Spring 2009

An In-Depth Look at Fractal Image Compression

Sarah Detty
Regis University

AN IN-DEPTH LOOK AT FRACTAL IMAGE COMPRESSION

A thesis submitted to
Regis College
The Honors Program
in partial fulfillment of the requirements
for Graduation with Honors

by

Sarah Detty

May 2009

Thesis written by

Sarah Detty

Approved by

_____

Thesis Advisor

_____

Thesis Reader

Accepted by

_____

Director, University Honors Program

REGIS UNIVERSITY

Regis College Honors Program
Honors Thesis

Authorization to Publish Student Work on WWW

I, _____, the undersigned student, in the
Print student name
Regis College Honors Program hereby authorize Regis University to publish through a Regis University owned and maintained web server, the document described below ("Work"). I acknowledge and understand that the Work will be freely available to all users of the World Wide Web under the condition that it can only be used for legitimate, non-commercial academic research and study. I understand that this restriction on use will be contained in a header note on the Regis University web site but will not be otherwise policed or enforced. I understand and acknowledge that under the Family Educational Rights and Privacy Act I have no obligation to release the Work to any party for any purpose. I am authorizing the release of the Work as a voluntary act without any coercion or restraint. On behalf of myself, my heirs, personal representatives and beneficiaries, I do hereby release Regis University, its officers, employees and agents from any claims, causes, causes of action, law suits, claims for injury, defamation, or other damage to me or my family arising out of or resulting from good faith compliance with the provisions of this authorization. This authorization shall be valid and in force until rescinded in writing.

Print Title of Document(s) to be published: _____

_____ _____

Student Signature **Date**

Check if applicable:

☐ The Work contains private or proprietary information of the following parties and their attached permission is required as well:

Complete if you do not wish to publish your work on the WWW:

☐ I do not authorize Regis University to publish my work on the WWW.

_____ _____

Student Signature **Date**

Regis University

Regis College Honors Program

Honors Thesis

Certification of Authorship for Honors Thesis

Print Student's Name_____

Telephone_____ Email_____

Date of Submission_____ Degree Program_____

Title of Submission_____

Submitted To _____

Certification of Authorship:
I hereby certify that I am the author of this document and that any assistance I received in its preparation is fully acknowledged and disclosed in the document. I have also cited all sources from which I obtained data, ideas or words that are copied directly or paraphrased in the document. Sources are properly credited according to accepted standards for professional publications. I also certify that this paper was prepared by me for the purpose of partial fulfillment of requirements for the Regis College Honors Program.

_____ _____

Student Signature *Date*

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr. Trenary for being so supportive during this entire thesis process and my reader, Dr. Seibert for his helpful suggestions and advice. Also, I would like to thank Dr. Bowie for his guidance throughout the past four years at Regis. I am so grateful for my friends and family, who have encouraged me to accomplish more than I ever thought possible.

I. INTRODUCTION

Walking along a sandy beach, have you ever wondered how long the coastline was? You probably haven't, and you probably wouldn't have guessed that its length approaches infinity. That doesn't seem to make sense, does it? How can a coastline be of infinite length? The answer has everything to do with fractals. In his 1988 book *Fractals Everywhere*, Michael Barnsley warns, "Fractal geometry will make you see everything differently. There is a danger in reading further. You risk loss of your childhood vision of clouds, forests, galaxies, leaves, feathers, flowers, rocks, mountains, torrents of water, carpets, bricks, and much else besides" (1). Fractals enable us to perceive and understand our world through a different perspective. Ordinary things that we take for granted, such as the nature surrounding us, are extraordinary fractals in the eyes of mathematicians. What is a fractal, you ask? Benoit B. Mandelbrot, the mathematician who coined the term "fractal," explains in his 1977 book, *Fractals: Form, Chance, and Dimension*, "*Fractal* comes from the Latin adjective *fractus*, which has the same root as *fraction* and *fragment* and means 'irregular or fragmented;' it is related to *frangere*, which means 'to break'" (4). The words *fraction* and *fragment* are familiar terms, bringing to mind the idea of something being *broken* or *divided*. However, in order to gain more insight into this mathematical concept, let us return to our sandy beach.

As we begin to relax to the sound of the waves crashing along the shore, we are enveloped by an overwhelming desire to measure the length of the shoreline. Let's also say that we happen to spot a beach vender selling yardsticks and rulers. After purchasing one of each, we set out on our mission to discover the length of the coastline. In order to make our measuring process more reasonable, we will assume that we are on a fairly small island. First we take the yardstick and begin measuring, placing the yardstick down, marking its end, and moving its other end to that point. However, we notice that the length of the yardstick misses some of the curvature of the shoreline. After taking down our measurement of the length of the shoreline using the yardstick, we proceed to measure the island again with the ruler. Recording the length of the shoreline, we also notice that this measurement is larger than the previous one. How is this possible? Each of the smaller curves that we were able to measure actually added length to our measurement. Therefore, the smaller our measuring tool, the longer our shoreline's measured length became. Even though the ruler was able to measure accurately some of the curves and crevices of the shore that the yardstick missed, both measurement devices are still missing details of the shoreline. By using an inch-long ruler, we would be able to obtain

Figure 1: Shoreline to be measured

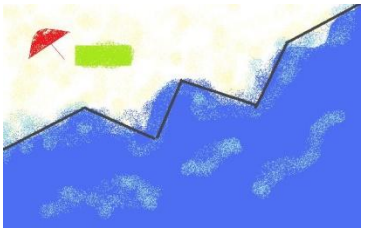


Figure 2: Measuring with yard stick

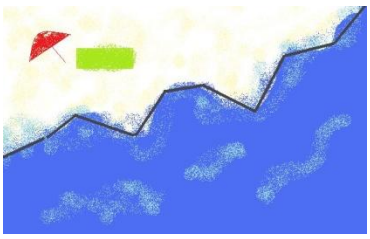


Figure 3: Measuring with a ruler

an even more accurate length, which would be larger than our previous measurements, but we would still be unable to measure even more minute curves with complete accuracy.

While we begin to understand this phenomenon on a small scale, magnifying this example will enable us to see even more implications. Mandelbrot further explains,

> It is indeed striking that when a bay or peninsula first noticed on a map scaled to 1/100,000 is reexamined on a map at 1/10,000, innumerable sub-bays and subpeninsulas become visible. On a 1/1,000 scale map, sub-sub-bays and sub-subpeninsulas appear, and so forth. (35)

Each time our view zooms in, we are able to notice coastline structures that were not apparent farther away, which add to the measured length of the coastline (Mandelbrot, *The Fractal Geometry of Nature* 26). Using a mile-long ruler, for example, would yield the same dilemmas that we experienced while measuring the smaller island's coastline. As we decreased the size of our measuring tool, we would find that our measurements of the coastline would increase towards infinity. Therefore, it is quite possible for a coastline to have infinite length, but perhaps, not in the way that we might expect. One might conceptualize infinite length as a line stretching outwards with no end. However, the coastline's infinite length stems from the seemingly never-ending amount of sub-bays and sub-peninsulas that increases its length as they become evident.

While the coastline example yields a more intuitive glimpse into the world of fractals, the Koch curve allows us to model the coastline in a more mathematical

manner[1]. The Koch curve was presented by a Swedish mathematician Helge von Koch in 1904 (Pietgen et al. 103). Like the coastline, it too is a fractal[2]. The construction of this curve is much easier to understand when visualized, but Mandelbrot gives a fairly straightforward description of its creation:

> The present construction begins with an 'inner island,' namely, a black equilateral triangle with sides of unit length. Then the points in the middle third of each side are displaced perpendicularly to the side in question, until they become aligned along a V-shaped peninsula, both sides of which are straight and of length 1/3. This second stage ends with a Star of David. The same process of formation of peninsulas is repeated with the Star's sides, and then again and again, ad infinitum. (36)

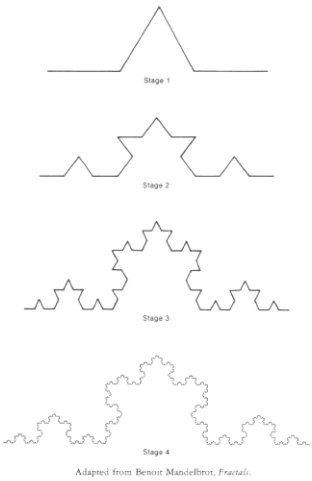


Figure 4: Construction of Koch Curve

After several iterations, the curve begins to resemble a coastline with its innumerous bays and sub-bays described by Mandelbrot[3]. If we were to zoom in on the curve, we would be able to detect its sub-sub-bays and sub-subpeninsulas as with the coastline example. Furthermore, if we wanted to measure the length of the Koch

[1] Even though we are choosing to model a coastline with this curve, it could also be used to model the outline of a snowflake or other similar fractals in nature (Pietgen et al. 104).
[2] "It is at best a first approximation, but not because it is too irregular, rather because, in comparison with a coastline, its irregularity is by far too systematic" (Mandelbrot 40).
[3] "One should mention why this and other plates choose to represent islands rather than coastlines and other 'solid areas' rather than their contours. This method's superiority is that it takes much fuller advantage of the fine resolution of our graphics system" (Mandelbrot 39). In other words, by coloring the interior of the coastline (i.e. the island), the intricate curves are easier to identify.

curve, we would discover that it too has infinite length. Each bay or peninsula that became visible would add length to the curve's previous measurements.

In the 1992 book *Fractals for the Classroom: Part One: Introduction to Fractals and Chaos*, Pietgen et al. highlight two important properties of the Koch curve:

> First of all – as the name already expresses – it is a curve, but this is not immediately clear from the construction. Secondly, this curve contains no straight lines or segments which are smooth in the sense that we could see them as a carefully bent line. Rather, this curve has as much complexity which we would see in a natural coastline, folds within folds within folds, and so on. (104)

Even though the initial iterations of the Koch curve appear to be just a collection of straight-lines, each following iteration demonstrates more clearly that these lines are actually curves. Like the coastline, the Koch curve is created from one intricate curve, whose subtleties are revealed as you move in closer. It is also important to note that the curve's structure is never completed; its iterations continue forever. In other words, you essentially could keep zooming in on the Koch curve and discover even more sub-bays and sub-sub-bays and so on, forever. Coastlines' features are finite; there is a certain point at which the sub-bays will stop appearing when you zoom in. Because of this, we should refine our previous assertion that coastlines have infinite length. While the length of a coastline is finite, the Koch curve allows us to model the fundamental attributes of a coastline, with a length that actually approaches infinity.

With the Koch curve in mind, we will explore more of the mathematical subtleties associated with fractals, specifically self-similarity. When we zoomed in on the

coastline, we noticed intricate sub-bays and sub-peninsulas that we weren't able to see from far away. Comparing the maps that were, for example, scaled to 1/100,000 versus 1/10,000, Mandelbrot notes,

> We find that although the different maps are very different in their specific details, they are of the same overall character and have the same generic features. In other words, it appears that the specific mechanisms that brought about small and large details of coastlines are geometrically identical except for scale. (35)

Sub-bays generally resemble sub-sub-bays, except that they are just of a different scale. In other words, they are *self-similar*. Most of us might have an intuitive definition of what self-similarity means, but let's try to refine this definition mathematically[4]. One only has to walk outside to examine a flower or tree to witness an instance of self-similarity (Pietgen et al. 153). Self-similarity abounds in nature, yet it can be difficult to define. Pietgen et al. explain, "Self-similarity extends one of the most fruitful notions of elementary geometry: similarity. Two objects are similar if they have the same shape, regardless of their size" (154). Therefore, in order for an object to be *self*-similar, it must consist of images similar to itself. For example, the coastline was *self-similar* because each sub-sub-bay was similar to a sub-bay within the coastline (i.e. to itself). Taking the Koch curve, we can zoom in on a branch of the "coastline" and compare these sections of the curve to the original branch. While there may be slight variations with the two images, it is clear that the Koch curve exhibits self-similarity. Even though self-

[4] "One would guess that the term [self-similarity] has been around for centuries, but it has not. It is only some 25 years old [as of 1992]" (Pietgen et al. 153).

similarity is a characteristic property of many fractals, it is important to note that simply displaying self-similarity does not automatically classify an object as a fractal. Pietgen et al. note, "Take, for example, a line segment, or a square, or a cube. Each can be broken into small copies which are obtained by similarity transformations. […] These structures, however, are not fractals" (230). Therefore, we must be careful to define a fractal solely on the basis of self-similarity.

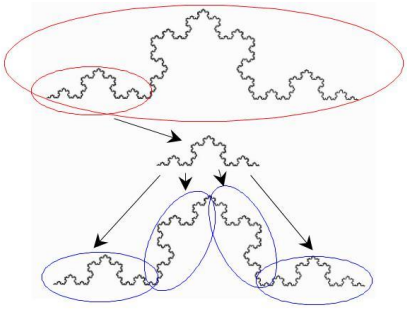


Figure 5: Self-similarity in the Koch curve

While self-similarity cannot categorize an object as being a fractal, fractal dimension can help us clarify our working definition of a fractal. The word 'dimension' probably brings to mind phrases, such as '1-D', '2-D', or '3-D.' However, have you ever imagined what '1.2618-D' would look like? Well, it is approximately the fractal dimension of the Koch Curve (Mandelbrot, *Fractals: Form, Chance, and Dimension* 42). How can an object have a non-integer dimension? Before we delve into this question, let's first examine how dimension is determined. We'll begin intuitively with the following recognizable mathematical objects that have dimensions of zero, one, two, and three, respectively: a point, a line segment, a square, and a cube. In the 2001 book *Understanding Analysis*, Stephen Abbott explains,

> Without attempting a formal definition of dimension (which there are several), we can nevertheless get a sense of how one might be defined by observing how the dimension affects the result of magnifying each particular set by a factor of 3. […] A single point undergoes no change at all, whereas a line segment triples in

length. For the square, magnifying each length by a factor of 3 results in a larger square that contains 9 copies of the original square. Finally, the magnified cube yields a cube that contains 27 copies of the original cube within its volume. Notice that, in each case, to compute the 'size' of the new set, the dimension appears as the exponent of the magnification factor. (77)

We can see that with each example, the "size" of the new set can be computed using the following equation:

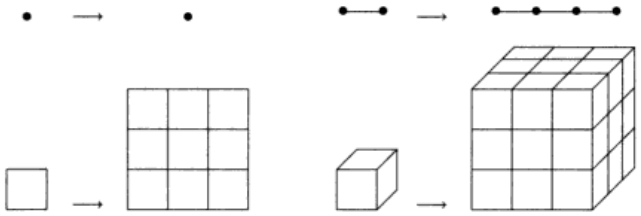


Figure 6: Magnifying by a factor of 3

$$(\text{Magnification factor})^{\text{dimension}} = \text{"Size" of the new set.}$$

For example, the line segment, with a dimension one, tripled when we magnified it by a factor of three, which means we had $3^1 = 3$. The same result is given with the square with dimension two when magnified by three: $3^2 = 9$. While we have been given the dimensions for the above objects and can intuit them easily, how can we determine the dimension objects, such as the Koch curve? The dimension of the curve is not immediately understood at first glance. In order to calculate the dimension, let's use the above formula and pick an arbitrary magnification factor of three. When we magnify the Koch curve, we see that the curve creates four new copies of itself. The above formula yields the following, substituting the variable *x* for dimension:

$$3^x = 4.$$

The next step is to solve for *x*, which means that we need to apply logarithms to both sides of the equation. This gives us:

$$x \ln 3 = \ln 4.$$

Divide both sides by ln 3 to get the resulting dimension:

$$x = \ln 4 / \ln 3 \approx 1.2618.$$

Therefore, we see that the dimension of the Koch curve is not an integer as with the more recognizable mathematical objects discussed earlier. Mandelbrot explains, "Every set with a noninteger D[imension] is a fractal[5]" (*The Fractal Geometry of Nature* 15). If an object has a non-integer dimension (i.e. a fractal dimension), it can be classified as a fractal.

Furthermore, it important to note that fractal dimension and self-similarity are intricately linked. Dr. Robert L. Devaney's Boston University faculty website defines fractal dimension by the following equation:

$$\text{Fractal dimension} = \frac{\text{Log (number of self-similar pieces)}}{\text{Log (magnification factor)}} \quad (2).$$

While determining the dimension of an object, we are calculating the number of self-similar pieces of the object, after applying a magnification factor. By claiming that a non-integer dimension automatically classifies an object as a fractal, we ignore the role of self-similarity within the object. Therefore, both properties of self-similarity and fractal dimension play a crucial role in determining whether an object is a fractal.

[5] "However, a fractal may have an integer D[imension]" (Mandelbrot, *The Fractal Geometry of Nature* 15). Mandelbrot classifies the "trail of Brownian motion" as a fractal with dimension 2, but for the purpose of this paper, we will exclude this example from our scope.

However, we needn't restrict ourselves to the Koch curve and coastline examples in order to demonstrate these fractal properties. One of the most famous and simple examples of a fractal is the Cantor set. Abbott discusses the method in which Georg Cantor created his infamous set of numbers:

> Let C_0 be the closed interval [0,1], and define C_1 to be the set that results when the open middle third is removed; that is
>
> $$C_1 = C_0 \setminus (1/3, 2/3) = [0, 1/3] \cup [2/3, 1].$$
>
> Now, construct C_2 in a similar way by removing the open middle third of each of the two components of C_1:
>
> $$C_2 = ([0, 1/9] \cup [2/9, 1/3]) \cup ([2/3, 7/9] \cup [8/9, 1]).$$
>
> […] It may be useful to understand *C* [the Cantor set] as the remainder of the interval [0, 1] after the iterative process of removing open middle thirds is taken to infinity. (75)

The Cantor set's construction is as simple and systematic as the construction of the Koch curve. After several iterations, the set begins to consist of smaller and smaller line segments, which eventually become just the end points of these infinitely many line segments. When we calculate the dimension of this set, Abbott notes, "Magnifying the Cantor set by a factor of 3 yields *two* copies of the original set. Thus, if *x* is the dimension of *C*, then *x* should satisfy $2 = 3^x$, or $x = \ln 2 / \ln 3 \approx .631$"

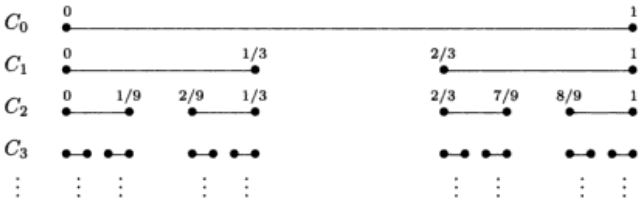


Figure 7: Construction of the Cantor Set

(77). Therefore, the Cantor set also has a non-integer dimension, on which Abbott comments, "The notion of a noninteger or fractional dimension is the impetus behind the term 'fractal,' coined in 1975 by Benoit Mandelbrot to describe a class of sets whose intricate structures have much in common with the Cantor set" (78). Furthermore, the Cantor set exhibits self-similarity. Each of the line segments created after several iterations is self-similar to the original line segment [0, 1]. Therefore, we see that the Cantor set, in addition to the Koch curve, can be classified as a fractal due to its fractal dimension and self-similarity.

Through our explorations with coastlines, the Koch curve, and the Cantor set, we have discovered two key properties of fractals: fractal dimension and self-similarity. Each of our examples displayed these characteristic properties of fractals, but some mathematicians argue that the definition of a fractal should be more ambiguous. In the 1995 book *Fractal Image Compression: Theory and Application,* Yuval Fisher offers a quote from Kenneth Falconer:

> My personal feeling is that the definition of a 'fractal' should be regarded in the same way as the biologist regards the definition of 'life.' There is no hard and fast definition, but just a list of properties characteristic of a living thing…In the same way, it seems best to regard a fractal as a set that has properties such as those listed below, rather than look for a precise definition which will almost certainly exclude some interesting cases. (25-26).

We have denoted the two properties of self-similarity and fractal dimension as being characteristic of fractals, but there are other properties that mathematicians may require

an object to posses in order to be considered a fractal. In the same way, by requiring too many properties, some unique "fractals" may be excluded. Mathematicians must find the right balance of properties in order to define a fractal, but as mentioned before, our focus will consist of the above mentioned properties.

Since we have determined our two key properties for an object to be classified as a fractal, one may ask, what does this have to do with image compression? How can fractals be used to compress the data contained within images? In his 1997 book *Fractal Imaging*, Ning Lu explains, "In summary, fractal image compression and representation is based on a strong belief – a belief that there are tremendous similarities and redundancies in most real-world images" (12-13). This thesis will attempt to explore how fractal image compression takes advantage of fractal properties in order to compress images.

II. INTRODUCTION TO FRACTAL IMAGE COMPRESSION

Let's return to our favorite sandy beach once more (this time, no more measuring!) Because we had such an enjoyable time discovering that the length of the coastline approaches infinity, we take a picture with our digital camera to remember the spot. After returning home, we upload our vacation pictures onto our computer. Whether or not we realize, we have just experienced image compression at its finest. Image compression is the process of encoding and decoding images so that they can be transmitted faster, more efficiently, and with the smallest loss of necessary data. There are many motivations for efficient methods of image compression, which Yuval Fisher explains in the 1995 book *Fractal Image Compression: Theory and Application,*

Figure 8: The beach

> Although the storage cost per bit is (in 1994) prices about half a millionth of a dollar, a family album with several hundred photos can cost more than a thousand dollars to store! This is one area where image compression can play an important role. Storing images in less memory cuts cost. (1)

While our single picture of the beach may seem like an insignificant amount of data to store and transmit, hundreds of photos can add up to a lot of memory. Memory is not as pricy as it was ten years ago, but cutting costs is still a motivation for companies and

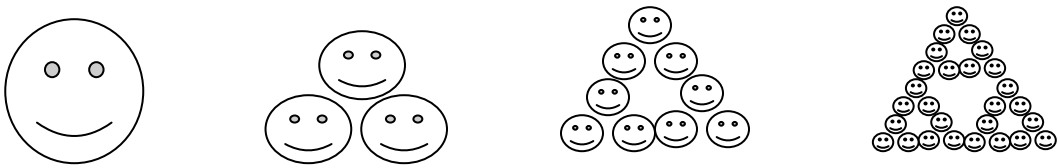
individuals. Fisher also notes, "Another useful feature of image compression is the rapid transmission of data; fewer data requires less time to send" (1). In addition to cutting costs, effective methods of image compression can diminish the amount of time to transmit and store images, which is perhaps more valuable to people today. While there are varying methods of image compression, this thesis will focus on fractal image compression.

Common to all forms of image compression is the desire to remove or reduce redundancies in the given image. Fisher writes, "Most images contain some amount of redundancy that can sometimes be removed when the image is stored and replaced when it is reconstructed" (1). If these image redundancies could be recreated through a sort of algorithmic or formulaic process, it would be inefficient to store these extra pieces of data. Therefore, image compression seeks to identify the redundancies and recreate the image without the original data associated with the redundancy. One might ask, can't we, the viewer of the image, tell that an image has been compressed and is missing the redundant data? Fisher explains, "Fortunately the human eye is insensitive to a wide variety of information loss. That is, an image can be changed in many ways that are either not detectable by the human eye or do not contribute to 'degradation' of the image" (1). If the image compression technique alters the image in a manner that the human eye cannot detect, then the viewer will have little perception of the compression done to the image. However, if the image compression is not efficient, the viewer will be able to tell that the image has been altered. A good image compression method would make the

compressed and decompressed image appear identical to the original image (or at least not noticeably different to human perception.)

Before we delve into fractal image compression, let's consider a special photocopying machine. This photocopying machine "reduces the image to be copied by a half and reproduces it three times on the copy" (Fisher 2). If we continue to feed the output image as input, we see that "the copies seem to be converging to the same final image […] We also see that this final image is not changed by the process, and since it is formed of three reduced copies of itself, I must have detail at every scale – it is a fractal" (Fisher 2).

Figure 9: The special copying machine with an input image and its resulting output image (above) and several iterations later (below).



The iterative process of the special photocopying machine may seem familiar (think back to the Koch curve or the Cantor set.) However, we are now using images to create fractals! The resulting image created by this process would have a fractal dimension and be self-similar. Fisher explains,

> We call this image the *attractor* for this copying machine. Because the copying machine reduces the input image, the copies of any initial image will be reduced

to a point as we repeatedly feed the output back as input; there will be more and more copies, but each copy gets smaller and smaller. So, the initial image doesn't affect the final attractor; in fact, it is only the position and orientation of the copies that determines what the final image will look like. (F2)

Therefore, if the input image were twice the size of our original image, for example, the copy machine would generate the same resulting image, i.e. the same *attractor*.

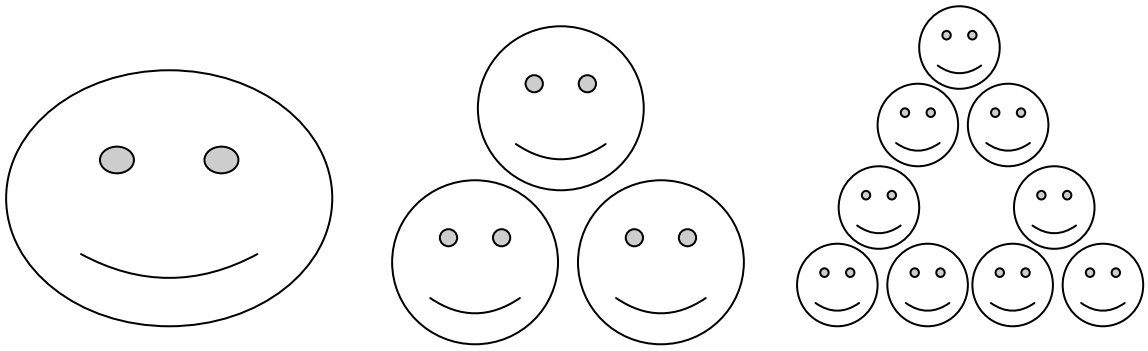


Figure 10: Starting with twice as large input image – same attractor results, but just takes longer for it to reach a smaller size.

Only if we changed the position and orientation of the copies would our final image change: for example, if we decided to rotate the input image before photocopying.

Figure 11: Rotating the initial image produces a similar result, but with a different orientation.

What does all of this photocopying have to do with fractal image compression? Before we answer that question, we must acquire the necessary terminology. We noticed in the above example that while the size of the original input image did not alter the

attractor, the position and orientation of the copies did. These transformations of the new input images are called *affine transformations*, and "each can skew, stretch, rotate, scale, and translate an input image" (Fisher 3). These simple transformations have a significant effect on the resulting *attractor* of the photocopying machine example. For example, when we rotated the smiley face input image in Figure 10, our attractor also was rotated. This rotation was an affine transformation. Why are we so interested in such seemingly simple transformations? Michael Barnsley, the pioneer of fractal image compression, writes in the introduction to Ning Lu's book *Fractal Imaging*,

> Why affine transformations? Simply because they feature in computer graphics, in human vision, in motion compensation and image segmentation, they are of low complexity, and because…real world images contain significant affine redundancy. (xix)

Since affine transformations are the most prevalent and versatile transformations in "real world images," it makes sense that image compression utilizes them. However, the transformations in the photocopying example are not only affine, but they are also *contractive*. Fisher notes, "Different transformations lead to different attractors, with the technical limitation that the transformations must be contractive – that is, a given transformation applied to any two points in the input image must bring them closer together in the copy. (2) Each time the input image is fed back into the copying machine, its resulting output displays each original input image as closer to one another. If the transformation of the copying machine was not contractive, the output image would show the original input images getting farther and farther apart.

Upon examining the photocopying example, we have noticed that a fractal was created using both affine and contractive transformations. The resulting image, i.e. the *attractor* of the photocopying machine, was determined by the position and orientation of the copies (the transformations) rather than the size of the original input images. Fisher analyzes this result, explaining, "Since the final result of running the copy machine in a feedback loop is determined by the way the input image is transformed, we only describe these transformations" (2). In other words, our resulting image can be described using only the nature of the transformations of the input image. Barnsley noticed this same phenomenon when he attempted to recreate an image of a fern. Lu explains this process by allowing the reader to witness the power of affine transformations:

Figure 12: Barnsley's fern

> Start with a piece of blank paper with an (x, y)-coordinate system marked and pick an arbitrary point on the paper; then find its coordinates. Randomly select one of the four affine transformations listed below:
>
> w_1: $(x, y) \mapsto (0.85 \cdot x + 0.04 \cdot y, -0.04 \cdot x + 0.85 \cdot y + 40)$
>
> w_2: $(x, y) \mapsto (0.20 \cdot x - 0.26 \cdot y, 0.23 \cdot x + 0.22 + y + 40)$
>
> w_3: $(x, y) \mapsto (-0.15 \cdot x + 0.28 \cdot y, 0.26 \cdot x + 0.24 \cdot y + 11)$
>
> w_4: $(x, y) \mapsto (0, 0.16 \cdot y)$.
>
> Then apply the transformation to this point, and the coordinates of a new point are obtained. Notice the new point. Plot it in black on paper. Again select randomly

> one of the above transformations and apply it to this point to obtain the next new point… [etc.] If you are patient and persistent enough, gradually the Iterated Systems trademark, a spleenwort fern, will appear on the paper like magic. (2)

Lu is right to attribute the creation of fern to magic. It is almost unbelievable that the iterative process of randomly choosing of an affine transformation from the list can generate such a complex natural image. However, Lu does not stop there. He continues to intrigue us, explaining,

> The process above shows that with the right mathematical model a perfect picture can be described with infinitely fine and marvelously rich textures in only 24 numbers:
>
> 85, 4, 0, -4, 85, 40; 20, -26, 0, 23, 22, 40; -15, 28, 0, 26, 24, 11; 0, 0, 0, 0, 16, 0.
>
> These four affine transformations form an *iterated function system (IFS)*. The fern image created is an example of a *fractal*, which mathematically is the *attractor* of this IFS. (2)

This process of creating the fern seems to echo our photocopying machine example. We notice here that the size of the input image is not what determines the *attractor*, but rather the affine transformations. From this iterative generation of the fern from four affine transformations, Barnsley "suggested that perhaps storing images as collections of transformations could lead to image compression" (Fisher 3). This would lead to a dramatic reduction of data needed to be stored in order to compress the image and then to later generate the same image. Fisher explains,

> His [Barnsley's] argument went as follows: the fern looks complicated and intricate, yet it is generated from only four affine transformations. Each affine transformation w_i is defined by six numbers, a_i, b_i, c_i, d_i, e_i and f_i, which do not require much memory to store on a computer (they can be stored in 4 transformations x 6 numbers per transformation x 32 bits per number = 768 bits). Storing the image of the fern as a collection of pixels, however, requires much more memory (at least 65,536 bits for the resolution shown…). So if we wish to store a picture of a fern, we can do it by storing the numbers that define the affine transformations and simply generating the fern whenever we want to see it. Now suppose that we were given any arbitrary image, say a face. If a small number of affine transformations could generate that face, then it too could be stored compactly. (3)

By identifying the affine transformations that create a given image, the storing of just those numbers that represent the transformations would enable us to generate that image again at will. This is the theory motivating fractal image compression, which was ultimately developed by Barnsley and his associates.

III. HOW DOES FRACTAL IMAGE COMPRESSION WORK?

In the last chapter, we explored some of the basic ideas behind fractal image compression. Here, we finally delve into the technical algorithms that are implemented for this compression technique. Let's first return to the concept of *iterated function systems* (IFS's). Fisher explains, "IFS's served as the motivating concept behind the development of fractal image compression, and most of the work on fractal image compression is based on IFS's or their generalization" (28). In other words, in order to understand how fractal image compression works, we must first understand iterated function systems. On the website of the Waterloo Fractal Coding and Analysis Group, E. R. Vrscay, an Applied Mathematics professor at the University of Waterloo, et al write, "IFS is the term originally devised by Michael Barnsley and Steven Demko for a collection of contractive mappings over a complete metric space, typically compact subsets of R^n" (1). Intuitively, an iterated function system is a collection of transformations that "contract" the points on a given plane. For example, the special copying machine from the previous chapter is an iterated function system. Each time an output image is inputted into the machine, the resulting image is "contracted." Therefore, we already have worked with iterated systems without even knowing it: not only with the copying machine, but also when we created the fern. We were utilizing contractive functions to map each point of the fern. The iterative process of applying those functions yielded an iterated function system.

However, we must be careful to equate iterated functions systems with fractal image compression. Welstead warns, "IFS's are, at best, only a crude form of image compression. It should be stressed that IFS's in their original form…are not the basis of current approaches of fractal image compression" (11). While IFS's can help us to understand the fundamentals of fractal image compression, we must realize that they are only an inspirational concept. Current methods of fractal image compression are not based purely on iterated function systems. Furthermore, on the limitations of iterated function systems, Welstead writes,

> Simple IFS's apply only to images that are *self-similar*, that is, images that are made up of subimages that are copies of the entire image. Notice that each leaf of the fern is actually a copy of the entire fern. This is not true of arbitrary images. In general, we can only hope to find subimages that are copies of other subimages. (11)

In other words, iterated function systems, like the ones we have been using, are very basic and cannot be applied to images lacking self-similarity, a concept discussed in the introduction of this paper. Both the fern and the special copying machine created images that were made up of subimages of themselves. Therefore, the fern and the copy machine images were self-similar, i.e. fractals. However, fractal image compression must be able to work with all images if it is to be effective and its use

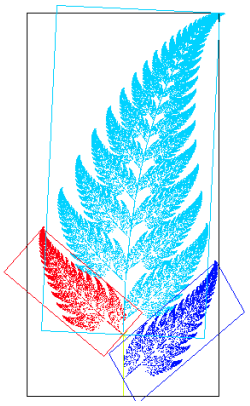


Figure 13: Self-similarity in Barnsley's fern

widespread, which is why iterated function systems are only its inspiration, not its fundamental algorithm.

With this in mind, let us explore another aspect of iterated function systems. In the last chapter, we mentioned that iterated function systems are collections of *affine transformations*. These are simple transformations, such as a 90^{o} rotation or a diagonal flip. Fisher alludes to this idea while he explains the nature of self-similarity in images:

> A typical image of a face, for example [Figure 14], does not contain the type of self-similarity found in the fractals [such as Figure 15]. The image does not appear to contain affine transformations of itself. But, in fact, this image does contain a different sort of self-similarity. [Figure 16] shows sample regions of Lenna that are similar at different scales: a portion of her shoulder overlaps a smaller region that is almost identical, and a portion of the reflection of the hat in the mirror is similar (after transformation) to a smaller part of her hat. The difference is that in [Figure 15] the image was formed of copies of its *whole* self (under the appropriate affine transformation), while here the image will be formed of properly transformed *parts* of

Figure 14: Lenna - a standard image

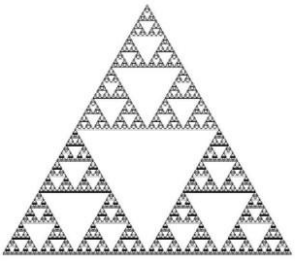


Figure 15: Sierpinski Gasket - a classic fractal

Figure 16: Self-similarity in Lenna

itself. (9)

In other words, arbitrary images, such as faces, do not exhibit the kind of self-similar found in fractal images, such as Barnsley's fern or the Sierpinski Gasket. Like Welstead, Fisher emphasizes the importance of finding subimages that are copies of other subimages (i.e., subimages that are similar to other subimages.) Affine transformations can be applied to these subimages in order to map them to their respective similar subimage. In this way, iterative function systems of affine transformations can be implemented for images that do not exhibit natural self-similarity. Fisher explains, "This restricted self-similarity is the redundancy that fractal image compression schemes attempt to eliminate" (10). Therefore, fractal image compression takes advantage of the self-similarity exhibited by the subimages to reduce data redundancy.

As we begin to analyze the algorithm for fractal image compression, it will be helpful to break up the process into two separate routines: encoding and decoding. Welstead gives a general explanation of this process:

> A digital image is an array of pixel values, which we can think of as a list of numbers. The compression problem consists of two main parts: encoding and decoding. Encoding attempts to represent the original list of numbers in a different way, hopefully requiring less storage than the original list. Decoding tries to recover something like the original image from the encoded image. (3)

Our discussion thus far has emphasized the encoding part of compression – reducing the redundant data found in images by utilizing self-similarity. However, decoding is the other essential process of any image compression technique. If one cannot decode and

reproduce the given image, the compression technique is useless. Furthermore, Welstead states, "If the decoded image is exactly the same as the original image, then the encoding-decoding algorithm is said to be a *lossless* algorithm. If the decoded image differs from the original image, then the algorithm is a *lossy* algorithm" (3). According to Welstead, fractal image compression is a lossy compression algorithm (9). Iterated function systems, at best, can only approximate images using affine transformations. Therefore, the resulting image is an approximation, not a replication.

While decoding is an important step, let's first understand how the encoding process works. Lu gives a six-step algorithm, which will be a good starting place:

1. *Input the original image*. Read the original image into the designated buffer.
2. *Create a reference region list*. Prepare the list of all possible reference regions for a destination region to match. […]
3. *Initialize destination region list*. […]
4. *Search for fractal match*. Given a destination region, loop over all possible reference regions to find the best match using a given metric. This is the computationally most involved step of the algorithm.
5. *Select fractal elements*. […]
6. *Pack the fractal codes*. Store the fractal parameters into a file by further lossless packaging using some entropy coding methods. (65)

At first glance, this seems like a very complicated algorithm. However, let's break it down into manageable pieces so that we can understand it better. The first step is fairly self-explanatory: input the given image so that the fractal image compression software can access it. The next two steps require some definitions to interpret their meanings. A *reference region* is a partitioned block of the image that will be mapped to a *destination region* using a contractive affine transformation. Reference regions are also called

domain blocks or *domain cells*[6]. A *destination region* is a partitioned block of the image to which a reference region gets mapped. Destination regions are also referred to as *range blocks* or *range cells*[7].

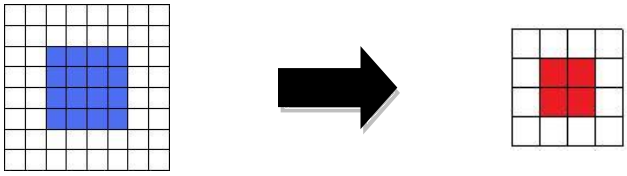


Figure 17: A domain block being mapped to a range block, using a contractive affine transformation. (Blue is used to denote a domain block. Red is used to denote a range block).

Therefore, the second and third steps of the algorithm require that we create a list of all the possible reference regions and destination regions for the image. This is done by partitioning the image into equal-sized regions (usually squares.) There are many ways of doing this[8], and Fisher comments on this, stating, "Partitioning schemes come in as many varieties as ice cream" (18). However, we will use the most popular method, quadtree partitioning, which will be explained in detail later (Lu 92).

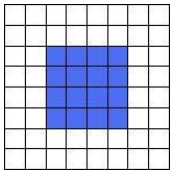
Once we have a list of both the domain blocks and range blocks, we are ready to move on to the next step: *search for fractal match*. In this step of the algorithm, the software starts with the first range block in the list of range blocks and tries to find a matching domain block, by applying a contractive affine transformation to the domain

[6] This is the terminology that I prefer.
[7] This is the terminology I prefer.
[8] Partitioning schemes can be classified as either image-independent or image dependent (Lu 91). The first is simple to implement, using geometric shapes (squares or rectangles), but it is difficult to "take into account regional difficulties and natural connections between areas" (Lu 91). The second lacks any "satisfactory algorithm to segment an image" this way (Lu 92). Lu references M.C. Escher's artwork as an inspiration for this type of partitioning scheme.

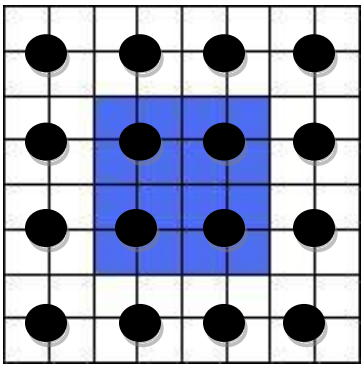
block. In the 2006 book *Introduction to Data Compression*, Khalid Sayood lists the eight possible affine transformations for a square (i.e. the given domain block):

1. Rotation by 90 degrees
2. Rotation by 180 degrees
3. Rotation by -90 degrees
4. Reflection about midvertical axis
5. Reflection about midhorizontal axis
6. Reflection about diagonal
7. Reflection about cross diagonal
8. Identity mapping (562-63)

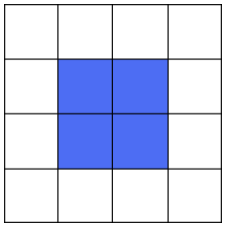
However, before one of these transformations is applied, the domain block must be shrunk down to the same size as the range block. Otherwise, it will be difficult to determine if there is a good match between the two blocks. While there are many ways that this can be done, we will examine a simplistic method, which we will demonstrate on the following domain block:



Next, let's examine the block in groups of four pixels (we will zoom in to get a better view):

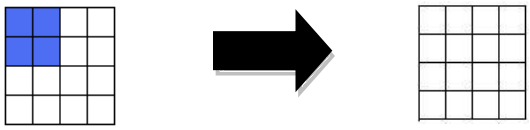


Each group of four pixels has a black circle in the center denoting the group. For each group, if there is a colored pixel (a blue pixel is technically a black pixel), then the group of four cells becomes a single colored pixel. If there is no colored pixel in the group, the group of four becomes one white pixel. Therefore, the above domain block shrinks to the following:



Now that the domain block is shrunk to the same size as the given range block, an affine transformation can be applied, and then a meaningful comparison can be made between the two blocks.

For this specific domain block, the identity transformation is used. We now compare the domain result on a "pixel by pixel basis to the range cell to determine the quality of the fit" or match (Welstead 55). A predetermined error tolerance is used to judge if the match is adequate. Again, there are various methods of calculating an error tolerance. We will use a simplistic method that calculates the ratio of the number of matching black pixels in the two blocks to the total number of black pixels in the block. For example, if we use an error tolerance of 0.50, we will not accept a match unless more than 50% of the black pixels match. In the above example, we have a ratio of 1, so the match is accepted[9]. The following is an example of a match that does not fall within the given error tolerance, with a ratio of 0:



If a match does not meet the error tolerance, such as the example above, then a different affine transformation is applied to the shrunken domain block. If after all of the affine transformations have been applied and the resulting matches have not met the error tolerance, then the next domain block is chosen to repeat the process.

If a match is found that meets the error tolerance, then the software records the data associated with the domain block and the transformation that was applied. Welstead explains,

> The 'code' for a fractal-encoded image is a list consisting of information for each range cell, including the location of the range cell, the domain (usually indentified

[9] Granted, while this error tolerance is unrealistic, it will help simplify the example that will be presented later in the chapter.

by an index), that maps onto that range cell, and parameters that describe the transformation mapping the domain onto the range. (50)

This step must be done for all range blocks, so one can begin to see why this is a computationally intensive step. After we have determined all of the transformations to be applied to the domain cells to achieve a contractive mapping to a range cell, we must store these "fractal codes." These codes will enable us to decompress the image, similar to the iterated function systems that we looked at earlier. The final step of Lu's algorithm is to store these codes even more efficiently and compactly using entropy encoding[10].

After walking through Lu's algorithm, we now have a slightly better understanding of what is happening during the encoding process, but there are several details that need to be filled in. First, let's explore how quadtree partitioning is applied to an image. Fisher explains, "A quadtree partition is a representation of an image as a tree in which each node, corresponding to a square portion of the image, contains four subnodes, corresponding to the four quadrants of the square. The root of the tree is the initial image" (55). Figure 18 depicts the tree-like nature of an image (the large gray square) being quadtree partitioned into four blocks of equal size. Then a second quadtree partition is applied to one of the four small blocks (the smaller gray square), dividing it into four equally-sized blocks. Essentially, a quadtree partition divides an image or a block into four equal parts.

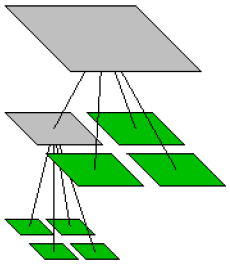


Figure 18: Quadtree partitioning - an image as a tree

[10] See chapter 12 in Nu's book for more information on how this step works. It is not essential to understanding the fundamental algorithm for fractal image compression.

Therefore, each time a quadtree partition is applied in our algorithm, we can expect the given block or cell to be divided into fourths.

Welstead gives an alternate algorithm for the encoding process that specifically applies quadtree partitioning:

> For each range cell, the algorithm tries to find the domain and corresponding contractive transformation that best cover the range cell. In order to produce contractive transformations, range cells larger than the largest domain cells are subdivided into smaller range cells. Contrast and brightness are adjusted for the best fit through a least-squares process. If the fit is within the error threshold, then that range cell is considered to be covered, and the algorithm moves on to the next range cell. If the fit is not within threshold, the algorithm checks to see if the maximum quadtree depth has been reached. If it has, processing stops for that range cell, and the range cell is considered covered with the best domain and transformation available. If the maximum quadtree depth has not been reached, then the algorithm subdivides that cell into 4 smaller range cells, and the search for optimal domains and transformations begins anew with these 4 new range cells. Processing is complete when all range cells have been covered, either by finding a domain-transformation fit within the error tolerance, or by reaching the maximum quadtree depth. (W 51-52)

This algorithm is more complicated than Lu's general one, but it will help us understand how the quadtree partition factors into the encoding process. It may help to look at the flow diagram provided by Welstead[11] that accompanies the algorithm (49).

[11] It is almost identical to Welstead's flow diagram, but it doesn't include a "search for best domain." For the sake of simplicity, we will accept the first match that fits within the error tolerance.

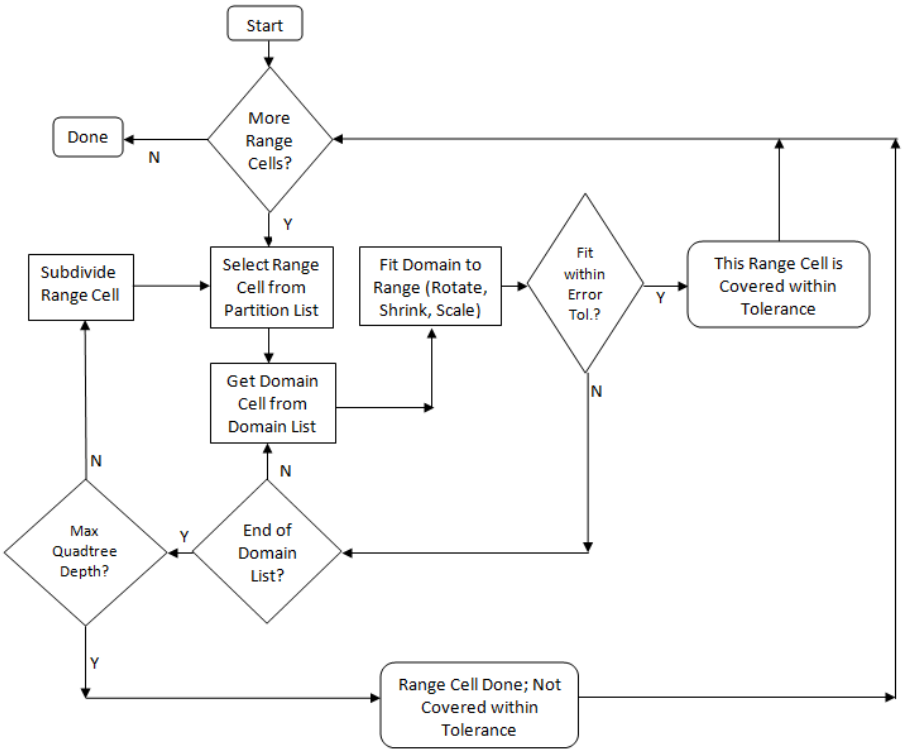


Figure 19: Welstead's flow diagram

First, let's examine what Welstead means by a maximum quadtree depth. Each time the quadtree partition is applied constitutes a depth level for the image (represented using a tree structure.) If we look back to Figure 18, we can see that it has a quadtree depth of 2. Therefore, a maximum quadtree depth is a predetermined limit to the number of quadtree partitions that can be applied to the image. With this in mind, let us return to the algorithm. When we begin traversing through the flow diagram, we already have accomplished several steps:

1. Determined the maximum quadtree depth and error tolerance[12].
2. Read the image to be compressed.

[12] "Note that using a smaller error threshold leads to more range cells, and using a larger quadtree depth also leads to more range cells. […] In either case, more range cells means poorer compression (and sometimes no compression at all), but usually better decoded image quality" (Welstead 52).

3. Created domain and range block lists based on the maximum quadtree depth.

We have discussed in detail how the first two steps are to be accomplished, but let's examine how to go about implementing the third step. Fisher explains that the ranges "are compared with domains from the domain library (or domain pool) **D**, which are twice the range size" (55-56). Therefore, each domain block is twice[13] as large as each range block. For example, if you have an 8 x 8 pixel range block, you would have 16 x 16 pixel domain blocks. This will enable us to form a list of all domain and range blocks, according to their sizes. If we are given a maximum quadtree depth, the smallest range block will be on that depth level. This means that the smallest domain block will be twice the size of the smallest range blocks (no smaller domain blocks are necessary.) With this information, we can create a domain library, listing all of the possible domains, as well as a list of all possible ranges.

Once we have our lists of domain and range blocks, we are ready to begin Welstead's algorithm. The first step is to check if there are more range cells that need to be covered by a domain cell. Since we are just starting the algorithm, we have an entire list of range cells to cover. Next, we select the first range cell from the list and the first domain cell from the domain library. We then shrink the domain cell and apply a transformation as we did with Lu's algorithm. If the match is within the error tolerance, we store the transformation and domain information and move on to the next range cell. If the match does not fall within the error tolerance, we check to see if there are any more domain cells in the list. When there are more domain cells, we pick another domain cell

[13] To be exact, each domain block is actually four times as large as each range block.

and try to find a suitable transformation. However, if there are no domain cells that match within the error tolerance and we have gone through the entire list, we then must check to see if we have reached the maximum quadtree depth. We apply the quadtree partition to the range cell, dividing it into four smaller range cells if the maximum quadtree depth has not been reached, and continue with the algorithm using the first of the smaller range cells. On the other hand, if the depth has been reached, we consider the range cell to be covered, despite not being within the error tolerance, and move on to the next range cell. This process repeats until we have covered the entire list of range cells.

After stepping through Lu's six-step algorithm and Welstead's quadtree partitioning algorithm for the encoding process, everything makes perfect sense, right? If you're still lost, it's okay. In order to obtain a better grasp of this algorithm, the following section gives a step-by-step example that applies the quadtree partitioning scheme to the encoding process:

Algorithm Walk-Through:

1. Input the image to be compressed. This is an extremely simplified example that we will use to demonstrate the algorithm's finer details.

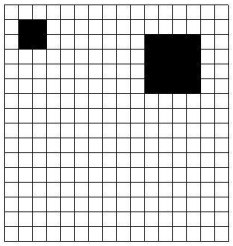


Figure 20: Image to be compressed

2. Define maximum quadtree depth = 3

3. Error tolerance[14] = 0.50

4. Create domain list

Domain Library

(Blue is used to designate a domain cell.)

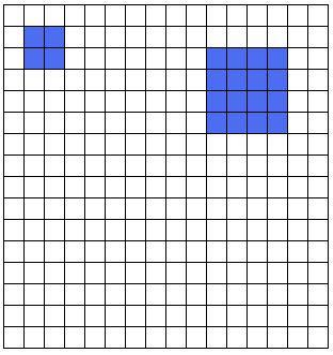


Figure 21: 16 x 16 domain cells

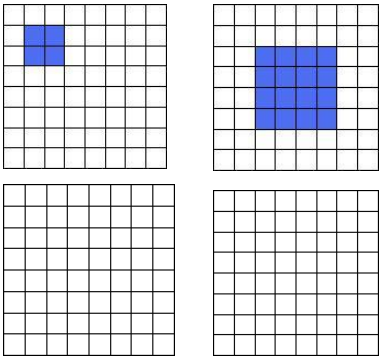


Figure 22: 8 x 8 domain cells

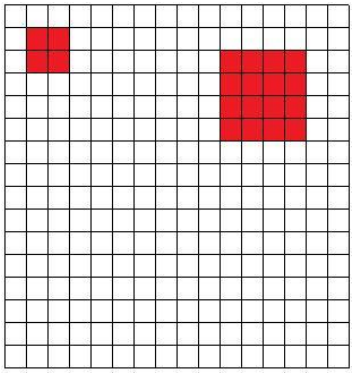
[14] We will calculate whether a domain/range cell match is below the threshold, using the metric defined in the above section.

5. Create a list of all possible range cells.

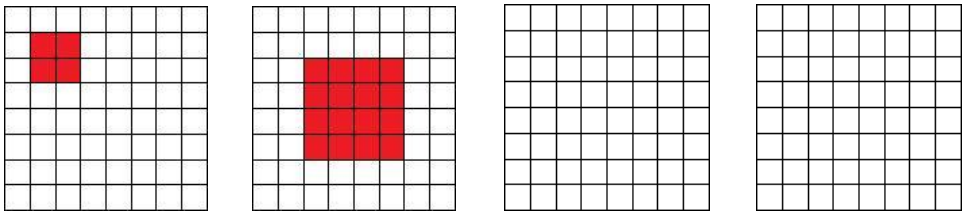
Range Cells

(Red is used to designate a range cell.)

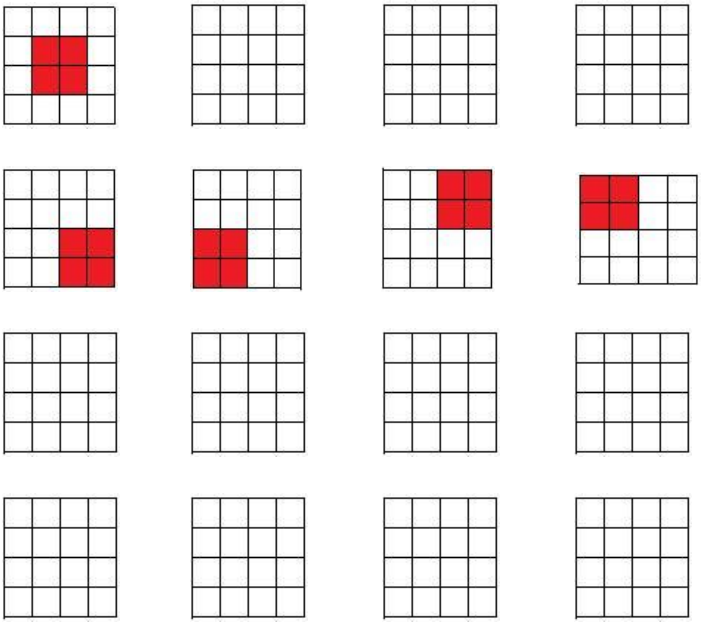
16 x 16 Ranges



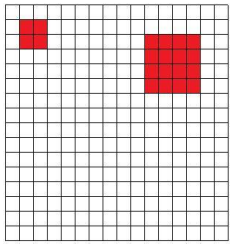
8 x 8 Ranges



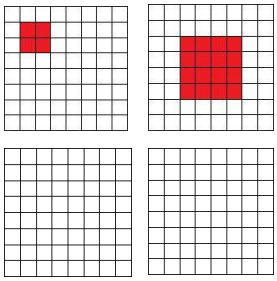
4 x 4 Ranges



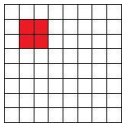
6. Begin with range cell (1,0,0)[15] (Original image)



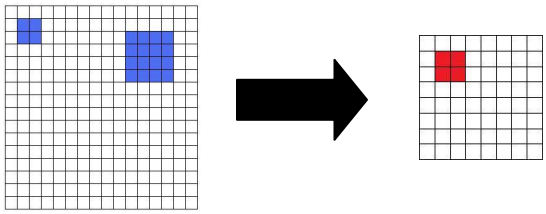
   a. Since there is no domain cell to match with it[16], we must partition the range cell using the quadtree method.
      i. Divide range cell into 4 equal pieces.



7. Take range cell (1,1,0) and continue with the process.

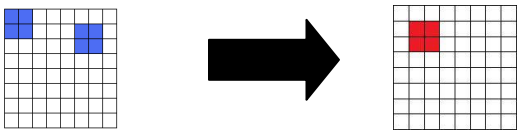


   a. Compare range cell with larger domain cells[17] to find an acceptable affine transformation.



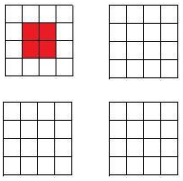
      i. Shrink domain cell using method described in above section and compare.

[15] Range cells are indexed beginning with the left-hand corner and proceeding clockwise.
[16] Since the first range cell is the original image, there are no domain cells that are larger than it, which is required in order to achieve a contractive mapping.
[17] There is only one larger domain cell – the original image.



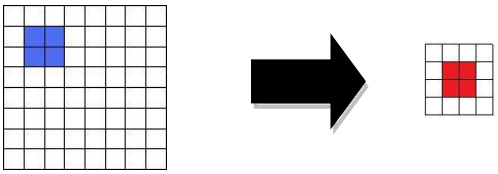
1. There are no transformations that can be applied to the domain cell to make it match the range cell.

ii. Therefore, we must apply the quadtree partition to the range cell.

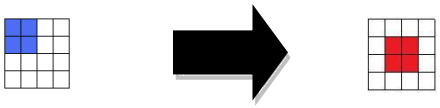


8. Take range cell (1,1,1) and continue with the process

a. Compare with larger domain cells to find an acceptable affine transformation.

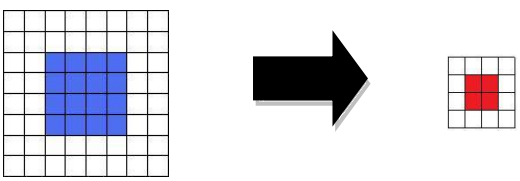


i. Shrink domain cell and compare.

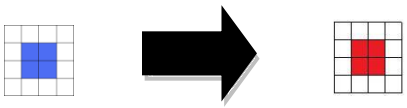


1. No transformations can be applied to match the domain cell to the range cell.

ii. Take next domain cell and compare.



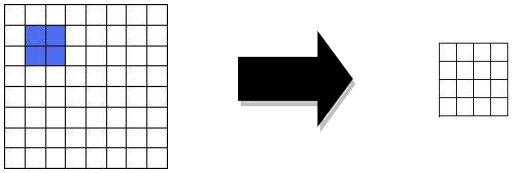
iii. Shrink domain cell and compare.



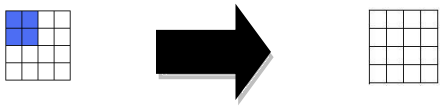
1. If we apply the identity transformation, we have a match.
2. Since the match is within the error tolerance, store domain block and transformation data.

9. Take range cell (1,1,2) and continue with the process

a. Compare with larger domain cells.

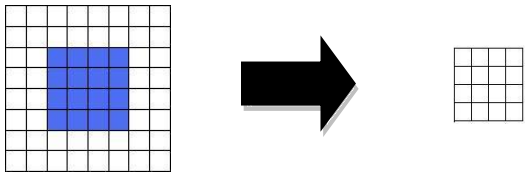


i. Shrink domain in order to make comparison.



1. No transformation can give us a match within the error tolerance.

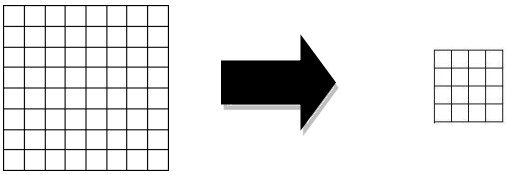
ii. Take next domain cell and compare.



iii. Shrink domain in order to make comparison.

1. No transformation can give us a match within the error tolerance.

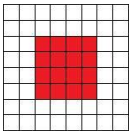
iv. Take next domain cell and compare.



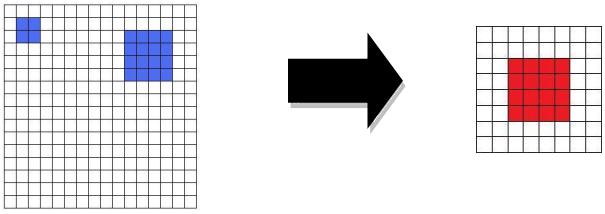
v. Shrink domain in order to make a comparison.

1. If we apply the identity transformation, we have a match.
2. Since the match is within the error tolerance, store domain block and transformation data.

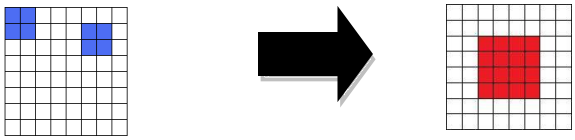
10. Take range cell (1,1,3) and continue with this process.
    a. Same match as range cell (1,1,2)
11. Take range cell (1,1,4) and continue with this process.
    a. Same match as range cell (1,1,2)
12. Take range cell (1,2,0) and continue with this process.



a. Compare range cell with larger domains.

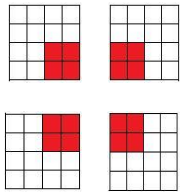


i. Shrink domain cell in order to compare.



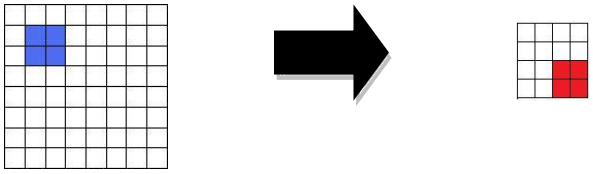
1. There are no transformations that can be applied to the domain cell to make it match the range cell.

ii. Therefore, we must apply the quadtree partition to the range cell.

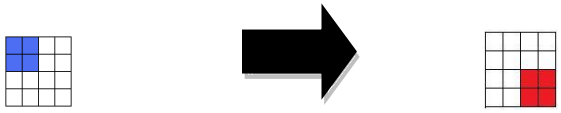


13. Take range cell (1,2,1) and continue with the process.

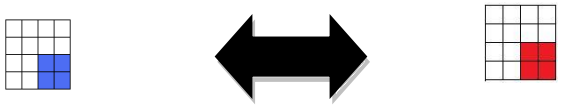
a. Compare with larger domains.



b. Shrink domain cell in order to make comparison.



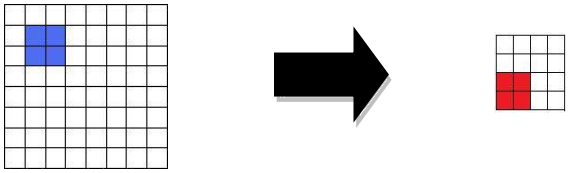
c. Transformation: Rotate 180 degrees



1. If we rotate the domain cell 180 degrees, we have a match.
2. Since the match is within the error tolerance, store domain block and transformation data.

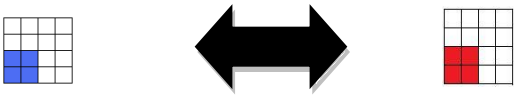
14. Take range cell (1,2,2) and continue.

a. Compare to larger domains



b. Shrink domain cell in order to make comparison.

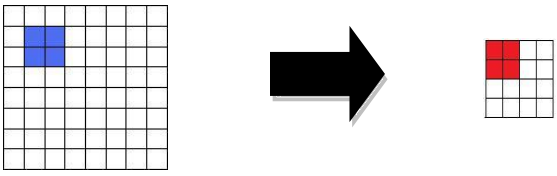
c. Transformation: Rotate -90 degrees



1. If we rotate the domain cell -90 degrees, we have a match.
2. Since the match is within the error tolerance, store domain block and transformation data.

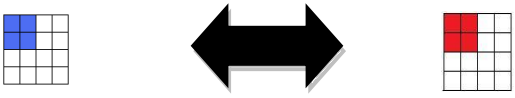
15. Take range cell (1,2,3) and continue.

a. Compare range cell with larger domains.



b. Shrink domain cell in order to make a comparison.

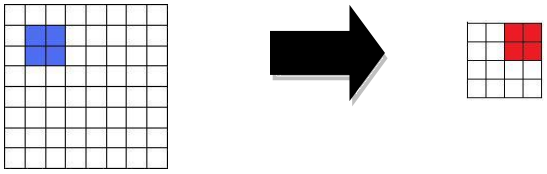
c. Transformation: Identity



1. If we apply the identity transformation, we have a match.
2. Since the match is within the error tolerance, store domain block and transformation data.

16. Take range cell (1,2,4) and continue.

a. Compare with larger domains

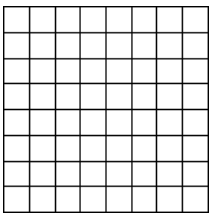


b. Shrink domain cell in order to make a comparison.

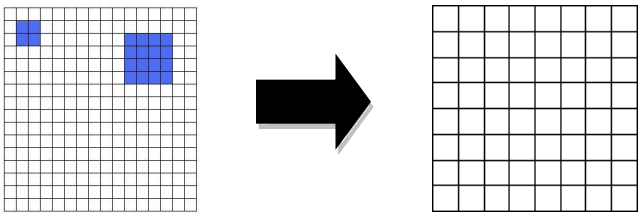
c. Transformation: Rotate 90 degrees

1. If we rotate the domain cell 90 degrees, we have a match.
2. Since the match is within the error tolerance, store domain block and transformation data.

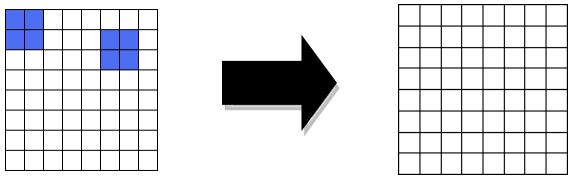
17. Take range cell (1,3,0) and continue.



a. Compare to larger domains.

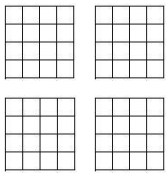


b. Shrink domain cell in order to make comparison.



   i. No transformations can be applied to match the domain cell to the range cell within the error tolerance.
   ii. Therefore, we must apply the quadtree partition to the range cell.

c. Divide the range cell into four smaller range cells.



d. Take range cell (1,3,1) and compare with domain cells.
   i. Same match as range cell (1,1,2).
e. Take range cell (1,3,2) and compare with domain cells.
   i. Same match as range cell (1,1,2).
f. Take range cell (1,3,3) and compare with domain cells.
   i. Same match as range cell (1,1,2).
g. Take range cell (1,3,4) and compare with domain cells.
   i. Same match as range cell (1,1,2).

18. Take range cell (1,4,0) and continue with the process.
    a. Same matches as range cell (1,3,0).
19. Since all of the range cells are covered, the algorithm is complete.

The following table details which domain blocks were mapped to a given range block:

Domain Block	Range Block	Transformation
1	(1,2,1)	180 degrees
	(1,2,2)	270 degrees or -90 degrees
	(1,2,3)	Identity
	(1,2,4)	90 degrees
2	(1,1,1)	Identity
3	(1,1,2)	Identity
	(1,1,3)	Identity
	(1,1,4)	Identity
	(1,3,1)	Identity
	(1,3,2)	Identity
	(1,3,3)	Identity
	(1,3,4)	Identity
	(1,4,1)	Identity
	(1,4,2)	Identity
	(1,4,3)	Identity
	(1,4,4)	Identity
4	N/A	N/A

From this example, it is easy to see that the encoding algorithm for fractal image compression is very computationally intensive. The above example is one of the simplest images that could be contrived in order to demonstrate this process. Therefore, one can imagine how involved an actual image would be[18].

Thus far, we have discussed the encoding process in great detail. The decoding process is very similar to encoding but differs in a few ways: simplicity, speed, and order.

[18] Our example is not even a grey-scale image, which would complicate the algorithm further. Color images add even more complications.

Essentially, the algorithm is a backwards version of the encoding one. Lu gives a four step algorithm:

1. *Input the fractal codes*. Read and unpack the code file of the fractal model.
2. *Do decompression iteration* about 8 to 16 times. At each iteration, we process one destination region at a time, by mapping the resampled and rescaled reference region, as instructed by the fractal code, to the destination region.
3. *Render the decoded image* (optional). […]
4. *Output the decoded image* in some desired pixel format… (66)

Intuitively, the decoding process is very similar to how we created the fern using four functions. The fractal codes are the coefficients for the equations, and by iterating through the list of function, the image becomes clearer. The third step of the algorithm enables the image to be adjusted for irregularities, such as block edges or blockiness (Lu 66). The final product is the recreated approximation of the image.

IV. CONCLUSION: THE FUTURE OF FRACTAL IMAGE COMPRESSION

After examining the algorithm for fractal image, some drawbacks become apparent. First, the encoding process alone can be very time intensive. Welstead comments, "Encoding times of over two days on a workstation have been reported for cases where the number of domains exceeds 100,000" (56). This situation does not seem unreasonable if one considers the extreme simplicity of the example presented in the last chapter. Our example used a 16 x 16 pixel image, which is only a minute fraction of a typical image to be compressed. There were five possible domain cells, so one can imagine how this number could grow exponentially due to the size of an image. The algorithm's painful lack of speed has led to its negative reputation, granted, for good reason. In a 2008 article, "Introduction to Fractal Compression," John Kominek, a professor at the University of Waterloo, comments on the algorithm's speed, specifically before it was completely automated:

> Barnsley's patent has come to be derisively referred to as the 'graduate student algorithm.'
>
> Graduate Student Algorithm[19]:
>
> - Acquire a graduate student.
> - Give the student a picture.

[19] Kominek also comments that "it was one of Barnsley's PhD students [Arnaud Jacquin] that made the graduate student algorithm obsolete" by completely automating the process (2).

- And a room with a graphics workstation.
- Lock the door.
- Wait until the student has reverse engineered the picture.
- Open the door. (2)

Even though software can be programmed to implement the entire algorithm, one can see that it is still a painfully slow process. However, Sayood comments, "The fractal approach has one significant advantage: decoding is simple and fast. This makes it especially useful in applications where compression is performed once and decompression is performed many times" (568). Therefore, if the encoding process could be sped up, there is a possibility more people would utilize this compression technique[20].

Over the years, there have been various attempts at speeding up the encoding process. One of these attempts has focused on classifying the domain cells while creating the domain pool list. Lu explains,

> The method of speeding up a searching or sorting algorithm is as follows: to classify reference objects [domain cells] into groups and to compare each target object [range cell] with only one of the groups that has the highest possibility of finding a good match. Thus to have an optimum speed, the number of objects in a group need to be small, and consequently the number of groups becomes inevitably large. (174)

[20] Lu agrees with Welstead, writing, "The unbearably slow rate of early fractal compression algorithms has been the main reason that many applications use alternative methods, even though the compression time seems less critical than the decompression time because an image compressed once can be used forever" (174).

In other words, by pre-classifying the domains into groups, the process of matching domain and range cells can be sped up by comparing a range cell to a group of domain cells that are most likely to be a good match, thereby drastically reducing the number of overall comparisons. However, as Lu points out, this method is a delicate balancing act. While it would be more efficient to compare the range cell to small groups, the number of groups will explode. However, if the groups are too large, then once we pick a matching domain cell group for the given range cell, we will have to traverse through the large group, which may be just as slow as when we didn't classify the domain cells[21].

Other areas of research dedicated to speeding up fractal image compression also allude to a delicate balance of elements. Welstead writes, "The process of setting up a system of domains is a balancing act between providing a large enough domain set for good range match possibilities and keeping the domain set small enough so that searched can be performed in a reasonable amount of time" (50-51). Too few domains will make it harder to find an adequate range match, whereas too many domains will take forever to search through the list and find a match. On a similar note, Welstead also explains,

> Good compression depends on the ability to find a good domain-range match without having to subdivide the range. Too much range subdivision leads to too many ranges, which hurts the compression ratio (and in fact can lead to "image expansion" rather than compression if one is not careful!) (50).

[21] While there are several methods detailed about classifying domain cells, I was not able to find any research concerning the classification of similar range cells. This seems like it would also speed up the matching process.

In the same way that too many domains can slow down the search for an acceptable domain-range match, too many ranges can hurt the compression quality. Fisher also writes, "For a fixed image, more transformations lead to better fidelity but worse compression" (19). While a greater number of ranges will allow us to find better quality domain-range matches leading to a truer resulting image, our compression will suffer. The more we subdivide the image, the less we are actually *compressing*, which, at some point, will counterbalance our goal of reducing data redundancy.

While these optimizations continue to be researched, there has been a shift towards developing new image compression techniques based on the advantages of fractal image compression. One of the most recent advances focuses on wavelets. In the 1995 article, "An Introduction to Wavelets," Amara Graps explains, "Wavelets are mathematical functions that cut up data into different frequency components, and then study each component with a resolution matched to its scale" (1). What does this even mean, and how do wavelets have anything to do fractal image compression? Graps writes,

> Wavelets are functions that satisfy certain mathematical requirements and are used in representing data or other functions. This idea is not new. Approximation using superposition of functions has existed since the early 1800's, when Joseph Fourier discovered that he could superimpose sines and cosines to represent other functions. However, in wavelet analysis, the *scale* that we use to look at the data plays a special role. Wavelet algorithms process data at different *scales* or *resolutions*. (1)

In other words, wavelets are approximating functions that are especially concerned with scales. Fractals have everything to do with scales, specifically being defined as having detail at all scales (Fisher 26). Wavelets consist of basis functions, which vary "in scale by chopping up the same function or data space using different scale sizes" (Graps 4). These basis functions approximate the given function. Wavelet transforms, the fundamental operation for approximating a function, "do not have a single set of basis functions like the Fourier transform, which utilizes just the sine and cosine functions. Instead wavelet transforms have an infinite set of possible basis functions" (Graps 6). Herein lies the powerful nature of wavelets. This infinite set of basis functions allows us to approximate any given function. The following is the mathematical structure of a wavelet basis function:

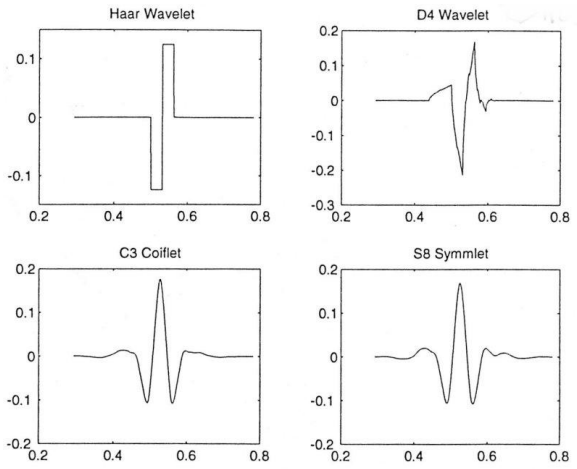


Figure 23: Common wavelets

$$\varphi_{(s,l)}(x) = 2^{-s/2}\,\varphi(2^{-s}x - l) \qquad \text{(Graps 8)}$$

Graps examines this structure, writing, "The variables *s* and *l* are integers that scale and dilate the mother function φ to generate wavelets… The scale index *s* indicates the wavelet's width, and the location index *l* gives its position" (8). From this mathematical equation, it is apparent that the wavelet basis function is essentially an affine transformation, like the ones we examined earlier concerning iterated function systems and fractals.

When wavelets are utilized in image compression, the coefficients of the wavelet basis functions are used in the very same way as the fractal codes in fractal image compression. Welstead explains the general algorithm used in wavelet compression techniques:

> One applies a wavelet transform to an image and then removes some of the coefficient data from the transformed image. Encoding may be applied to the remaining coefficients. The compressed image is reconstructed by decoding the coefficients, if necessary, and applying the inverse transform to the result. (93)

From this explanation, one can see its resemblance to how fractal image compression works. However, one difference between the two is that wavelet compression throws out some of the coefficients of the wavelet bases, whereas fractal image compression retains all contractive affine transformation coefficients[22].

While wavelet compression techniques are still being researched and implemented, there is also a movement towards fractal-wavelet hybrid methods. Hubbard writes, "Wavelets seem particularly well suited to the job of understanding fractals, or multifractals, often characterized by self-similarity at different scales" (61). Since wavelets and fractals share similar features, it makes sense that combining their strengths may lead to more efficient and powerful compression techniques. In the 2003 article "Wavelets and Fractal Image Compression Based on Their Self-Similarity of the Space-Frequency Plane of Images, Yoshito Ueno explains, "The fractal encoding method requires large amount of computation to process images. After the wavelet transformation

[22] This is additionally compressed using entropy coding, as explained in the last chapter.

of images, we can utilize the self-similarity among sub-bands on the wavelet space in order to reduce the amount of computation for the fractal encoding" (394). By applying the approximation power of wavelets to fractal image compression, the encoding speed can be dramatically reduced. Furthermore, the self-similarity that is exploited by fractal image compression can enhance the compression for wavelet methods[23].

Most recently, JPEG2000, a newly released compression standard based on JPEG and wavelets, looks like a promising advancement for image compression. Before we explore JPEG2000, let's look at the basics of JPEG. Sayood explains,

> The JPEG standard is one of the most widely known standards for lossy image compression. It is a result of the collaboration of the International Standards Organization (ISO), which is a private organization, and what was the CCITT (now ITU-T), a part of the United Nations. The approach recommended by JPE is a transform coding approach using the DCT [Discrete Cosine Transform]. (410)

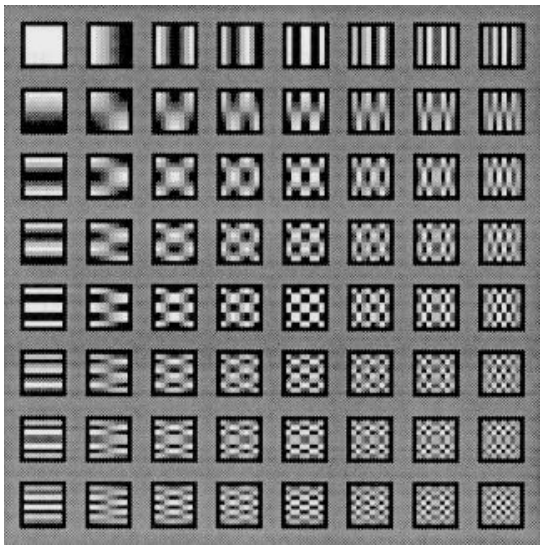


Figure 24: The basis matrices for the DCT

[23] Lu also comments, "The advantages of hybridizing the wavelet and the fractal is not just for pure compression performance. In fact, there are two important features that are strongly demanded in today's digital imaging market: *progressiveness*, which is requested by the tremendous market of the World Wide Web, and *resolution independence*, which is a key requirement for desktop publishing. […] The wavelet subband coding technology has a natural progressive structure that has been proven by Said and Pearlman [SP]. And the fractal image representation gives a powerful resolution-independent image representation that serves for multi-usage" (256).

JPEG has become a household name, being recognized as a file format for images. As a standard in the image compression world, its use is understandably widespread. The DCT's coefficients are the key to reducing data redundancy. Sayood writes, "The discrete cosine transform (DCT) gets its name from the fact that the rows of the N x N transform matrix **C** are obtained as a function of cosines" (402). The figure at the left displays the basis matrices of the DCT, which are used to approximate an image. Notice that as you move further down and to the right of the matrices, the bases have higher rates of change (between black and white.) Once the bases have been applied to approximate the image, the high-frequency coefficients essentially are thrown out because the human eye does not detect such extreme changes in a relatively small pixel (Sayood 412). Huffman coding is used to further compress the coefficients of the basis matrices and the resulting data is used to decompress the image in the future[24]. Now that we have a very basic understanding of JPEG, let's see how wavelets are applied in JPEG2000. Sayood writes, "The JPEG2000 standard, which is designed to update and replace the current JPEG standard, will use wavelets instead of the DCT to perform decomposition of the image" (494). In other words, wavelet basis functions, instead of the DCT basis matrices, will be used to approximate images. Huffman coding and the rest of the steps following the DCT transform would be used after that initial step.

[24] This explanation of JPEG is not intended to be thorough by any means, but is only meant to be a very general introduction to its general concepts and algorithm. Please refer to Sayood's book or other resources listed in the works cited of this thesis for more details about JPEG.

With so many different compression techniques to choose from, how is a person to rate one method above another? In the image compression world, there are a collection of images that are designated for such research purposes. Lu explains,

> The most commonly used testing image in digital image compression is *Lena*, the picture of Swedish born *Lenna Sjooblom*, that first appeared in *Playboy*, November 1972. This legendary face with a Mona Lisa smile can be found in almost every digital imaging journal, particularly in the past decade [as of 1997]. […] Scientists chose this image because it is easy but not trivial. Furthermore, for a researcher to spend long hours in front of a testing image, this picture is a pleasant one. (76)

Testing images, such as Lena, are chosen because of their esthetic nature and also the subtle challenges presented to image compression techniques. Lu also writes, "For scientists using common testing images, the advantage is clear: If you are familiar with these images you are able to judge more intuitively research results illustrated in these images and better comprehend the statistics gathered from these images" (76). In other words, designated sample images allow everyone to be on the same page for the research studies. It is much easier to determine whether a compression technique is more effective than another if one is familiar with the image at question. Furthermore, Welstead explains,

> Error measurement is an important aspect of determining the effectiveness of an image compression scheme. Obviously, we want to know how far off the decoded image is from the original image. Because perception of image quality is

> subjective, the question of how to measure this difference is not an easy one to answer. […] PSNR [Peak Signal-to-Noise Ratio] is a standard measure in the image compression world.[25] […] Error measure is an area of active research in image compression. Human perception of error does not always coincide with absolute measures of error. (58-59)

Despite using designated test images and having standard metrics to judge the effectiveness of an image compression technique, researchers are likely to disagree over their findings from time to time. Some people may not notice certain blemishes, such as blockiness, whereas others may be more sensitive to such distortions. Ongoing research hopes to eliminate this subjective uncertainty over compression methods.

Despite the drawbacks of fractal image compression, such as its encoding time, and the establishment of JPEG and JPEG2000 as standards for image compression, one may begin to wonder why more research should be committed towards this compression technique. Regarding this idea, Welstead comments, "Rather than condemning fractal encoding, […] there is need for further research to unlock its potential" (164). Fractal image compression is a fairly new concept in the image compression world and has not had the amount of research devoted to it as JPEG. Welstead also writes, "New technologies such as fractals and wavelets should not be viewed as competitors but allies in establishing new standards" (163). As the need for digital images increases, more efficient image compression techniques are required. Fractal image compression allows researchers to view image compression in a whole new light, utilizing the self-similarity

[25] The exact calculation is beyond the scope of this thesis. To learn more about this topic, see Welstead's book or other references in the works cited.

of images and subimages in order to reduce data redundancy. With further optimization research, there is a possibility that fractal image compression can enhance the JPEG standard (which has already been done in some sense with the application of wavelets). Fisher writes, "While the fractal scheme currently has more of a cult following than the respectful attention of the average engineer, today's standards and fashions become tomorrow's detritus, and at least some of today's new ideas flower into popular use" (23). Researchers must be open to explore questions presented by fractal image compression and other novel ideas if they are to continue to advance their field.

V. BIBLIOGRAPHY

Abbott, Stephen. Understanding Analysis. New York: Springer, 2001.

Barnsley, Michael. Fractals Everywhere. Boston: Academic Press, 1988.

Barnsley, Michael F. and Lyman P. Hurd. Fractal Image Compression. Wellesley: AK Peters, 1993.

Barnsley, Michael F., Dietmar Saupe, and Edward R. Vrscay, eds. Fractals in Multimedia. New York: Springer-Verlag New York, 2002.

Devaney, Robert L. "Fractal Dimension." BU Math. 1995. <http://math.bu.edu/DYSYS/chaos-game/node6.html>.

Fisher, Yuval, ed. Fractal Image Compression: Theory and Application. New York: Springer Verlag New York, 1995.

Graps, Amara. "An Introduction to Wavelets." 12 Feb. 2009 <http://www.amara.com/IEEEwave/IEEEwavelet.html>

Helmberg, Gilbert. Getting Acquainted with Fractals. New York: Walter de Gruyter GmbH & Co., 2007.

Hubbard, Barbara Burke. The World According to Wavelets: The Story of a Mathematical Technique in the Making. 2nd edition. Natick, MA: A K Peters, 1998.

Kominek, John. "Introduction to Fractal Compression." 29 Jan. 2009. <http://www.faqs.org/faqs/compression-faq/part2/section-8.html>

Lu, Ning. Fractal Imaging. Boston: Academic Press, 1997.

Mandelbrot, Benoit B. Fractals: Form, Chance, and Dimension. San Francisco: W. H. Freeman and Company, 1977.

---.The Fractal Geometry of Nature. New York: W. H. Freeman and Company, 1982.

Peitgen, Heinz-Otto, Hartmut Jürgens, and Dietmar Saupe. Fractals for the Classroom: Part One: Introduction to Fractals and Chaos. New York: Springer-Verlag New York, 1992.

Sayood, Khalid. Introduction to Data Compression. 3rd edition. Boston: Morgan Kaufmann Publishers, 2006.

Ueno, Yoshito. "Wavelets and Fractal Image Compression Based on Their Self Similarity of the Space-Frequency Plane of Images." International Journal of Wavelets, Multiresolution and Information Processing. 1.4 (2003): 393-405. Academic Search Premier. EBSCOhost. Dayton Memorial Library, Denver, CO. 14 Jan. 2009 <http://search.epnet.com>

Vrscay, Edward R. "A Hitchhiker's Guide to 'Fractal-Based' Function Approximation and Image Compression." University of Waterloo. 1995. <http://links.uwaterloo.ca>.

Welstead, Stephen. Fractal and Wavelet Image Compression Techniques. Bellingham, WA: SPIE Optical Engineering Press, 1999.

IMAGES:

Bourke, Paul. Fern. 15 Mar. 2009 <http://images.google.com/imgres?imgurl=http://local.wasp.uwa.edu.au/~pbourke/fractals/fracintro/fern.gif&imgrefurl=http://local.wasp.uwa.edu.au/~pbourke/fractals/fracintro/&usg=__ye36ofzcRm19Icw1wrkuDS22Vdo=&h=591&w=300&sz=34&hl=en&start=1&um=1&tbnid=erpwWnMVHHxkiM:&tbnh=135&tbnw=69&prev=/images%3Fq%3Diterated%2Bsystems%2Bfern%26ndsp%3D18%26hl%3Den%26rlz%3D1T4TSHB_enUS276US283%26sa%3DN%26um%3D1>

Canon iR 4579. 15 Mar. 2009 <http://images.google.com/imgres?imgurl=http://www.officetoday.com.my/catalog/images/copier/ir4570%2520copy.jpg&imgrefurl=http://www.officetoday.com.my/catalog/COPIER-MACHINES/Canon-iR-4570/&usg=__XoGBgXvkMQ4aqAoBhzP-5sld2vQ=&h=295&w=295&sz=31&hl=en&start=3&um=1&tbnid=GhWgPs15ad0fjM:&tbnh=115&tbnw=115&prev=/images%3Fq%3Dcopying%2Bmachine%26hl%3Den%26rlz%3D1T4TSHB_enUS276US283%26um%3D1>

DCT Basis Matrices. 15 Mar. 2009 <http://www.cs.cf.ac.uk/Dave/Multimedia/DCT.jpg>

Fractal Fern Using IFS. 15 Mar. 2009 <http://images.google.com/imgres?imgurl=http://upload.wikimedia.org/wikipedia/commons/thumb/4/4b/Fractal_fern_explained.png/150px-Fractal_fern_explained.png&imgrefurl=http://en.wikipedia.org/wiki/Iterated_function_system&usg=__I_tf1BgOfy6Cfq6m2P2U9yGVtQM=&h=209&w=150&sz=30&hl=en&start=30&um=1&tbnid=MEUlCya2vH2OhM:&tbnh=106&tbnw=76&prev=/images%3Fq%3Dself-similarity%2Bin%2BBarnsley%2527s%2Bfern%26ndsp%3D18%26hl%3Den%2

6rlz%3D1T4TSHB_enUS276US283%26sa%3DN%26start%3D18%26um%3D1
>

Lenna. 15 Mar. 2009.
<http://images.google.com/imgres?imgurl=http://photoshopnews.com/wp-content/uploads/2007/04/lenna-lg.jpg&imgrefurl=http://photoshopnews.com/2007/04/24/geek-love-the-lenna-story/&usg=__gGVGynA36ZoTWlelHwbLs-c7f00=&h=480&w=480&sz=51&hl=en&start=1&um=1&tbnid=Nt4uryWpDkcynM:&tbnh=129&tbnw=129&prev=/images%3Fq%3Dlenna%26hl%3Den%26rlz%3D1T4TSHB_enUS276US283%26sa%3DN%26um%3D1>

Lenna's Self-similarity. 15 Mar. 2009
<http://images.google.com/imgres?imgurl=http://classes.yale.edu/Fractals/Panorama/ManuFractals/ImageCompression/Lena2.gif&imgrefurl=http://classes.yale.edu/Fractals/Panorama/ManuFractals/ImageCompression/ImageCompression.html&usg=__hYuiYijiEIgJ2re-VctILcCaSVo=&h=256&w=256&sz=55&hl=en&start=2&um=1&tbnid=CgJNmyC0Yi4EAM:&tbnh=111&tbnw=111&prev=/images%3Fq%3Dself-similarity%2Bin%2BLenna%26hl%3Den%26rlz%3D1T4TSHB_enUS276US283%26um%3D1>

Mandelbrot, Benoit. Koch Curve. 15 Mar. 2009
<http://images.google.com/imgres?imgurl=http://abyss.uoregon.edu/~js/images/koch_curve.gif&imgrefurl=http://abyss.uoregon.edu/~js/cosmo/lectures/lec18.html&usg=___VXJd0v6Tb0D05MGp90HkQ5amZw=&h=946&w=678&sz=26&hl=en&start=1&um=1&tbnid=KMHQmTPgEXrbwM:&tbnh=148&tbnw=106&prev=/images%3Fq%3Dkoch%2Bcurve%26hl%3Den%26rlz%3D1T4TSHB_enUS276US283%26sa%3DN%26um%3D1>

---. Self-similarity in the Koch Curve. 15 Mar. 2009
<http://images.google.com/imgres?imgurl=http://elaine98.byus.net/fig004.JPG&imgrefurl=http://elaine98.byus.net/page03.htm&usg=__baIyxX06O1A5QrJvk0pNTuzRpAY=&h=435&w=594&sz=30&hl=en&start=5&um=1&tbnid=x4Oto8sQdGnUWM:&tbnh=99&tbnw=135&prev=/images%3Fq%3Dself-similarity%2Bin%2Bthe%2BKoch%2Bcurve%26hl%3Den%26rlz%3D1T4TSHB_enUS276US283%26um%3D1>

Stout, Quentin et al. A Quadtree Decomposition. 15 Mar. 2009
<http://images.google.com/imgres?imgurl=http://www.eecs.umich.edu/~qstout/pap/SC97quadtree.gif&imgrefurl=http://www.eecs.umich.edu/~qstout/pap/SC97.html&usg=__myvepJ5poYFG2TU2OFIh3YKNM3M=&h=490&w=600&sz=5&hl=en&start=12&um=1&tbnid=cQ7axU2P92IQJM:&tbnh=110&tbnw=135&prev=/

images%3Fq%3Dquadtree%2Bpartition%26hl%3Den%26rlz%3D1T4TSHB_enUS276US283%26sa%3DN%26um%3D1>

Toon 0111. 15 Mar. 2009
<http://www.bg.ic.ac.uk/Staff/khparker/homepage/BSc_lectures/2002/orthogonal_wavelets.jpg>

Touhey, Kyle. Sierpinski Gasket. 15 Mar. 2009
<http://images.google.com/imgres?imgurl=http://www.math.ubc.ca/~cass/courses/m308-02b/projects/touhey/lined-sierpinski.JPG&imgrefurl=http://www.math.ubc.ca/~cass/courses/m308-02b/projects/touhey/index.html&usg=__KVJbhs0Q6FZj65Vam91vAje0XYA=&h=282&w=305&sz=20&hl=en&start=11&um=1&tbnid=Y6S-_i6IKjXY5M:&tbnh=107&tbnw=116&prev=/images%3Fq%3Dsierpinski%2Bgasket%26hl%3Den%26rlz%3D1T4TSHB_enUS276US283%26sa%3DX%26um%3D1>