed with the main goal of efficiently rendering animations of fractals. This means that you click on an area of a fractal, and watch, in real time, as that area is enlarged. You can direct the animation as XaoS zooms in, creating your own fractal "movie". XaoS also supports recording transcripts of your exploration, and adding textual notes, pauses, and other effects, to create a fractal presentation that can be shared with friends and colleagues. XaoS comes with several movies that act as a tutorial for all of its features and modes of operation, along with a few that just show off some incredibly beautiful animations. XaoS can also be used as a screen saver by putting it in autopilot mode. In this mode, it uses some heuristics to choose "interesting" parts of a fractal to explore, and zooms in and out automatically.

XaoS is a great program to use to introduce someone to the world of fractals. It is awe-inspiring to watch as an area that was only a tiny speck a moment ago comes into full view, and then races away as even lower depths are brought to the foreground. It is fascinating to the uninitiated to watch as a miniature copy of the Mandelbrot set comes into full view, and then a smaller one, and on even deeper. Then, with a click of a button, you can zoom out entirely to the original picture, and come to grips with the fact that you were exploring the tiniest detail of a detail. XaoS is optimized to use a highly efficient rendering algorithm, so there is no waiting to zoom in as with other programs such as Fractint, which makes it a lot more visually rewarding to explore a fractal.

XaoS has support for true-color video modes, which means that it can display millions of colors at a time. It has quite a few coloring modes for points inside and outside of the set, which allows for some vivid and incredibly beautiful fractal images. XaoS even has a mode for rendering in ASCII art, which means that the screen is drawn in letters, numbers, and punctuation! Unfortunately, the detail is not very easy to pick out, and this mode doesn't seem to have much use other than entertainment. In the same vein, XaoS can generate its fractals as random-dot stereograms, which are also known as "Magic Eye Pictures".

XaoS is an extremely fun program to use, and is a great exploration tool for readers just starting to explore the world of fractals.

B.3 FDESIGN

Author	Doug Nelson
Where to find	http://spanky.triumf.ca/pub/fractals/programs/ibmpc/fdesi313.zip
Platforms	DOS
Distribution	Free to download; source code available

FDESIGN is a program written in 1990 that deals specifically with the design of iterated function systems (see Section 5 on page 28). It was written for DOS, but it runs in Microsoft Windows XP Home Edition without a problem. Though this program is over 12 years old, it does its job admirably, and it is still an extremely useful tool.

FDESIGN allows the user to graphically edit each of the transformations in an IFS individually, and see the results of each modification rendered in real time. This means that, instead of calculating numbers and loading them for rendering in Fractint (by way of a .IFS file), the user can see the results of his modifications and modify his fractal accordingly.

The design of an IFS in FDESIGN is fairly easy, and the documentation provides a fairly good explanation. The user first draws a base triangle, which is used as a reference point for defining the transformations in the IFS. For each transformation the user draws another triangle, which represents what happens to the base triangle after the transformation is applied. After three transformations are defined, the program begins to render a preview of the fractal. The user can then move the vertices of the triangles, add new ones, or delete existing ones. When finished, the user is presented with a full screen rendering of the IFS, along with options to save a screen image, save the file in FDESIGN's internal .TRN file format, or save a Fractint .IFS parameter file. This file can be used to view the fractal in Fractint. All iterated function systems shown in the earlier Section 5 have corresponding .IFS and .TRN files on the accompanying CD-ROM.

B.4 Ultra Fractal

Author	Frederik Slijkerman
Where to find	http://www.ultrafractal.com/
Platforms	Microsoft Windows
Distribution	Shareware, $49 to register

Ultra Fractal is a fractal rendering tool designed for the purpose of artistic creation. It includes many features of Fractint such as built-in formulas and user-defined formulas. It is compatible with user-defined formulas for Fractint, and it supports arbitrary-precision floating point calculations for deep zooming. Formulas may also be written in Ultra Fractal's own formula language, and these formulas are compiled for speed. Thousands of formulas are available on the Ultra Fractal web site, submitted by users of the program.

Ultra Fractal's features are mostly geared toward artists. True color rendering is supported, along with many rendering effects. Users can manually modify the colors used, also referred to as the "gradient". The gradient is a method of color-choosing that allows the user to specify, in general, which colors are used, but also allows for the smooth use of the millions of colors that true color rendering allows.

One of the most powerful features that Ultra Fractal supports is the creation of composite images based on layers. The artist can choose several different fractal images, and stack them on top of each other, deciding how to combine the layers together into a final piece of art. This is similar to printing each image on a sheet of transparency film, and stacking them together to create the final picture. However, Ultra Fractal can combine the layers in many different ways, which has allowed artists to create some incredibly beautiful works of fractal art. Some of these are showcased on Ultra Fractal's web site, but many more can be found through internet searching. Ultra Fractal is widely recognized as one of the best tools for the creation of fractal art, and its users seem to consider it well worth the price.

C Fractal Poster

On a wall in the main stairwell of the WPI mathematics bulding hung a poster depicting, in vivid color, an image of a spiral from a fractal. Along the left side and top was the title, “The Mathematics Art Bridge”. This image is brightly colored, but it was created some time ago, and its effectiveness is somewhat diminished by the low printing quality. The image was 22 inches wide by 28 inches tall, at a resolution of approximately 20 pixels per inch – large enough that individual squares were easily visible.

As part of this IQP, we set out to replace the poster with an updated version using modern printing technology. We selected a portion of the Julia set (described in section 4.2 on page 18) that resembled the original image, focusing on one of the main spirals of the Julia set with $c = 0.375 + 0.375\imath$. We rendered the image using *mandelbrot.cpp* described in appendix A.1, with the following command line:

```
mandelbrot 500 2 -0.16677 0.433232 0.205014 0.805014
  10000 10000 2 julia 0.375 0.375
```

We used the rainbow palette, but rotated it slightly to accentuate the center of the image. This produced a 10000x10000 pixel PNG file in approximately 15 minutes on one of WPI’s servers. We then opened the image in Paint Shop Pro 7, and cropped the image to the dimensions available in the poster. We reduced the image to 9000x6600 pixels, so that it would be a bit larger than 22x28 inches when printed at 300 dots per inch. Next, we added the fading effects and the title text. Paint Shop Pro is unable to lighten a specific area of an image that uses only 256 colors, so we had to increase the image to true color. This means that every pixel is defined by 3 values ranging from 0 to 256. This means that every pixel requires one byte of storage, so the entire image filled approximately 170 megabytes of memory during editing. We had the poster printed by the Academic Technology Center on glossy white paper, and it now hangs in a frame on the wall in Stratton Hall. Figure 30 contains a low detail version of the image.

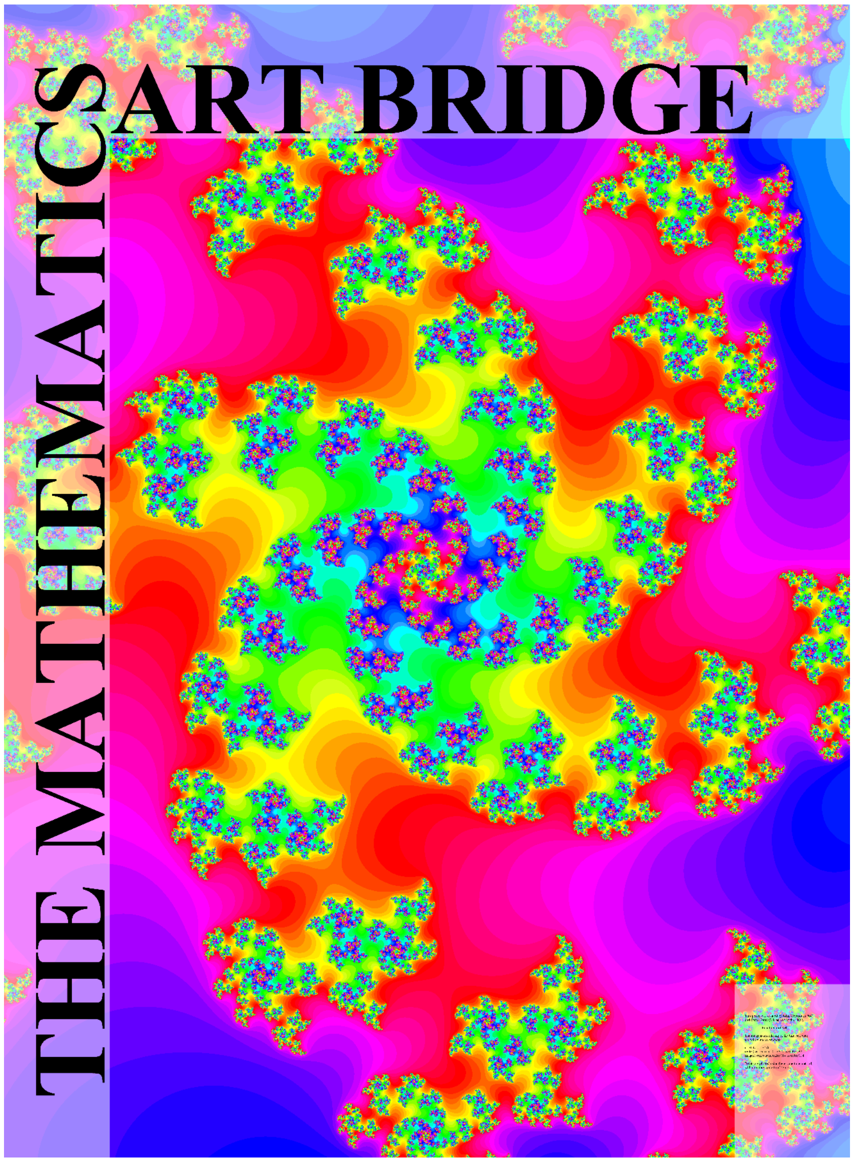


Figure 30: A low detail version of the poster we created for the WPI Mathematics Department

D Supplementary CD

Included with this report is a supplementary compact disc. The disk contains data referred to in this report as well as all of the files used to generate this document. All computers and almost all new audio CD players will be able to access the audio tracks on this disk, but older audio CD players may not be able to. The following materials are included on the CD-ROM:

- This document in PDF format (*report.pdf*)
- Source code for the Mandelbrot/Julia set rendering program, box-counting program, and fractal music generators
- Computer sound files of fractal music examples in WAV format
- CD audio tracks of the fractal music examples that can be played using an audio CD player. Track one is inaccessible in an audio CD player because it contains computer files. Tracks two through four are one-minute samples generated by FractalNoteGenerator, FractalMultipleNoteGenerator, and FractalChordGenerator respectively (see section 9.1 on page 56).
- All files used to generate the report document including:
  - LaTeX files for all sections
  - Computer image files for all figures
  - Instructions for building the report using LaTeX
- Parameter files used to generate figures
- Full-size image of the poster

E References and Further Reading

E.1 Are Fractals Art?

- “Does Fractal Art Exist?” http://perso.wanadoo.fr/charles.vassallo/en/art/intro3.html Vassallo, Charles. 5/99

- “The Fractal Art Manifesto” http://www.fractalus.com/info/manifesto.htm Kerry Mitchell. 1999

- “Fractal Art – A Deliberate Approach” http://www.parkenet.org/jp/writing/ylem.html Janet Parke.

E.2 The Mandelbrot and Julia Sets

- “Guide to the Mandelbrot Set” http://www.globalserve.net/~derbyshire/manguide.html

- “Fractal Geometry of the Mandelbrot Set” http://math.bu.edu/DYSYS/FRACGEOM/ Prof. Robert L. Devaney 2001

- “Fractint: The Mandelbrot Set” http://spanky.triumf.ca/www/fractint/mandelbrot_type.html Noel Griffin

- “Chaos and Fractals” Peitgen, Heinz-Otto. Jürgens, Hartmut. Saupe, Dietmar. Springer-Verlag, New York 1992. pp769-896

- “Fractals: An Animated Discussion” Peitgen, H-O. Jürgens, H, Zahlten, C.

- “Fractals and Chaos” Crilly, A.J, Ernshaw, R.A, Jones, H. Springer-Verlag, New York 1990.

- “Google Search: fractal gallery” http://www.google.com/search?q=fractal+gallery Google 2003

E.3 Iterated Function Systems

- “Fractals Everywhere” Barnsley, Michael. Academic Press, Boston, MA. 1988.

- “Affine Transformation” http://mathworld.wolfram.com/AffineTransformation.html Eric W. Weisstein, Wolfram Research 1999

- "Fractint: Iterated Function Systems" http://spanky.triumf.ca/www/fractint/ifs_type.html Noel Griffin
- "FDESIGN Source Code and DOS Executable" http://spanky.triumf.ca/pub/fractals/programs/ibmpc/fdesi313.zip Doug Nelson, 1990
- "Chaos and Fractals" Peitgen, Heinz-Otto. Jurgens, Hartmut. Saupe, Dietmar. Springer-Verlag New York, Inc. 1992.

E.4 L-Systems

- "Chaos and Fractals" Peitgen, Heinz-Otto. Jurgens, Hartmut. Saupe, Dietmar. Springer-Verlag New York, Inc. 1992.
- "Fractint: L-Systems" http://spanky.triumf.ca/www/fractint/lsystem_type.html Noel Griffin
- "Fractint L-Systems Tutorial" http://spanky.triumf.ca/www/fractint/lsys/tutor.html William McWorter 1997

E.5 Fractal Dimension

- http://plus.maths.org/issue9/turner2/index.html
- http://math.bu.edu/DYSYS/chaos-game/node6.html
- http://www.weiheustephan.de/ane/dimensions/dimensions.html
- http://www.fch.vutbr.cz/lectures/imagesci/download/nummetfradim.pdf
- "Chaos and Fractals" Peitgen, Heinz-Otto. Jurgens, Hartmut. Saupe, Dietmar. Springer-Verlag New York, Inc. 1992. pp769-896

E.6 Jackson Pollock's Fractal Paintings

- Taylor, Richard R., *Order in Pollock's Chaos* in Scientific American, Dec 2002, Vol. 287 Issue6, p116, 6p, 1c, 4bw
- Taylor, Richard R., *Splashdown* in New Scientist, July 25, 1998
- Taylor, Micolich, Jonas, *Fractal analysis of Pollock's drip paintings* in Nature, June 3, 1999, 399, 422(1999)
- http://materialscience.uoregon.edu/taylor/art/fractal_taylor.html

- http://plus.maths.org/issue6/turner2/index.html
- http://plus.maths.org/issue9/news/pollock
- http://www.discover.com/nov_01/featpollock.html
- http://perso.wanadoo.fr/charles.vassallo/en/art/intro.html
- http://math.bu.edu/DYSYS/chaos-game/node6.html
- http://www.weiheustephan.de/ane/dimensions/dimensions.html
- http://www.fch.vutbr.cz/lectures/imagesci/download/nummetfradim.pdf

E.7 Fractal Music

- "Fractal Music Lab" http://www.fractalmusiclab.com David Strohbeen 2000
- "Mandelbrot Music" http://www.fin.ne.jp/~yokubota/mandele.shtml Yo Kubota
- "Mandelbrot Music Program" http://www.fractovia.org/fractal_generators/screenshots/mmusic.html Fractovia.org 2002
- "Fractal Music Gallery" http://thinks.com/sounds/fractal.htm Thinks.com Ltd. 2003
- "MIDI Manufacturers Association" http://www.midi.org/ 2003 Midi Manufacturers Association Inc.

E.8 Art and Society

- Lomax, Song Structure and Social Structure
- Fischer, Art Styles as Cultural Cognitive Maps
- Albrecht, Barnett, Griff, The Sociology of Art and Literature - A Reader, Praeger Publishers, New York, 1970.
- Stein, M. (Editor), Jungian Analysis 2nd Edition, Open Court Publishing, Chicago, 1995
- Vygotsky, The Psychology of Art, Cambridge, Mass., M.I.T. Press [1971]
- Arnheim, R., New essays on the psychology of art, Berkeley: University of California Press, c1986.

- Brand-Clausen, Beyond reason: art and psychosis: works from the Prinzhorn Collection., London: Hayward Gallery ; Berkeley: University of California Press, c1996.
- Jung, The Archetypes and the Collective Unconscious, 1969, Bollingen Foundation, Princeton, NJ.

E.9 Making Art More Accessible

- "The Beauty of Fractals" Peitgen, Heinz-Otto, Springer-Verlag Publishing, New York, 1986
- "FrActivity" http://www.parkenet.org/jp/fractvty.htm Janet Parke 2002
- "Fractal Art – A Deliberate Approach" http://www.parkenet.org/jp/writing/ylem.html Janet Parke.
- "Fractalus" http://www.fractalus.com Damien Jones 2000
- "Janet Parke Interview" http://www.toray.co.jp/e/square/mag/interview/dca/interview1.html Toray Industries, Inc 2003
- "Toray Art Gallery: DCA 2002" http://www.toray.co.jp/artspace/e/dca_02/index.html Toray Industries, Inc 2002
- "Fractal Layering for Novices" http://www.fractovia.org/tutorials/layers01.shtml Fractovia.org 2003
- "Fractal Tutorials" http://fractalarts.com/ASF/tutorials.html Doug Harrington 2003
- "My Fractal Art Gallery" http://www.members.tripod.com/~NavaneethaKrishnan/fract/myfract.htm Navaneetha Krishnan

E.10 Fractal Art Psychology

- "The Sense of Order" E. H. Gombrich, Cornell University Press, Ithaca, New York, 1979
- "Art and Fractals" http://perso.wanadoo.fr/charles.vassallo/welcome.html Charles Vassallo 2002