BRIDGES
Mathematical Connections
in Art, Music, and Science

Harmony, Chromatics, and Chaos

Michael Field
Department of Mathematics
University of Houston
Houston, TX 77204-3476
E-mail: mf@uh.edu
URL: nothung.math.uh.edu/~mike/

Abstract

The aim of these notes is to provide an introduction to the idea of *symmetric chaos* and to some of the questions that arise in using symmetric chaos in design. In particular, we discuss the problem of symmetrically coloring designs, more especially repeating patterns with 2-color symmetries.

1. Introduction

Mathematics has long been intertwined with art – especially the fine arts – and design. This relationship is most frequently seen through the use of concepts based on *geometry*, *symmetry* and, more recently, *topology*. The use of symmetry and geometric proportion can give harmony and unity to a design. Indeed, this is often the way in which the designer or artist conveys unity and balance. This is seen in architectural design, or in the intricate repeating patterns of Islamic art, or in the design of car wheels and modern textiles. In the music of Bach, symmetry and proportion are woven into the fabric of the score. Sometimes the presence of symmetry and proportion can be quite subtle. Thus, the works of Brent Collins attract our attention not only because of their complex geometry and topology but also because of the local symmetries that are *implicit* in that geometry. In general, our perceptions are strongly influenced by symmetry and proportion. When we confront a complex image or metaphor, our first grasp at comprehension is often an attempt to fit the subject within a symmetric framework.

Yet, geometry and symmetry have a static quality about them. We view the facets of a diamond or crystal from ever changing positions but the diamond itself, its geometry and symmetry, remain fixed in time and space. This emphasis on stasis is reminiscent of the medieval or classical view of the world. The idea that the geometry of the solar system or universe is fixed and ever repeating is seen in the classical Greek geometric models of the solar system and in Kepler's famous 1595 model based on regular solids and spheres [5]. While our contemporary world view is still influenced by these ideas, it has been profoundly changed by the Newtonian dynamic view of the universe. Instead of static geometry and symmetry, everything is dynamic and changing, nothing sure or immutable or exactly repeating – notwithstanding Nietzsche. Further, at the atomic level, quantum theory tells us that we can know nothing with complete certainty. Paradoxically, the apparent certainty that we perceive and feel, our very conscious self, may be but an average over the random fluctuations and chaos that occur at the quantum level of matter [6].

In this paper, we describe an approach to art and design that is based on chaos, inexactitude and symmetry. Like any system in art or music, there are constraints. But within those constraints there is the possibility of inventiveness and originality.

The word 'harmony' in the title is intended to suggest the way in which the presence of symmetry can unify and harmonize a design. However, the effective realization of images of symmetric chaos depends on the use of color. As we shall show, the chromatics of chaos can lead to intriguing questions, especially when we require that the underlying symmetries permute color in a consistent way.

The images shown in this article were all generated using the software package **prism**, which I started to develop over ten years ago after a visit to see Marty Golubitsky at the University of Houston in 1988. At that time Marty Golubitsky, with Pascal Chossat (Nice), had described the phenomenon of *symmetry creation* in dynamical systems as well as the creation of symmetric designs in the plane by the iteration of planar polynomial maps [1]. Back in Sydney, Australia, Jim Richardson wrote a first C-program that produced colored images of symmetric chaos. Subsequently, I developed an interactive program to display various flavors of symmetric chaos on workstations that ran on X-windows and a UNIX platform. In collaboration with Marty Golubitsky, a range of new algorithms were added including symmetric fractals and all of the 1- and 2-color quilt patterns. Many colored images of symmetric chaos, as well as mathematical explanations, can be found in the book *Symmetry in Chaos* [3].

The last two years, I have given an interdisciplinary course on "Patterns, designs and Symmetry" at the University of Houston that introduces Junior/Senior students in Art to ideas of symmetry and design and the use of **prism**. Part of the course involves students learning familiarity with some of the nuances of UNIX (IRIX) and Linux. For some of the results, see the *URL*: `wotan.art.uh.edu`. Many other colored images of symmetric chaos can be found at the *URL*: `nothung.math.uh.edu/~mike/`.

As we are restricted in these notes to black and white images, we shall make occasional reference to the *URL*: `nothung.math.uh.edu/~mike/bridges.html` for colored examples of 2-colored quilts, We also refer the reader to the companion article 'Color symmetries in chaotic quilt patterns' [2], where a more pictorial description is given of the effects that can be generated using some of our 2-coloring algorithms.

2. Algorithms for symmetric designs

In this section we describe some of the algorithms we use to generate (symmetric) patterns on a computer. The algorithms we describe are based on ideas arising from the study of *dynamical systems* and *chaos* – areas of mathematics that have their origins in the Newtonian dynamical model of the solar system. While our methods for generating designs and coloring are firmly rooted in mathematics, we shall avoid, as far as possible, getting entangled in the underlying mathematical theory. (An elementary introduction, with references, can be found in [3].) Instead, we shall look at the problem of pattern generation by asking how one might actually draw a symmetric pattern on a computer screen.

2.1. The computer screen and pixels. A modern high resolution computer monitor typically has about 1000 lines, each line comprised of about 1200 small rectangles or squares called *pixels*. A typical pixel would have edges of length approximately one quarter of a millimeter (that is, about one hundredth of an inch). Our monitor therefore has about 1,200,000 pixels. For the moment, we shall assume that the monitor is 'black and white'. Later, we look at what happens when we have a color monitor.

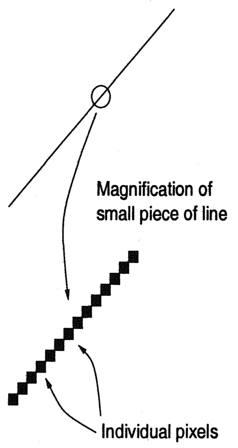


Figure 1: *Drawing lines on the screen*

By various electronic means, which need not concern us here, pixels can be switched on and off. Our convention will be that if a pixel is 'on' it is black and so seen as a small black dot on the screen. If the pixel is off, it is white. In particular, if all the pixels are off, the screen is white. Turning pixels on puts small black dots on the screen. In this way, the process of turning pixels on parallels drawing with a black pen on a white sheet of paper.

We draw on the screen by turning on pixels. For example, to draw a straight line, we turn on pixels along a line. Provided the pixels are small enough what we see is a straight line. Viewed more closely, the line may be rather jagged. See Figure 1, where we show a line and a portion of the line magnified.

For our purposes, we are going to think of the computer screen as representing the plane. Points in the plane will be represented by pixels. Of course, there are a lot more than a million points in the plane. Nevertheless, our assumption provides us with a good model to work with. In particular, we can discuss turning pixels on and off rather than worrying about how to compute points in the plane. If more 'points' are needed, we just imagine that our computer monitor is the very latest model: Say with $10^{10} \times 10^{10} = 10^{20}$ pixels.

As we shall be looking first at patterns that are bounded and have just rotational and reflectional symmetries, it is useful to establish at the outset the convention that the center of rotation of our pattern will always be the center of the computer screen. That is, the pixel (or point) situated exactly in the middle of the screen.

2.2 Pixel Rules – Pixels Rule. We denote specific pixels on our screen by capital letters such as $P, Q, R, \ldots$.

What is a pixel rule? Basically, a rule $\mathcal{R}$ that assigns to any pixel P a new pixel, say Q. It is probably easiest to think of a pixel rule as a 'black box' that takes pixels as input and has pixels for output. See Figure 2.

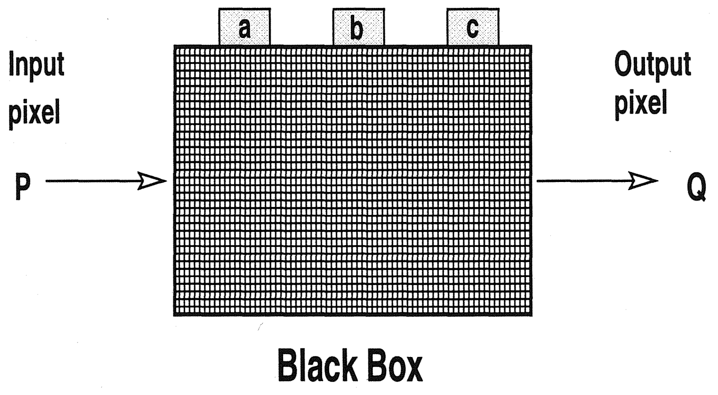


Figure 2: *A 'Black box' pixel rule, with controls*

Referring to Figure 2, notice that we have put some controls or 'dials', labeled **a**, **b**, **c** on the top of the box. The implication here is that even though we may not know much, if anything, about what goes on inside the box, nevertheless we may be able to affect the output by using the controls ('turning the dials')[1].

As we shall see, there are two basic types of pixel rule: Deterministic and non-deterministic. In a deterministic rule, a given input pixel P always leads to the same output pixel Q. In a non-deterministic rule, the same input does not always lead to the same output. We explain this dichotomy in the following examples.

Example 1. In Figure 3, we show how to construct a simple deterministic pixel rule. We fix a pixel, say the center pixel C. Suppose the input pixel is P. The output pixel Q is obtained in the following way. Draw a straight line from P to C. The output pixel Q will then be the midpoint of the line PC.

Here it is easy to envisage some controls. For example, when we turn the control **a**, we could move Q closer to C or away from C (depending on which way we turn the dial). On the other hand, changing **b** might move the point C. ♡

Example 2. Next we give a simple example of a non-deterministic pixel rule. Choose two distinct pixels from the screen. Say A and B. Take an input pixel P. Toss a fair coin. If the coin falls heads up, the output pixel will be A, otherwise it will be B. In this case, we never know with certainty what the actual output pixel will be. All we will know is that it will be either A or B. Even in this example, we can vary the controls a little. For example, turning the dial **a** might make the 'coin' less fair so that it tends to fall heads up more often than tails. We could also vary the pair of possible output pixels. For example, as we turn the dials, A and B might move along some preassigned path, perhaps a straight line or along a circle. ♡

In the previous example, there were two possible output pixels for each input. In general, we want to allow for a finite set of outputs for each input. In these notes we assume that each output is equally likely (though we can remove that restriction [3, p 187-189]). Before we give a more elaborate example, we need to explain what we mean by 'equally likely'.

[1]Otherwise put, we want to hide the mathematics!

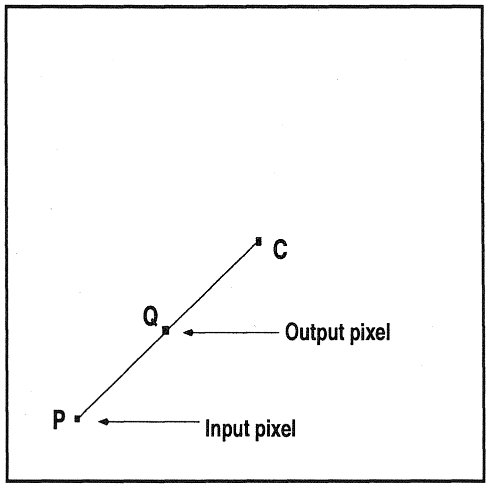


Figure 3: *A Deterministic pixel rule*

Imagine we have five tennis balls, all identical except that they are colored differently, say white, black, red, blue and green. Put the five balls in a bag, shake the bag and pick a ball out. What is the chance that you pick green? Since all colors are equally likely, the chance is one in five or 'one fifth'. More formally, we say that the probability of picking the green ball out of the bag is $\frac{1}{5}$. What this *means* is that if we repeat the experiment a very large number of times then on roughly one fifth of the occasions we will pick a green ball. We use the word 'roughly' here advisedly. In fact if we did the experiment one million times, it would be extremely *unlikely* that the green ball was picked *exactly* 200,000 times (one fifth of a million). However, it would be quite likely (and this can be made precise) that the green ball was picked between (say) 195,000 and 205,000 thousand times.

In what follows, we are going to imagine that inside our black box there is a cunning device that can make choices with equal probability from a given collection of objects. A kind of 'universal lotto machine' if you will. (Mathematicians or computer scientists would rather use the term 'random number generator'. If you suspect that it is difficult to make sense of all this, you are probably correct.)

Example 3. Our second example of a non-deterministic pixel rule is only a little more elaborate than our first example. Yet, as we shall see later, the rule leads to a symmetric pattern of stunning complexity.

Referring to Figure 4, we now describe the 'Sierpiński rule'. Let $\triangle ABC$ be an equilateral triangle. Let P be the input pixel. Let R, S, T be the midpoint pixels of the lines joining P to A, B and C respectively. We choose one of the pixels R, S, T with equal probability. That pixel will then be the output pixel.

Thus, for the input P, the possible outputs will be R, S and T. Each will be equally likely. That is, the probability of getting the output T is one third. If we run the pixel rule a large number of times with the same input P, then approximately one third of the outputs will be R, one third S and one third T. The rule is non-deterministic because even when we know the input we can never say with certainty what the output will be. All we can say is that there is a one third chance

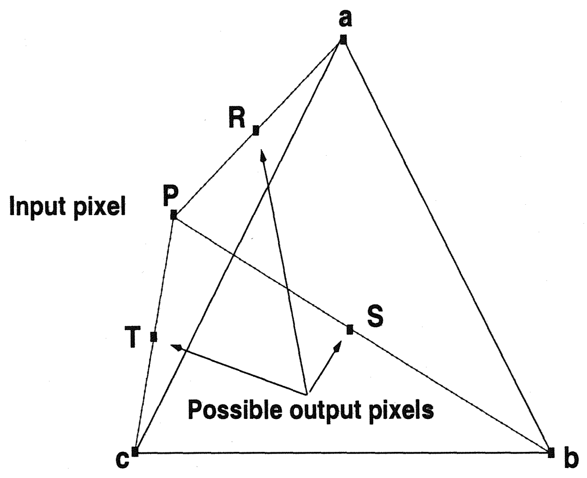


Figure 4: *The Sierpiński rule*

that the output will be, say, S. ♡

2.3. Drawing using a pixel rule. Our goal now is to use pixel rules to draw images on the computer screen. Here is the basic idea. Suppose we have a pixel rule. We start with an initial pixel P – the *seed*. We suppose this pixel is **on** and indeed is the only pixel that is turned on. Apply the rule to the pixel P to get a new pixel which we shall denote by P_1 (we use the '1' to indicate that the pixel comes from one application of the rule.) Turn P_1 on. We have now two points on the screen: P and P_1. Now we repeat the process, applying the rule to the pixel P_1 so as to obtain a new 'on' pixel P_2 (so P_2 results from two applications of the rule to the seed pixel P. In this way we obtain a string of 'on' pixels: $P, P_1, P_2, ...$ In the sequel we often refer to the points $P_1, P_2, ...$ as *iterates* of P and the underlying process as *iteration*.

Simply put, we feed the output of our black box back into the input and use this as a way to draw points on the screen. In practice we might apply the rule 50,000 times or more depending on what kind of image the pixel rule produces.

Example 4. What happens if we carry out this process using the rule described in Example 1? We show the results in Figure 5. Referring to the figure, we denote the seed pixel by P. We obtain P_1 by taking the pixel midway between P and center C of the screen. Then P_2 is obtained by taking the pixel midway between P_1 and the center and so on. In this way we obtain a line of dots that rapidly becomes indistinguishable from C. To emphasize the latter point, suppose that P was 10cm from the center C. Then P_1 is 5cm from C, P_2 2.5cm from C and so on. If we work out P_{20} (the result of applying the rule twenty times), we find that the distance between P_{20} and C is about one *ten thousandth* of a millimeter and so the points P_{20} and C become indistinguishable. ♡

An important feature of the previous example is that the long term behavior may be relatively independent of the seed pixel. Thus, for this example, we only see the limit pixel C in the long term, whatever the seed pixel P. This phenomenon is characteristic of most of the rules we look at, deterministic or non-deterministic. However, as we shall soon see, the long term behavior can be much more complex than just a single limiting pixel.

Turning to the simple non-deterministic rule we described in Example 2, whatever seed pixel P

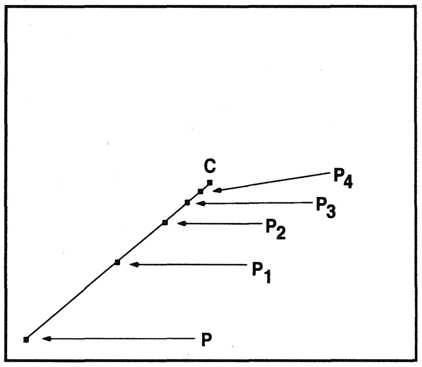


Figure 5: *Running a simple pixel rule*

we use, we only see a pair of pixels A, B in the output. Unless, that is, we are very unlucky. For example, if we run the rule 50,000 times it is *possible* that every single time the pixel A is chosen. The chances that this might happen are rather small. About 1 in 3 followed by 15,000 zeros.

Our next example, based on the algorithm described in Example 3 gives a much more interesting example of a non-deterministic pixel rule.

Example 5. In Figure 6 we show the result of running the Sierpiński rule 150,000 times. The resulting image is usually called the *Sierpiński triangle*. Note that we have not plotted the seed, or indeed the first 1,000 iterates of the rule.

There are some quite remarkable features about the Sierpiński rule.

The image we see is independent of the seed. That is, whatever seed we choose we always get the same final image.

The image, even though generated by a 'coin-tossing' procedure, is very regular. It has 3-fold reflectional symmetry and has the same structure on all scales.

The image is very complex: A non-deterministic process can lead to very definite, though complex, structure. ♡

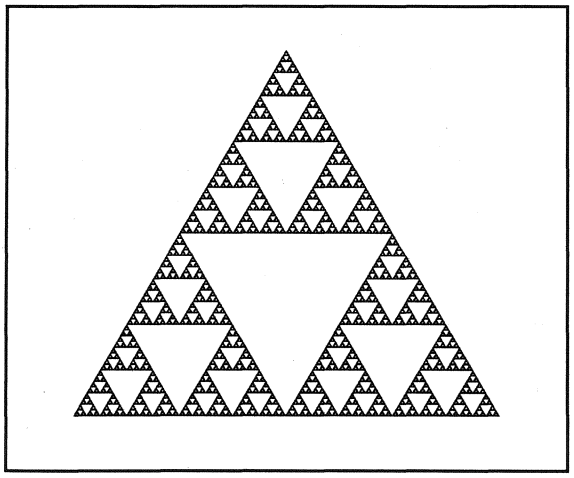


Figure 6: *The Sierpiński triangle*

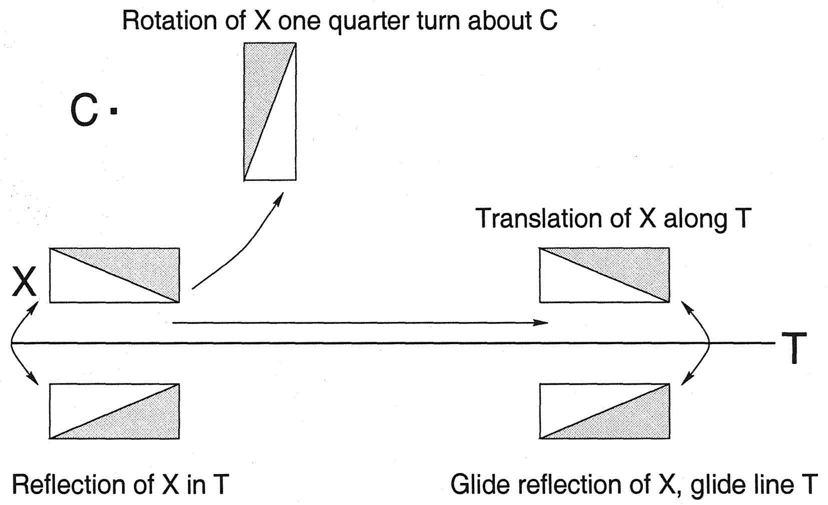

Rotation of X one quarter turn about C
C
X
Translation of X along T
T
Reflection of X in T
Glide reflection of X, glide line T

Figure 7: *Planar symmetries*

3. Planar symmetry

In this section, we give a very brief review of the concept of symmetry and describe the various types of symmetry that occur for bounded planar figures. In later sections, we look at unbounded repeating patterns.

3.1. Symmetry. There are several possible approaches to defining the concept of symmetry. We shall adopt an operational approach and define symmetry in terms of transformations that leave the object unchanged. Roughly speaking, we say that an object has a nontrivial symmetry if we can move it around in space and put it back in its original position but with a different orientation. Suppose that $\mathcal{X}$ is a planar figure. Then $\mathcal{X}$ can have precisely four types of symmetry.

- *Rotational symmetry*: $\mathcal{X}$ remains unchanged after rotation about some point C in the plane. (C is called the center of rotation.)
- *Reflectional symmetry*: $\mathcal{X}$ remains unchanged after reflection through a line L in the plane. (L is called a line of symmetry.)
- *Translational symmetry*: $\mathcal{X}$ remains unchanged when we slide parallel to a line T in the plane. (T is called an axis of translation.)
- *Glide reflection symmetry*: $\mathcal{X}$ remains unchanged under a translation followed by a reflection. (The axis of translation is called a glide line. Neither the reflection nor the translation should be a symmetry of $\mathcal{X}$.)

We illustrate the operations of rotation, reflection, translation and glide reflection in Figure 7. In Figure 8 we show a strip pattern that exhibits all four types of symmetry (rotational symmetries are through a half turn).

It is convenient to regard the operation of 'doing nothing' as a symmetry: The trivial symmetry, **I**. With this convention, any combination of rotations, reflections, translations and glide reflections is either the trivial symmetry or a rotation, reflection, translation or glide reflection. For example, if we reflect twice in the same line, we obtain the trivial symmetry. On the other hand, the result

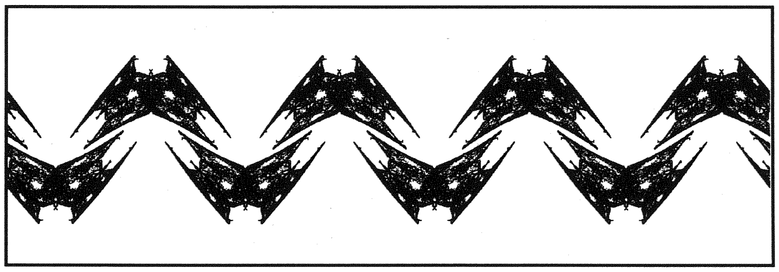


Figure 8: *Strip pattern of type* **pma2** *which exhibits all four types of symmetry*

of successive reflection in three lines which are not parallel and contain no common point, is a glide reflection (see [7, Appendix I] for justification).

If $\mathcal{X}$ is a planar figure then the *symmetry set* or *symmetry group* $\mathbb{S}(\mathcal{X})$ of $\mathcal{X}$ is the collection of all symmetries of $\mathcal{X}$ (including the trivial symmetry).

Remark Any composition of symmetries of $\mathcal{X}$ is a symmetry of $\mathcal{X}$. If S is a symmetry of $\mathcal{X}$ then there is a symmetry $S^\star$ of $\mathcal{X}$ such that the result of composing the symmetries S and $S^\star$ is the trivial symmetry of $\mathcal{X}$. ◇

3.2 Symmetries of bounded figures. Suppose that $\mathcal{X}$ is a bounded figure in the plane. It can

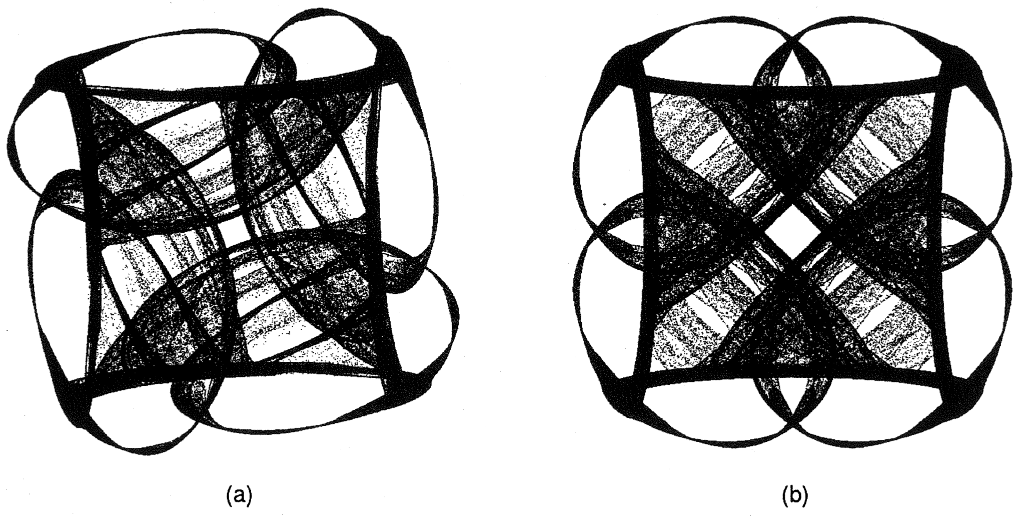


Figure 9: *Rotational and reflectional symmetry*

be shown that symmetries of $\mathcal{X}$ are either rotations or reflections. In Figure 9 we show examples of designs with 4-fold rotational and reflectional symmetries. Figure 9(a) has exactly four symmetries: The trivial symmetry $\mathbf{I}$, rotation through one quarter of a turn about the center, and rotations about one half and three quarters of a turn about the center. (We denote these symmetries by $\mathbf{R}_{\frac{1}{4}}$, $\mathbf{R}_{\frac{1}{2}}$ and $\mathbf{R}_{\frac{3}{4}}$ respectively.)

Figure 9(b) has a total of eight symmetries. In addition to four rotational symmetries there are four reflectional lines of symmetry (the horizontal, vertical and diagonal lines through the center of the figure). We say that Figure 9(b) has 4-fold reflectional symmetry.

We may similarly define n-fold rotational and reflectional symmetry for any integer $n \geq 2$. Note that 1-fold reflectional symmetry is just *bilateral symmetry*.

4. Symmetric pixel rules

In this section, we consider conditions that we can impose on a pixel rule so that it generates a symmetric image.

4.1 Symmetric deterministic pixel rules. What are the properties we might reasonably impose on a deterministic algorithm in order that it might generate an image with a given rotational or reflectional symmetry? In order that the image generated by the algorithm be symmetric, it is reasonable to ask that the algorithm itself be 'symmetric'.

For example, suppose we want a deterministic algorithm to generate an image with 4-fold rotational symmetric (for example, the image shown in Figure 9(a)). The symmetries we want the image to have are the quarter turn symmetries $\mathbf{R}_{\frac{1}{4}}$, $\mathbf{R}_{\frac{1}{2}}$, $\mathbf{R}_{\frac{3}{4}}$, together with the trivial symmetry $\mathbf{I}$.

Suppose that P is the initial input pixel and Q is the output pixel determined by our algorithm. Instead of P, we might have taken $P' = S(P)$, where S is one of the symmetries $\mathbf{R}_{\frac{1}{4}}$, $\mathbf{R}_{\frac{1}{2}}$, $\mathbf{R}_{\frac{3}{4}}$. For example, if we take $S = \mathbf{R}_{\frac{1}{2}}$, then P' is the point P rotated through a one half turn. Let Q' be the output pixel corresponding to P'. It is reasonable to ask how Q' might be related to Q. Without further conditions on our algorithm, there need be no relationship between Q' and Q. We shall say that our algorithm is *symmetric* (strictly, has 4-fold rotational symmetry) if $Q' = S(Q)$. What this means is that once we know what our algorithm does to a pixel P, we know what it does to the symmetric images of P. If we denote our algorithm by $\mathcal{A}$ and write $\mathcal{A}(P)$ to denote the effect of $\mathcal{A}$ on P, then the symmetry of $\mathcal{A}$ implies that

$$\mathcal{A}(S(P)) = S\mathcal{A}(P),$$

for all pixels P and symmetries S.

It can be shown that this condition on the algorithm is necessary for it to produce a symmetric image.

Example 6. Suppose that the pixel rule says: Rotate through one quarter of a turn about the center of the screen. If we take as input the pixel P and repeatedly apply the rule, we get four pixels P, $\mathbf{R}_{\frac{1}{4}}(P)$, $\mathbf{R}_{\frac{1}{2}}(P)$, $\mathbf{R}_{\frac{3}{4}}(P)$. Obviously this algorithm has 4-fold rotational symmetry as does the set of pixels that is drawn on the screen. ♡

The image plotted in the previous example depends on the initial pixel. If we change the initial pixel then the resulting set of pixels drawn on the screen will correspondingly change.

We shall require that our deterministic algorithm possesses the property that the resulting pixel image is essentially independent of the initial pixel. It is quite remarkable that such rules exist

– and even more remarkable that to get an interesting image, the rule basically has to generate 'chaotic dynamics'.

Example 7. In Figure 10, we show the result of plotting just a few points using an algorithm with

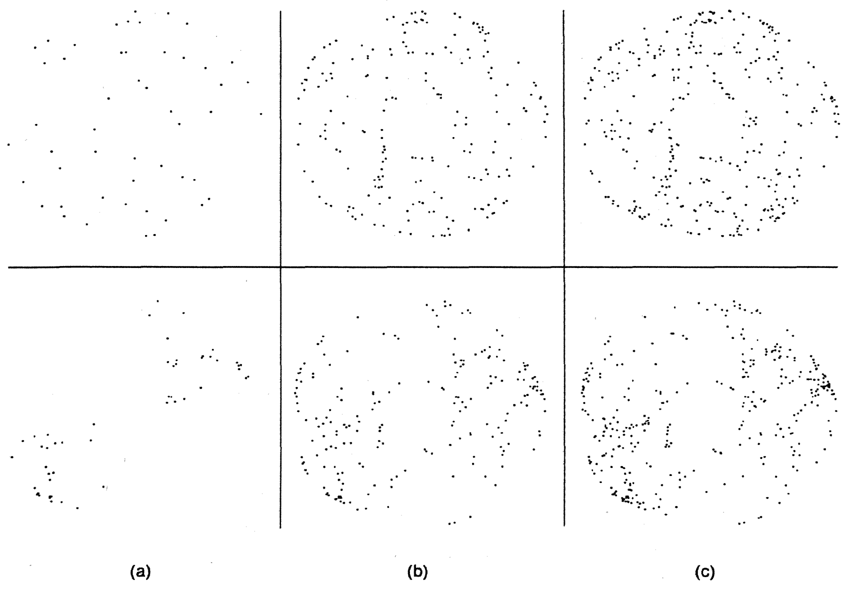


Figure 10: *Plotting small numbers of points*

5-fold rotational symmetry. Both rows of three images in Figure 10 were obtained in the following way. Starting with the same initial point, the first 20 iterates were computed, but not drawn on the screen. In (a), the next 50 points are plotted, in (b) we plot another 150 points and in (c) another 150 points. (Notice that every pixel hit in (a) also appears in (b) and (c).) In the first image, what one sees is a point that appears to move randomly and there is no evidence of any detailed structure. In (b) it appears that the point moves about on a disk shaped region and in (c), although there is some evidence of structure, about the best one can say is that pixels near the center appear not to be hit. The second row is computed using a different initial point from the first row. Although different pixels are hit, the images (b) and (c) are qualitatively the same. In fact, the initial points for the two sequences of pictures only differ in their x-coordinate and then only by 0.00001. Notice that although the initial points are very close, the two images shown in column (a) are quite different. This 'sensitive dependence on initial conditions' (aka the 'butterfly effect') is characteristic of the deterministic algorithms that lead to designs showing rich structure. Finally, notice that even though the algorithm we use has 5-fold rotational symmetry, the images we draw are hardly even approximately 5-fold symmetric. ♡

While the images we show in Figure 10 depend both on the initial pixel and the number of points plotted, it is a remarkable fact that if we plot a large number of pixels then the resulting image is apparently *independent* of the initial pixel. Moreover, the image may be both symmetric and possess detailed and intricate structure. In Figure 11, we show the result of plotting 100,000 points using the same algorithm used in Figure 10. Observe that this image apparently has 5-fold rotational symmetry. Very close examination of the dots in the image reveals that the 5-fold symmetry is only approximate though it is close to being exact. Experimentation shows that the more points we plot, the closer we apparently get to exact 5-fold rotational symmetry. Moreover,

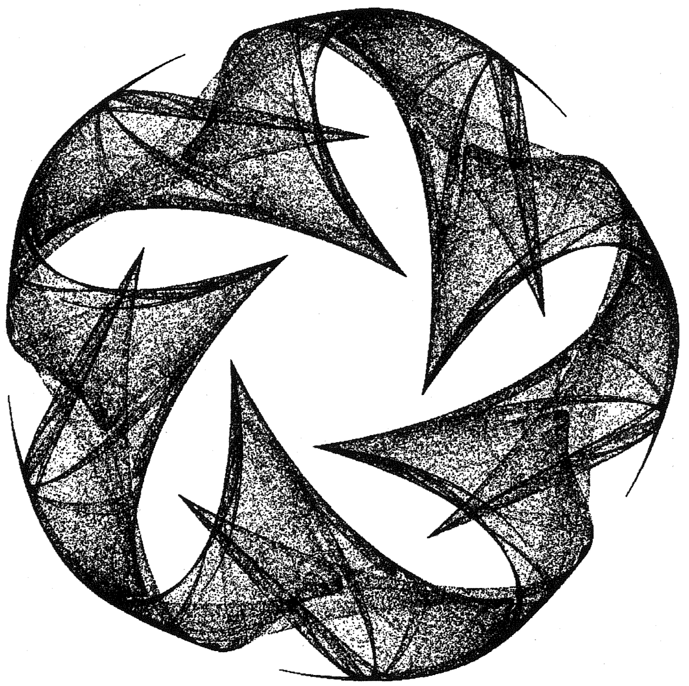


Figure 11: Thorns - *a symmetric icon with 5-fold rotational symmetry*

we also find that the resulting image is apparently independent of the initial pixel. Again, this independence is approximate, though close to exact. We call bounded symmetric images produced in this way *Symmetric Icons* [3].

What we see here is that if we plot just a few points with this algorithm then the resulting image possesses little or no structure and is highly dependent on the initial pixel. However, if we plot a large number of points then we get a well defined image which is, to a high order of approximation, independent of the initial pixel. To make sense of this apparent contradiction, it helps to view the process of drawing the image as a dynamic process. That is, the image is not drawn at one instant of time but is drawn pixel-by-pixel over a period of time. The dynamic process associated with plotting the points is *chaotic* – that is, depends sensitively on the initial pixel. What we see in the long-term is the *average* behavior of the dynamic process. That is, even though the process is chaotic, it is possible for it to have structure on average. Looking again at Figure 11, we see that some areas of the image are darker than others. The darkness of the image, or density of the dots, is a measure of the amount of time spent in that region. Areas sparsely populated by dots, will be infrequently visited as we iterate our algorithm. Densely populated areas are frequently visited. In a later section, we explain how we can use color to show fine structure in the image.

4.2. Symmetric non-deterministic pixel rules. Perhaps surprisingly, it is rather easy to describe a large class of symmetric *non-deterministic* pixel rules. Specifically, suppose we want to construct a non-deterministic algorithm $\mathcal{A}$ that produces an image $\mathcal{I}$ with a specific symmetry group $\mathbb{S}$. For simplicity, assume that $\mathcal{I}$ is bounded and that $\mathbb{S}$ consists of just rotations and reflections. A natural condition to impose is that for every pixel input P, the set of possible outputs $Q_1, \ldots, Q_N$ has symmetry group $\mathbb{S}$. In particular, the elements of $\mathbb{S}$ will permute the points $Q_1, \ldots, Q_N$. Here is a practical way to implement a rule with this property. Let $\mathcal{R}$ be a deterministic pixel rule that leads for almost all initial pixels to the same bounded image – for example, the center C of the

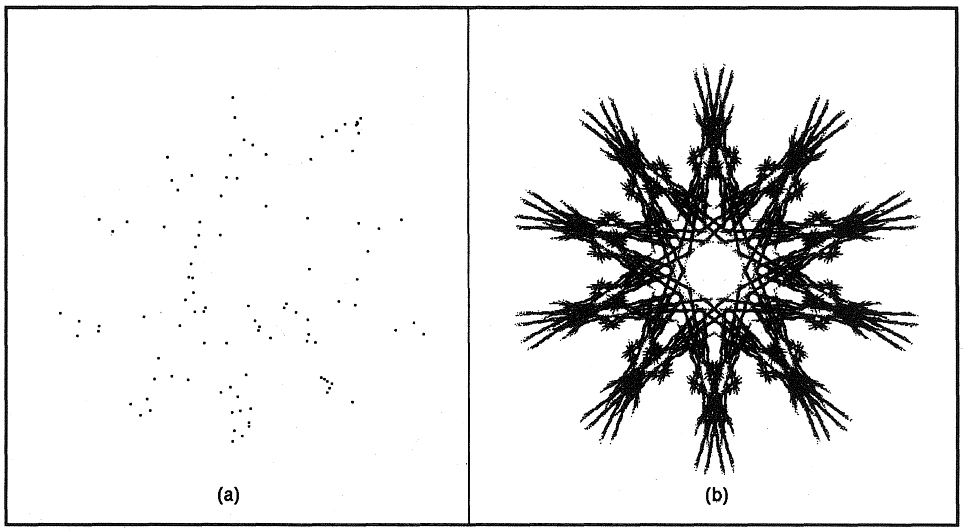


Figure 12: *Symmetric Fractal with 10-fold reflectional symmetry*

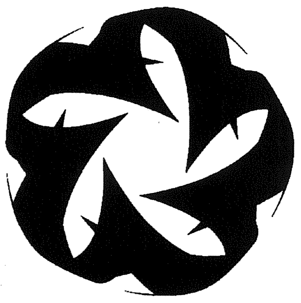
screen. We now describe a non-deterministic algorithm $\mathcal{A}$ associated to the rule $\mathcal{R}$. If P is the input pixel, let $T = \mathcal{R}(P)$ denote the output pixel determined by $\mathcal{R}$. Randomly choose a symmetry S from $\mathbb{S}$ and define the output $\mathcal{A}(P)$ to be $S(T)$.

For example, suppose that $\mathbb{S}$ was the group of rotations through multiples of one tenth of a whole turn about C. Pick P, apply $\mathcal{R}$ to get T and then make a rotation of T about C of some random multiple of one tenth of a whole turn.

It can be shown that in many cases iterating this procedure will lead to a bounded image with symmetry group $\mathbb{S}$. Images produced in this way are called *Symmetric Fractals* in [3]. This method can also be used to produce quilt patterns and indeed the strip pattern shown in Figure 8 was produced using a non-deterministic algorithm of this type. In Figure 12 we show two fractal images computed using the same algorithm. The left hand image was computed using just 100 points. For the second image we computed 250,000 iterates. (Note that the dots in the first image are drawn about 10 times normal size to enhance their visibility.)

5. Using color to illuminate chaos

Already in our black and white images of Symmetric Icons and Fractals, we have seen how both deterministic and non-deterministic symmetric algorithms can lead to images which contain fine structure and are quite non-uniform. If we take the algorithm that produced *Thorns* (Figure 11) but plot a larger number of points, we find that we lose detail. In the figure on the right, we show the result of computing 50,000,000 iterates. Clearly, there is a point when plotting more iterates loses rather than gains detail. Recall, that our hypothetical monitor screen has 1,200,000 pixels. Consequently, if we attempt to plot more than 1,200,000 points, some of the pixels on the screen will be hit more than once. In practice, for images like *Thorns*, we find that when we plot a large number of iterates, many pixels are hit infrequently (or not at all) while some pixels are hit very many times. Typically those pixels that



are hit many times appear as lines or other fine structure in the image. The way we can bring out the detailed structure of the image is to use color. Roughly speaking, we 'color by number'. More precisely, a pixel is colored according to the number of times it was hit in the iteration. The color of a pixel represents the frequency with which it is hit during the iteration. From the mathematical point of view, we may think of the coloring as a 'colored measure'. When we compute a large number of iterates, we find that the coloring of the image (or the colored measure) has the same symmetry as the image. Many examples of colored images produced using **prism** can be found in the book *Symmetry in Chaos* [3]. Other colored images, including *Thorns*, may be found at the *URL*: `nothung.math.uh.edu/~mike/Art/Art.html`.

5.1. Chromatics of Symmetric Icons and Fractals. Aside from their symmetry, Icons and Fractals are quite different in their appearance and texture. Icons typically exhibit intricate structure and fine detail. The detail manifests itself in the forms of curves, often ending in cusps. While the fine detail is what leads to the attractiveness of the design of the Icon, this detail often only occupies 1-2% of the area of the Icon. Much of the rest of the Icon contains regions with little or no detailed structure. These regions allow for shading effects. In brief, Icons emphasize 'edge data' rather than texture. On the other hand, Fractals typically display a wealth of textural information but little in the way of fine detail. This difference between Icons and Fractals is also shown in the statistics of pixel hits when we iterate to produce an image. For example, if we do 20,000,000 iterations to construct an Icon and a Fractal for the same size screen, we usually find that the range of pixel hits for the Fractal goes from 0 to maybe 1,000 while the range for the Icon might vary from 0 to 50,000 or more. Pixels that record a high number of hits show as lines in the image of an Icon. Thus the approach to coloring Icons and Fractals is rather different. For Icons, we choose colors so as to show fine detail and balance this with careful shading of colors over the remainder of the attractor. For Fractals, most of the effort goes into showing textures to good effect. Finally, we remark that even though the colored image appears highly symmetric, examination of symmetrically related pixels reveals that the symmetry of the coloring is only approximate. Again, this is just a reflection of the fact that the underlying dynamics is chaotic and that what we see is a representation of *average* behavior.

Remark The coloring schemes used for Symmetric Icons and Fractals and those used to color, for example, the *Mandelbrot set* are quite different. For the Mandelbrot set, colors are chosen on the basis of 'escape times'. The corresponding boundaries between different colors are characteristically sharp. Modifications of the coloring algorithms lead to smooth and gradual variations in color – effects that cannot be easily realized with Symmetric Icons and are unattainable with Symmetric Fractals. $\diamondsuit$

6. Quilt patterns

So far in these notes we have focused on bounded symmetric patterns in the plane. In this section, we give a brief discussion of unbounded symmetric patterns in the plane, also known as 'wallpaper' or 'quilt' patterns. Throughout, we shall use the term *quilt pattern* to be synonymous with two dimensional repeating pattern. It is possible to give a symmetry classification of two-dimensional repeating patterns. In this classification, there are exactly *seventeen* basic quilt patterns. Each of these basic patterns is characterized by their rotational, reflectional and glide reflection symmetries. We refer the reader to Washburn & Crowe [7] for a detailed description of

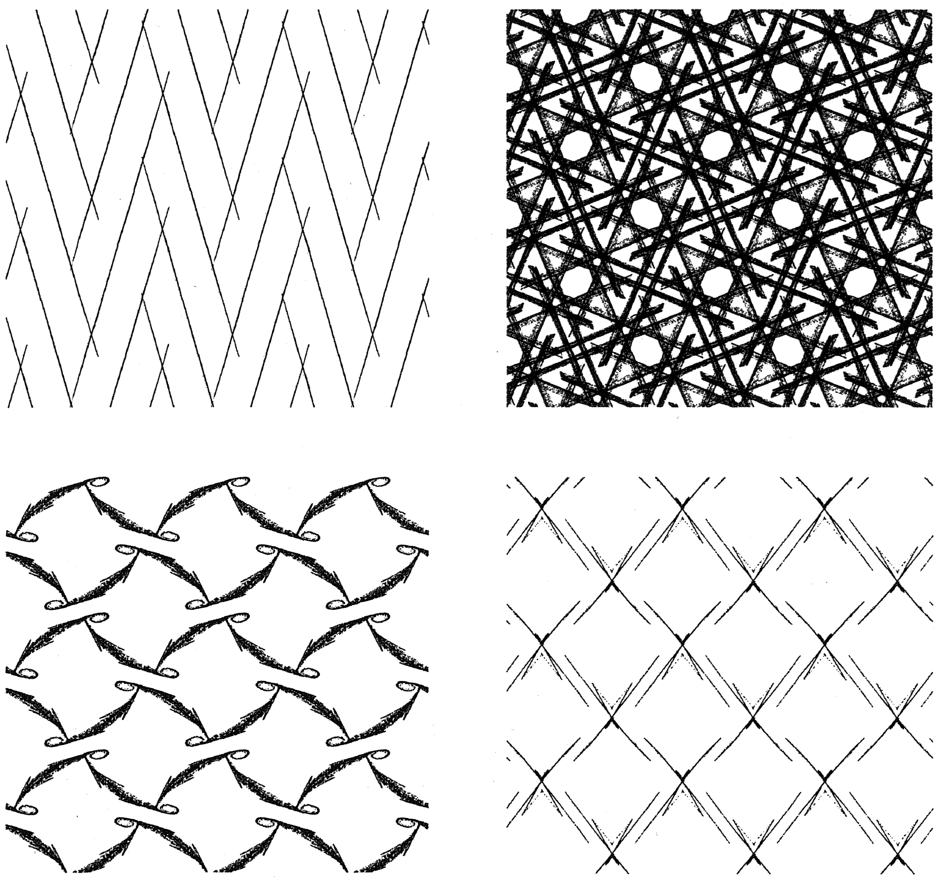


Figure 13: *Four quilt patterns*

the seventeen quilt patterns together with algorithms that enable one to identify the symmetry type of a given pattern.

Sometimes it is not so easy to identify the symmetry type of a quilt pattern. In Figure 13, we show four quilts generated using **prism**. In this figure, four distinct symmetry types appear: **pmg, pg, pgg** and **p4g**, where we follow the notational conventions of [7]. All four quilts have glide reflection symmetries.

Quilt patterns can be drawn using deterministic or non-deterministic algorithms. Designs for quilts of type **p4m, p4** (Square quilts) and **p6m, p6** (Hexagonal quilts) based on deterministic algorithms may be found in [3]. The character of these quilts is similar to that of Symmetric Icons. We have also developed a number of non-deterministic algorithms that generate all of the seventeen wallpaper patterns. Some of these algorithms can produce designs that have a certain angularity, even a cubist flavor. All of the designs in Figure 13 were produced with algorithms of this type[2]. On the other hand we have algorithms that yield designs intermediate in character between Symmetric Icons and Fractals. These designs typically contain fine structure (curves and cusps) as well as significant texture. We show four (black and white) examples of designs of this type in Figure 14.

[2]Note, however, that these examples were chosen so that most pixels were not hit in the iteration. We did this simply because of our restriction to black and white images.

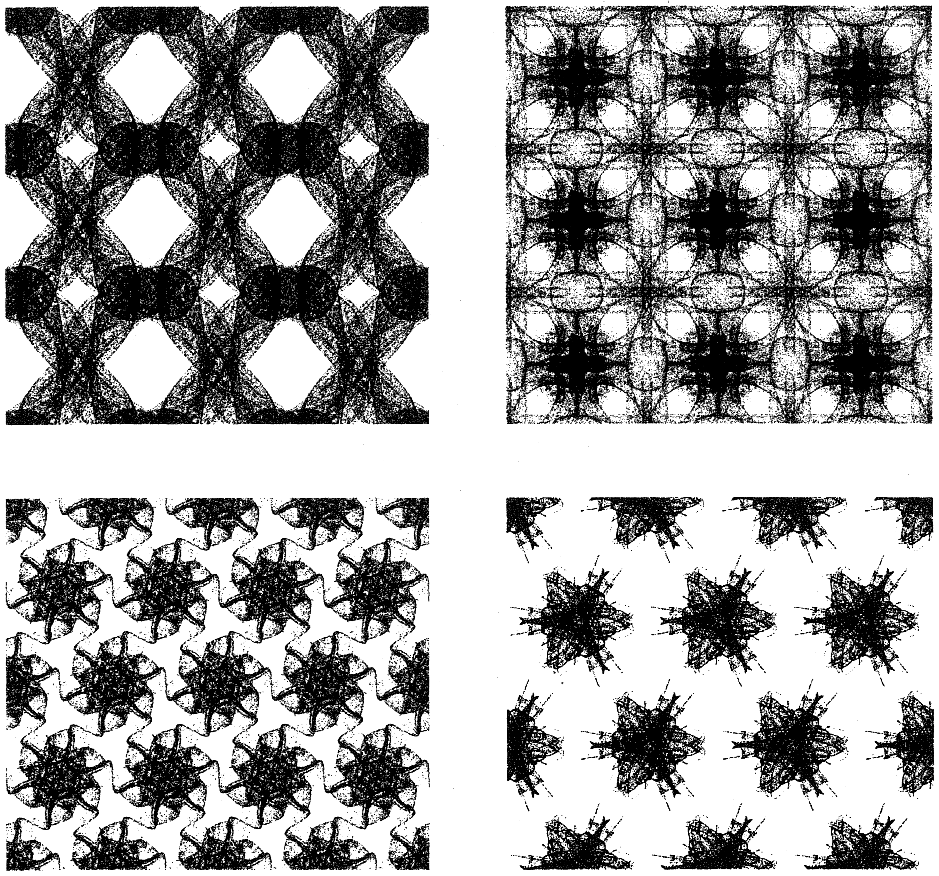


Figure 14: *Quilt patterns of type* **pmm, p4m, p6, p3m1**

7. 2-colorings of quilt patterns

For the remainder of this article, we focus on quilt patterns which admit *2-color symmetries*. To simplify our exposition, we will look at a model quilt pattern of type **p4m**. This symmetry is the classic 'square tiling' pattern an example of which was shown in Figure 14. (We refer to [7] for general details about the forty-six 2-color symmetric patterns.)

In Figure 15 we show a single square tile which has a two color symmetry. Reflections in the lines Q and P and rotations through a half turn about C preserve color, whereas reflections in the lines L and M and one quarter and three quarter turns reverse colors. Notice that this tile really has three colors: the two colors represented by the disks together with the background color (white or transparent). The background color, however, is preserved by all symmetries of the tile. If we tile the plane with these square tiles, placing them all with the same orientation, we obtain a 2-color quilt $\mathcal{Q}$ of type **p4′m′m** [7, p 154].

We shall regard the design of an individual tile as consisting of the two black disks and the two shaded disks. Let $\mathcal{A}$ denote the repeating pattern made from tiles containing just the two black disks, and $\mathcal{B}$ denote the repeating pattern comprised of the shaded disks. If S is a symmetry of the quilt $\mathcal{Q}$ then either $S(\mathcal{A}) = \mathcal{A}$ (S is a symmetry of $\mathcal{A}$), or $S(\mathcal{A}) = \mathcal{B}$ (S is a color reversing symmetry). In this way, we may think of a 2-color quilt $\mathcal{Q}$ as a combination of two sub-quilts, $\mathcal{A}$

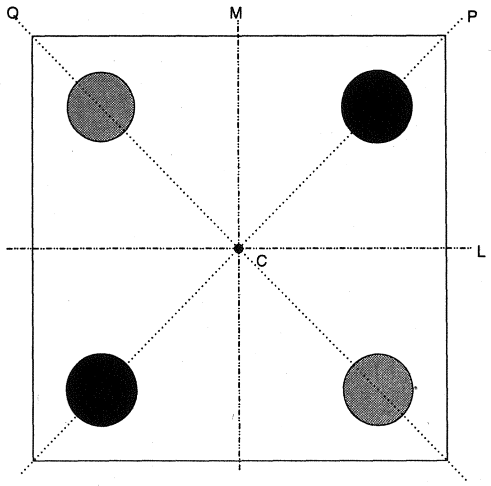


Figure 15: *A square tile with 2-color symmetry*

and $\mathcal{B}$, such that every symmetry of $\mathcal{Q}$ either preserves $\mathcal{A}$ and $\mathcal{B}$ or interchanges $\mathcal{A}$ and $\mathcal{B}$[3].

Both $\mathcal{A}$ and $\mathcal{B}$ may be thought of as symmetrically related designs on a transparent (white) background. From this perspective, we can start with a quilt $\mathcal{A}$ and the symmetries of the desired 2-colored quilt $\mathcal{Q}$. 'Half' of these symmetries will preserve $\mathcal{A}$. The remaining symmetries will produce the other half $\mathcal{B}$ of the 2-color quilt $\mathcal{Q}$. If there is no overlap between $\mathcal{A}$ and $\mathcal{B}$, then we just assign different colors to the designs in $\mathcal{A}$ and $\mathcal{B}$. But what do we do if there is overlap? See, for example, the design shown in Figure 16 where we have chosen the shaded disks sufficiently large so that they overlap with their symmetric images. One solution is just to introduce a third color for the overlap. When we act by symmetries of $\mathcal{Q}$ this new color will then be preserved just as happened for the background color. This coloring strategy leads to a minimum of four colors for a typical 2-color quilt: One color for each of the sub-quilts, a color for the background and a color for the overlap.

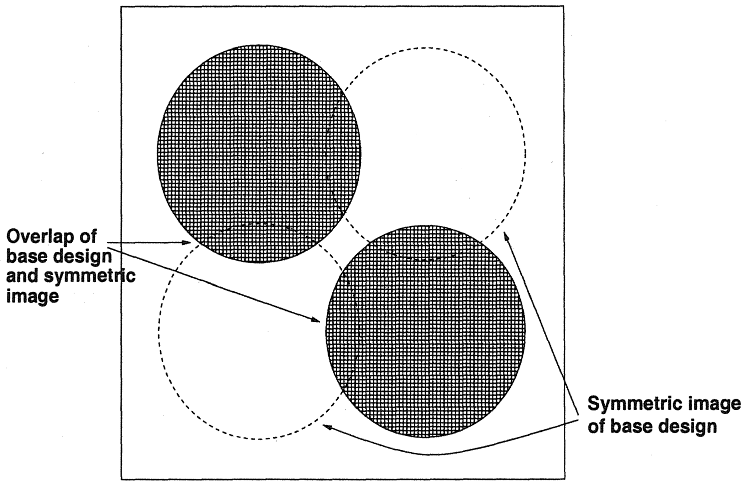

Overlap of base design and symmetric image
Symmetric image of base design

Figure 16: *Overlapping designs*

7.1. Algorithms for generating 2-color quilts Using methods depending on non-deterministic

[3]In our example, the individual (uncolored) quilts $\mathcal{A}$, $\mathcal{B}$ are of type **cmm**.

algorithms, we can generate all of the forty-six 2-color quilts[4]. When we come to color these quilts, a number of tricky questions arise. First of all, as we indicated above, every 2-color quilt $\mathcal{Q}$ can be expressed as the union of two symmetrically related sub-quilts, $\mathcal{A}$ and $\mathcal{B}$. Each pixel corresponding to a point in either $\mathcal{A}$ or $\mathcal{B}$ will typically be hit multiple times in the iteration producing the quilt. How do we color? One strategy is to color by *symmetry*.

7.2. Coloring by symmetry For each pixel P, let $n_{\mathcal{A}}(P)$ be the number of times the pixel P was hit in the iteration producing $\mathcal{A}$. We similarly define $n_{\mathcal{B}}(P)$. If $n_{\mathcal{A}}(P) + n_{\mathcal{B}}(P) > 0$, we define

$$\mathcal{A}(P) = \frac{n_{\mathcal{A}}(P)}{n_{\mathcal{A}}(P) + n_{\mathcal{B}}(P)}; \; \mathcal{B}(P) = 1 - \mathcal{A}(P).$$

Clearly, $0 \leq \mathcal{A}(P), \mathcal{B}(P) \leq 1$. If $\mathcal{A}(P) = 0$, then the pixel P is not hit in the iteration producing $\mathcal{A}$, while if $\mathcal{A}(P) = 1$, then the pixel P is not hit in the iteration producing $\mathcal{B}$. Where there is overlap between $\mathcal{A}$, $\mathcal{B}$, we have $0 < \mathcal{A}(P), \mathcal{B}(P) < 1$.

Now we can color $\mathcal{Q}$ in the following way. Choose two colors, say red and blue. Color P red if $\mathcal{A}(P) = 1$ and blue if $\mathcal{A}(P) = 0$. On the overlap, we take the color of P to be

$$\mathcal{A}(P) \times \text{red} + \mathcal{B}(P) \times \text{blue}.$$

That is, we linearly interpolate colors between red and blue on the overlap. If we work in terms of RGB values and take red to be $(255, 0, 0)$ and blue to be $(0, 0, 255)$, then the RGB value of P will be given by

$$\text{RGB}(P) = (255\mathcal{A}(P), 0, 255\mathcal{B}(P)),$$

rounded to the nearest integer RGB value. This coloring scheme results in a coloring of $\mathcal{Q}$ that is consistent with the 2-color symmetries of the pattern. Of course, we can vary the interpolation scheme as long as the interpolation is symmetric.

Examples of three different 2-colorings of the same underlying design – a quilt of type **pmg** – may be found at the *URL*: `nothung.math.uh.edu/~mike/bridges.html`. (Figures 3, 4 and 5 show symmetry based 2-colorings of types **pm′g, pm′g′** and **pmg′**.)

7.3. Coloring by symmetry and dynamics If we color a 2-color quilt $\mathcal{Q}$ solely by symmetry, then we lose most of the detail that comes from the *dynamics* of the iteration. In particular, suppose that $\mathcal{Q}$ is comprised of the two symmetrically related sub-quilts $\mathcal{A}$, $\mathcal{B}$ and that $\mathcal{A}$, $\mathcal{B}$ have no overlap. If we color by symmetry alone, we only use two colors (together with the background color). In this situation, where there is no overlap, the following approach would seem more natural. Start by choosing a coloring of $\mathcal{A}$ that takes account of the dynamics (that is, the relative frequency of pixel hits). Symmetrically transform the colors used for $\mathcal{A}$ to a new set of colors for $\mathcal{B}$ and color $\mathcal{B}$ using the same frequencies that we used for $\mathcal{A}$. In this way, we obtain a 2-coloring using many colors. We now have to take account of overlap. The 2-coloring algorithms that we have developed color the overlap using coloring rules based on symmetry and dynamics. Examples of 2-color fractal quilts colored using a relatively simple algorithm of this type may be found at *URL*:

[4]With the exception of $\mathbf{p}'_b\mathbf{1}$, examples of all of the 2-color quilts, generated using **prism**, may be found at the *URL*: `nothung.math.uh.edu/~mike/quilts/col2quilts/col2quilts.html`.

`nothung.math.uh.edu/~mike/bridges.html`. (Figures 8–10 are examples of 2-colorings of a quilt of type **pmg**, and Figure 11 is a 2-coloring of a quilt of type **cmm**.)

7.4. Two-color algorithms We conclude by formalizing some of the characteristic properties that a practical 2-color algorithm should satisfy. We denote RGB-space by $\mathcal{M}$. One possibility is to take $\mathcal{M} = \{0, \ldots, 255\}^3$ (the space of RGB-triples). Alternatively, we can use the continuous model $\mathcal{M} = [0,1]^3$. In what follows we adopt the continuous model on the grounds of mathematical convenience and ignore (non-trivial) issues of translating values in $[0,1]^3$ to TrueColor values or PseudoColor color maps.

Let $\mathbb{N}^+$ denote the strictly positive integers. A permutation κ of $\{1, \ldots, k\}$ induces a transformation $\bar{\kappa} : \mathcal{M}^k \to \mathcal{M}^k$ defined by $\bar{\kappa}(a_1, \ldots, a_k) = (a_{\kappa(1)}, \ldots, a_{\kappa(k)})$. We say that κ is an *involution* if κ^2 is the identity permutation. If κ is an involution then $\bar{\kappa}^2$ is the identity map of $\mathcal{M}^k$.

A *k-parameter two-coloring algorithm* consists of an involution κ of $\{1, \ldots, k\}$ together with a transformation

$$\mathbf{C} : \mathbb{N}^+ \times \mathcal{M}^k \times [0,1] \to \mathcal{M},$$

such that for all $n \in \mathbb{N}^+$, $\mathbf{a} = (a_1, \ldots, a_k) \in \mathcal{M}^k$ and $p \in [0,1]$ we have

$$\mathbf{C}(n, \mathbf{a}, p) = \mathbf{C}(n, \bar{\kappa}(\mathbf{a}), 1-p). \tag{1}$$

In the sequel, we usually set $\mathbf{C}(n, \mathbf{a}, p) = \mathbf{C_a}(n, p)$. In order to implement the algorithm $\mathbf{C}$, we make an initial choice of k colors $\mathbf{a} = (a_1, \ldots, a_k)$ and then color pixels P according to the rule

$$\mathbf{C_a}(n_{\mathcal{A}}(P) + n_{\mathcal{B}}(P), \mathcal{A}(P)).$$

Notice that the data $(n_{\mathcal{A}}(P) + n_{\mathcal{B}}(P), \mathcal{A}(P))$ uniquely determines $n_{\mathcal{A}}(P), n_{\mathcal{B}}(P)$, and $\mathcal{B}(P)$. Fix $\mathbf{a} \in \mathcal{M}^k$. We say that $\mathbf{C_a}$ is

1. *Symmetrically non-degenerate* if $\mathbf{C_a}(n,p) \neq \mathbf{C_a}(n,p')$ if and only if $\mathbf{C_a}(n, 1-p) \neq \mathbf{C_a}(n, 1-p')$, $p, p' \in [0,1]$, $n \in \mathbb{N}^+$;
2. *Dynamically non-degenerate* if $\mathbf{C_a}(n,p) \neq \mathbf{C_a}(m,p)$ if and only if $\mathbf{C_a}(n, 1-p) \neq \mathbf{C_a}(m, 1-p)$, $p \in [0,1]$, $m, n \in \mathbb{N}^+$;
3. *Strict* if $\mathbf{C_a}(m,p) = \mathbf{C_a}(n,q)$ if and only if $m = n$, $p = q$.
4. *N-strict* if whenever either of m, n is strictly less than N, then $\mathbf{C_a}(m,p) = \mathbf{C_a}(n,q)$ if and only if $m = n$, $p = q$.

Roughly speaking, a coloring is symmetrically non-degenerate if the colorings of a pair of pixels in $\mathcal{A}$ are distinct if and only if the colorings of their symmetric images in $\mathcal{B}$ are distinct. Notice that if a coloring is strict, then it is automatically symmetrically and dynamically non-degenerate.

We say that $\mathbf{C}$ is symmetrically non-degenerate if $\mathbf{C_a}$ is symmetrically non-degenerate for almost all choices of $\mathbf{a} \in \mathcal{M}^k$. That is, if $\mathbf{C_a}$ is generically symmetrically non-degenerate. We similarly define dynamic non-degeneracy and strictness for $\mathbf{C}$.

Example 9. Let $a_1, a_2 \in [0,1]^3$ be RGB-triples. Let $\kappa(a_1, a_2) = (a_2, a_1)$. Define $\mathbf{C_a}(n,p) = pa_1 + (1-p)a_2$, $n \in \mathbb{N}^+$, $p \in [0,1]$. This gives the coloring by symmetry discussed previously. This two-coloring is symmetrically non-degenerate provided that $a_1 \neq a_2$. It is neither strict nor dynamically non-degenerate. The same properties hold if instead we take $\mathbf{C}_{\mathbf{a}}(n,p) = p^2a_1 + (1-p)^2a_2$. On the other hand, the map $\mathbf{C}_{\mathbf{a}}(n,p) = p^2a_1 + (1-p)a_2$ is not a 2-parameter 2-coloring algorithm as it does not satisfy (1). ♡

We conclude by noting that there are many ways one can construct effective 2-color algorithms. For examples of 2-color quilts colored using an algorithm that gives a painted effect, we refer to the images 'Study in velvet' (Figure 12) at the *URL*: `nothung.math.uh.edu/~mike/bridges.html` and 'Sydney Walls', at the *URL*: `nothung.math.uh.edu/~mike/Art/Art.html`.

References

[1] Pascal Chossat and Martin Golubitsky, *Symmetry increasing bifurcations of chaotic attractors*, Physica D **32** (1988), 423–436.
[2] M J Field, *Color Symmetries in Chaotic Quilt Patterns*, to appear in Proc. ISAMA 99, San Sebastian, Spain, 1999.
[3] M J Field and M Golubitsky, *Symmetry in Chaos*, Oxford University Press, November, 1992.
[4] István Hargittai and Magdolna Hargittai, *Symmetry, a unifying concept*, Shelter Publications, Inc., Bolinas, California, 1994.
[5] Johannes Kepler, *Mysterium Cosmographicum*, 1595 (See also [4, p 95] or [8]).
[6] Roger Penrose, *The Emperor's New Mind*, Oxford University Press, New York – Oxford, 1989.
[7] D Washburn and D Crowe. *Symmetries of Culture*, University of Washington Press, 1988.
[8] Hermann Weyl, *Symmetry*, Princeton University Press, Princeton, New Jersey, 1952.