Math Article Review Symmetries of Fractal Tilings

Crista Moreno

May 10, 2009

Contents

1 Introduction **2**

2 Fractals **2**

2.1 Fractal Fundamentals 2

2.2 Lévy Dragon 6

3 Tiling **7**

3.1 Square Tiling 7

3.2 Triangle Tiling 14

3.3 Underlying Mathematics in Tiling 17

4 Julia Sets (Fractals and Complex Analysis) **19**

5 Conclusion **20**

References **20**

Appendix A **21**

1 Introduction

This review is a class project for Professor James Morrow's undergraduate honors mathematics course at the University of Washington in the spring of 2009. It was an honor and pleasure to be a student in Professor Morrow's course, and I thank him for inspiring me to study mathematics.

This review explores fractal tiling of the plane as discussed in *Symmetries of Fractal Tilings* by Palagallo & Salcedo (2008) [4]. For a more in depth read on fractals the reader is highly encouraged to look into *The Fractal Geometry of Nature* by Benoît B. Mandelbrot. Section 2.1 examines the definition of a fractal, and gives some mathematical background from set theory and point set topology. Section 2.2 gives the algorithm for producing the Lévy Dragon Fractal. Section 3.1 investigates the Pinwheel Fractile tiling of the plane Palagallo & Salcedo [4].

2 Fractals

2.1 Fractal Fundamentals

In the late 20^{th} century, mathematician Benoît Mandelbrot introduced the study of fractals. A fractal is a complex geometric figure that continues to display self-similarity when viewed on all scales. Mandelbrot developed the idea of *fractional dimension*, and coined the term *fractal*. One of the fundamental characteristics of fractals is that the length of its boundary is infinite, but its area is finite. The examples of fractals presented in this review appear strange and exotic, but fractals are in fact inherent in nature. They appear in the formation of clouds, mountain ranges, trees etc. [5]. In this review we provide the mathematical background for understanding the definition of a fractal, and then give the algorithm for constructing fractals in the plane. Let us commence with the formal definition of a fractal as stated by Mandelbrot [3].

Definition 1. *A **fractal** is a set for which the Hausdorff Besicovitch dimension (dimension of a fractal) D strictly exceeds the topological dimension D_T. Where D_T is always an integer and every set with a noninteger D is a fractal.*

In order to understand Definition 1, we provide some background from set theory and point set topology.

Definition 2. *A space X is a set. The points of the space are the elements of the set.*

Definition 3. *Let X be a non-empty set. A real-valued function d defined on $X \times X$, i.e. ordered pairs of elements in X, is called a **metric** or **distance function** on X if and only if it satisfies, for every $a, b, c \in X$ the following axioms:*

***M1**: $d(a, b) \geq 0$ and $d(a, a) = 0$*

***M2**: (Symmetry) $d(a, b) = d(b, a)$*

***M3**: (Triangle Inequality) $d(a, c) \leq d(a, b) + d(b, c)$*

***M4**: If $a \neq b$, then $d(a, b) > 0$*

Definition 4. *A metric space (X, d) is **complete** if every Cauchy sequence $\{x_n\}_{n=1}^{\infty}$ in X has a limit $x \in X$.*

Definition 5. *Let (X, d) be a complete metric space. The $\mathscr{H}(X)$ denotes the space whose points are the compact subsets of X, other than the empty set.*

Definition 6. *A family of sets $\mathscr{C} = \{C_i\}_{i \in I}$ is said to be a **covering** of a set E (or to cover E) if*

$$E \subset \bigcup_{i \in I} C_i$$

.

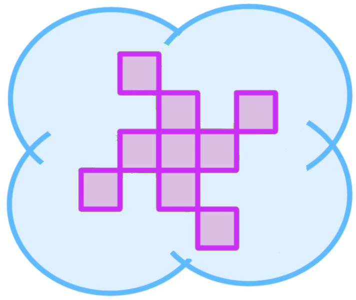


Figure 1: Covering

Definition 7. *Let* (X, d) *be a complete metric space. Let* $A \in \mathscr{H}(X)$. *Let* $\mathscr{N}(\epsilon)$ *denote the minimum number of balls of radius* ϵ *needed to cover* A. *If*

$$D = \lim_{\epsilon \to 0} \left(\sup \left(\frac{\ln(\mathscr{N}(\epsilon'))}{\ln(1/\epsilon')} : \epsilon' \in (0, \epsilon) \right) \right)$$

exists, then D *is called the* ***fractal dimension*** *of* A.

A topology for a set X is a family $\mathscr{T}$ of open subsets belonging to X, such that the null set, X, and the union of an arbitrary number of open sets, and the intersection of finitely many open sets, are open (see Definition 8).

Definition 8. *Let* X *be a nonempty set. A collection* $\mathscr{T}$ *of subsets of* X *is a* ***topology*** *on* X *if and only if* $\mathscr{T}$ *satisfies the following axioms:*

O1: $\emptyset$ *(empty set) and* X *are in* $\mathscr{T}$.

O2: *The union of the elements of any subcollection of* $\mathscr{T}$ *is in* $\mathscr{T}$.

O3: *The intersection of the elements of any finite subcollection of* $\mathscr{T}$ *is in* $\mathscr{T}$.

The members of $\mathscr{T}$ *are then called* $\mathscr{T}$***-open sets****, or simply* ***open sets****. The pair* $(X, \mathscr{T})$ *is called a* ***topological space****.*

Example 1. *The collection of sets* $\mathscr{T}$ *in Figure 2 form a topology on the set* X.

$$X = \{u, v, w\},\ \mathscr{T} = \{X, \emptyset, \{u\}, \{v\}, \{u, v\}\}$$

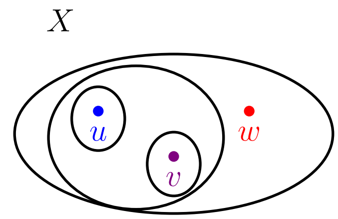


Figure 2: Topology

The topological space $(X, \mathscr{T})$ has a *topological dimension* D_T if, for every *covering* $\mathscr{C}$, has a refinement $\mathscr{C}'$ such that $\forall x \in X$ occurs in at most $D_T + 1$ sets in $\mathscr{C}'$, and D_T is the smallest such integer.

Definition 9. *A **refinement of a covering** $\mathscr{C}$ of E is another covering $\mathscr{C}'$ of E such that each set C'_j in $\mathscr{C}'$ is contained in some set belonging to $\mathscr{C}$.*

Example 2. *Figure 3 displays a refinement, from the union of blue colored sets to the union of green colored sets, of the covering of the set in purple.*

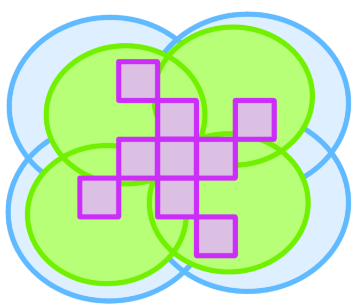
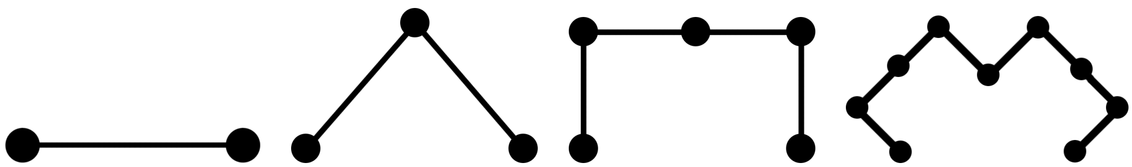


Figure 3: Refinement of a Covering

The word *fractal* is derived from the Latin word *frāctus*, meaning broken, shattered, or having been broken. This is appropriate because the meaning we wish to preserve is "irregular fragments". The fractal dimension gives a way to compare fractals [1].

A more intuitive way of interpreting the fractal dimension is to consider the geometric figure of a broken curve, where the number of breaks of the curve is $\mathscr{N}(\epsilon)$ and the length of each piece is $1/\epsilon$.

Example 3. *In Figure 4(b) there are two breaks, and the length of the two pieces are both $\sqrt{1/2}$ the size of the original length Figure 4(a). Here the fractal dimension of the Lévy Dragon fractal, displayed in Figure 5, is* $D = \dfrac{\log(2)}{\log\left(\sqrt{\frac{1}{2}}\right)} = 2.$



(a) level 1 (b) level 2 (c) level 3 (d) level 4

Figure 4: Lévy Dragon levels

2.2 Lévy Dragon

The Lévy Dragon Fractal, also known as the Lévy C Curve invented by a French mathematician Paul Pierre Lévy, is constructed from the base pattern +**F**- -**F**+. Where + means to turn 45°, **F** means to draw a line, and **-** means to turn −45°. This pattern expanded recursively produces the beautiful symmetric fractal illustrated in Figure 5. The fractal was generated in Java, and the color was changed every few iterations to help display the growth of the fractal.

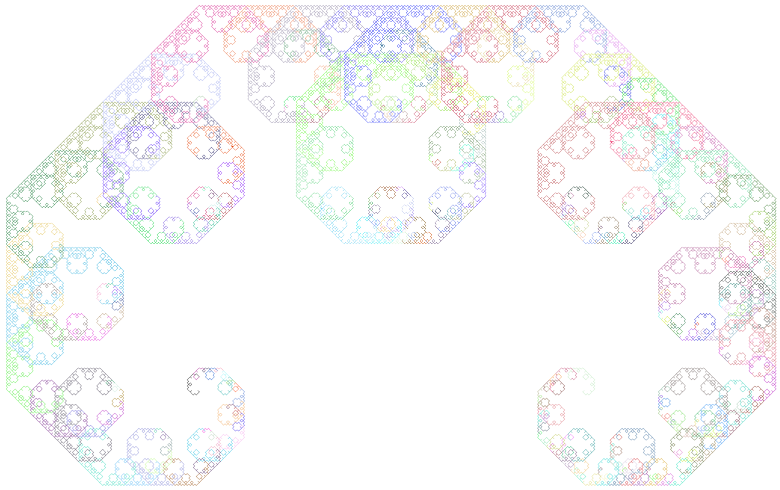


Figure 5: Lévy Dragon Fractal

3 Tiling

3.1 Square Tiling

To introduce the concepts discussed in *Symmetries of Fractal Tilings* [4] this section will first consider the Pinwheel Fractal tiling of the Euclidean plane $\mathbb{R}^2$.

Definition 10. *A **tiling of the plane** $\mathbb{R}^2$ is a countable family $\{A_i\}$ of compact sets that cover the plane with $int(A_i) \cap int(A_j) = \emptyset$ for $i \neq j$.*

For a given number of bounded sets, the interior of each set does not intersect the interior of any other sets, i.e. the tiling will have no over lap.

For the first example, the plane is tiled with closed unit squares. Each square is labelled $1 - 9$ starting from the lower left corner and ending at the top right corner as in Figure 6.

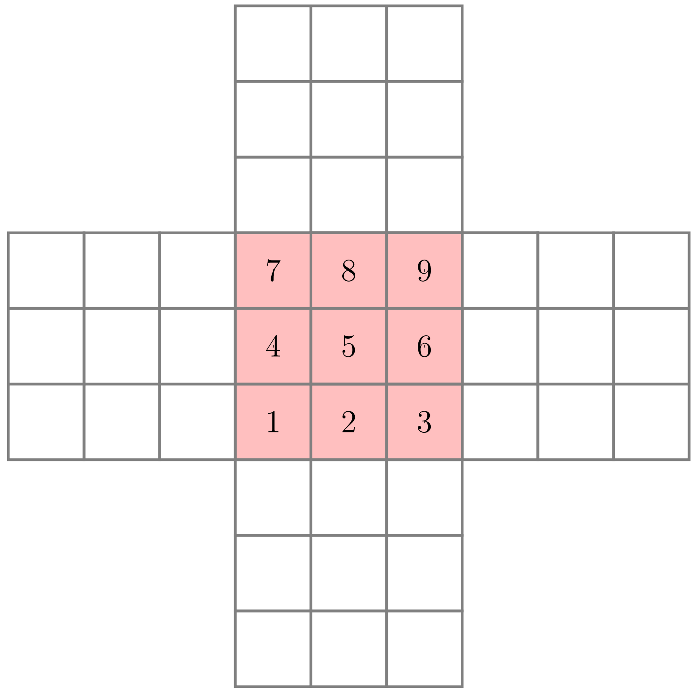


Figure 6: Tiling of the plane with unit squares.

Next the square tiles 1, 3, 7, and 9 are translated to their corresponding position in the square adjacent to their original square as shown in Figure 7.

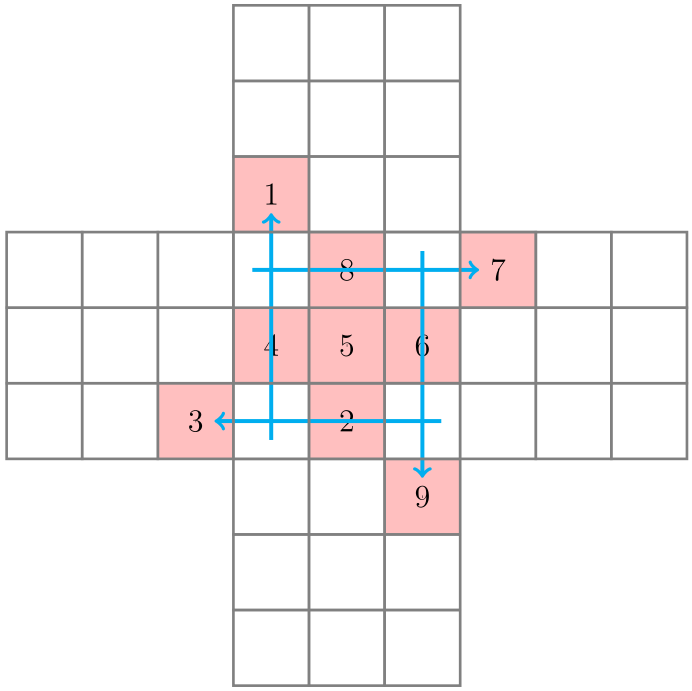
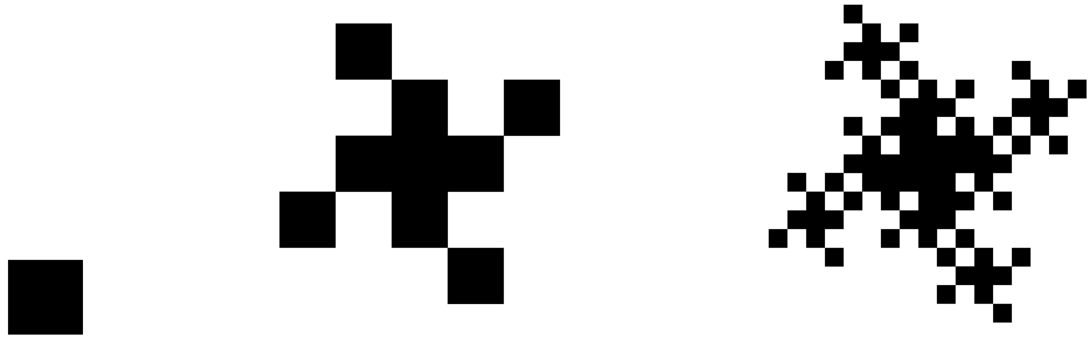


Figure 7: Pinwheel Fractal Pattern

Figure 7 displays the Level 1 pattern for the Pinwheel Fractal. This pattern is allowable; a necessary property for tiling, because otherwise the resulting fractal would have overlapping tiles and thus would not be self-similar.

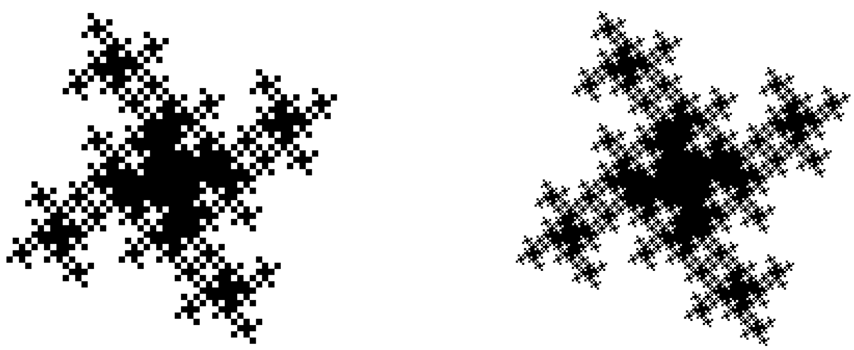
Definition 11. *An **allowable pattern** is a pattern that contains each tile, represented by a number, in the plane exactly once.*

This process is then repeated for each individual pink square resulting in 81 squares, each one ninth the size of the original square as shown in Figure 8(c).



(a) Level 0 (b) Level 1 (c) Level 2

Figure 8: Pinwheel Fractal Stages



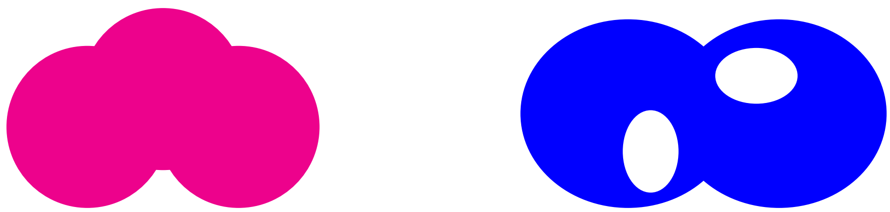
(a) Level 3 (b) Tile T

Figure 9: Pinwheel Fractal Stages

The process is repeated indefinitely, and with each iteration the area approaches its limiting tile T, Figure 9(b), which has a side length of $3\sqrt{\frac{10}{4}}$. The tile T is *connected* because it is composed of connected sets, but does not have a connected region.

Definition 12. *A topological space X is said to be* ***connected*** *if and only if there does not* *exist a pair of open nonempty subsets E and F such that $E \cap F = \emptyset$ and $E \cup F = X$.*

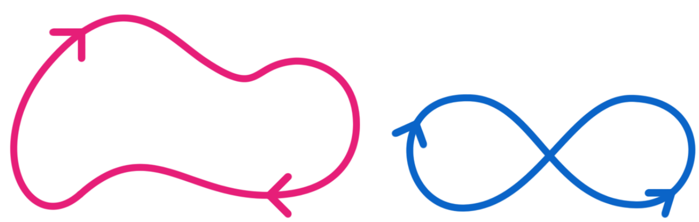
Definition 13. *A region R is **simply connected** if it has no holes; all closed curves can be shrunk to a point without passing through points in R^c.*



(a) Simply Connected (b) Not Simply Connected

Example 4.

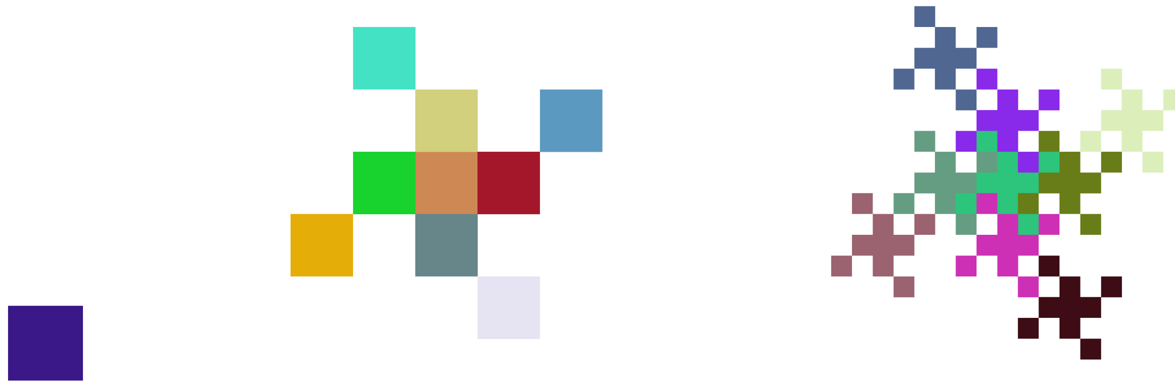
Definition 14. *A closed curve C is **simple** if it does not intersect itself.*



(c) Simple (d) Not Simple

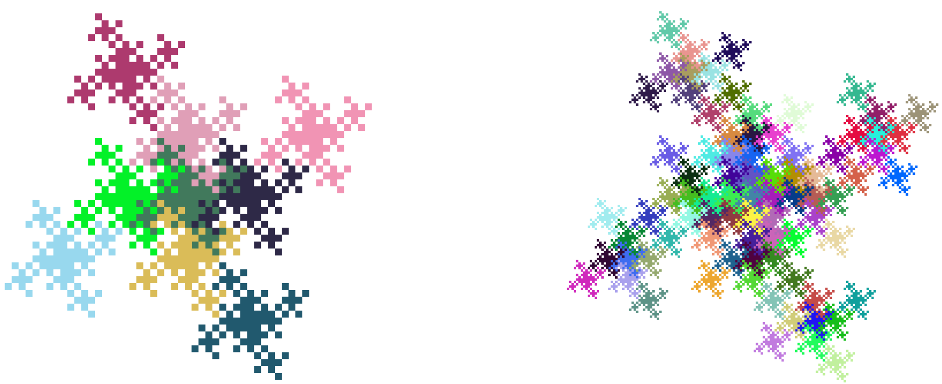
Example 5.

For each iteration the color is changed to show that no square coincides with another, and that the self-similarity of the fractal is preserved.



(e) Level 0 (f) Level 1 (g) Level 2

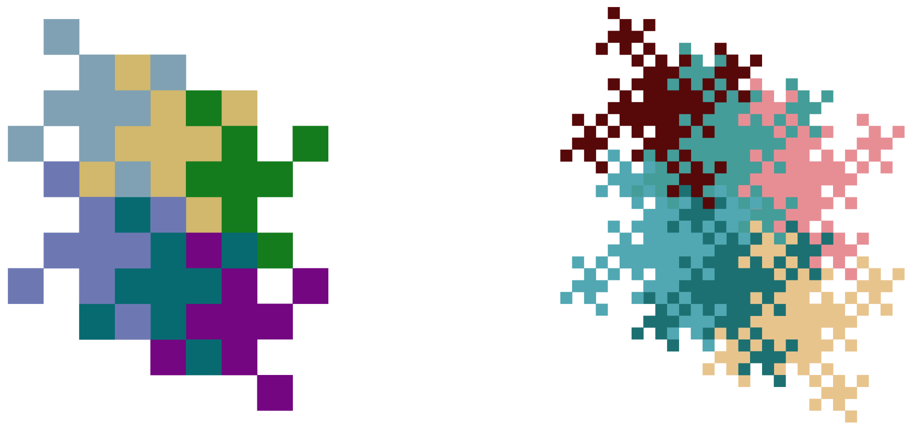
Figure 10: Pinwheel Fractal Stages



(a) Level 3 (b) Level 4

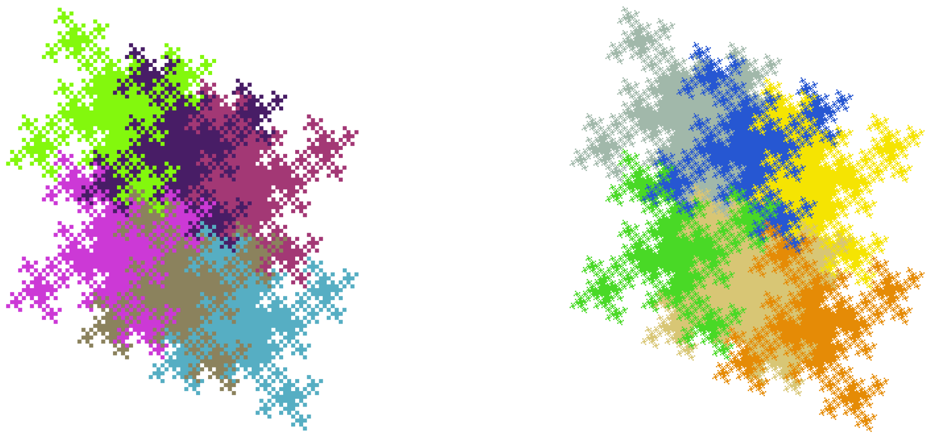
Figure 11: Pinwheel Fractal Stages

The objective in constructing these fractiles is to tile the plane $\mathbb{R}^2$. After a finite number of iterations the fractile reaches its limiting area. The four identical fractals are then fit together onto its boundary as shown in Figures 12(a), 12(b), 13(a), and 13(b). Looking at each of the stages, having restrained our fractile to be made up of an allowable pattern has paid off. Zooming in on the figures, each level 1 tile has preserved the pattern and the plane is completely tiled.



(a) Level 2 (b) Level 3

Figure 12: Pinwheel Fractal Tiling the Euclidean plane



(a) Level 4 (b) Level 5

Figure 13: Pinwheel Fractal Tiling the Euclidean Plane

3.2 Triangle Tiling

Presented here is my own fractal tiling of the plane using triangles. We begin with a triangle tiling of the plane. As with the square tiling, triangle tiling requires an allowable pattern. In Figure 14 the triangle outlined in black will serve as our base figure. In this triangle there are nine inner triangles. The triangles labelled 1, 2, 4, 5, 7, and 9 are translated outward onto their corresponding positions in the triangles adjacent to the base triangle, as shown in Figure 14. The process is then repeated for each of the triangles colored in pink. The second iteration produces the image in Figure 15(c) and so on.

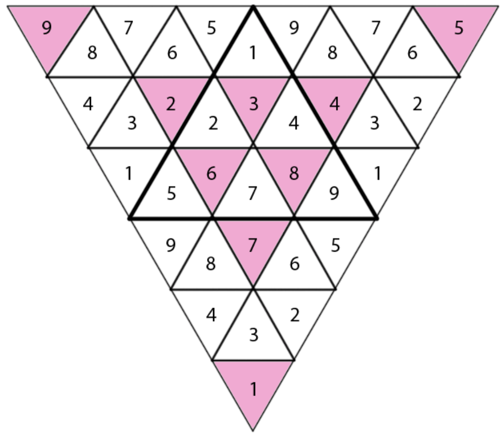
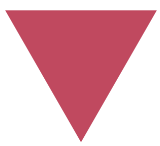
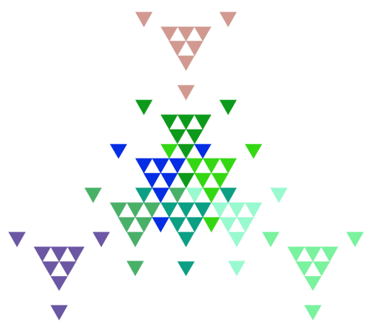


Figure 14: Triangle Pattern



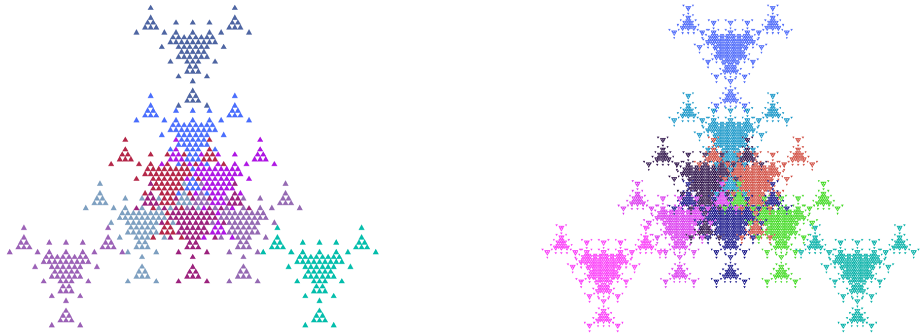
(a) Level 0

(b) Level 1



(c) Level 2

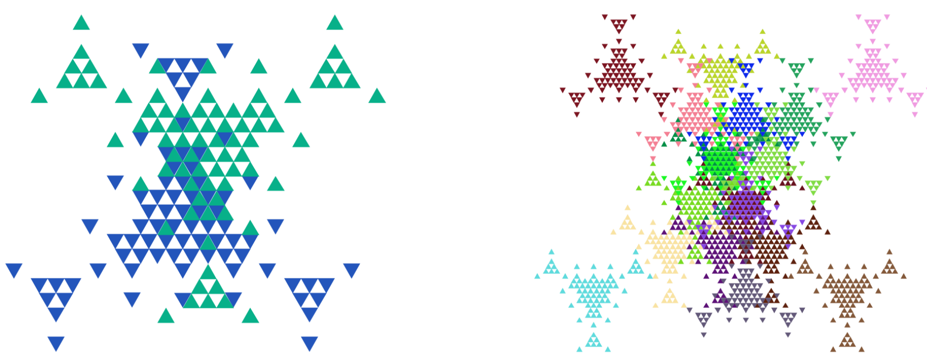
Figure 15: Triangle Fractal Stages



(a) Level 3 (b) Level 4

Figure 16: Triangle Fractal Stages

As with square tiling, the triangle fractal approaches a limiting area. We repeat the same process as with the square tiling, but this time because the triangle fractal has more symmetries, every adjacent fractile has to be rotated 60°. In each of the Figures 17(a), 17(b), and 18 notice that there are two triangle fractiles fitted together without any overlap.



(a) Level 2 (b) Level 2

Figure 17: Triangle Fractal Tiling of $\mathbb{R}^2$

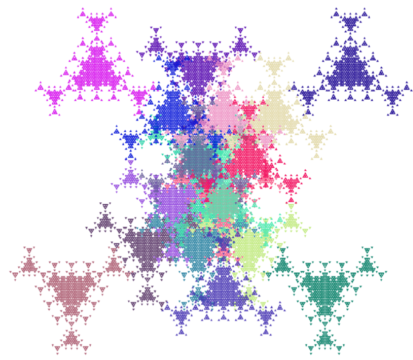


Figure 18: Triangle Fractal Tiling of $\mathbb{R}^2$

3.3 Underlying Mathematics in Tiling

To tile the entire $\mathbb{R}^2$ plane we need to think in two dimensions. Let M be a (2×2) square matrix with integer entries, where the columns represent the scaling factors of our tile, then the inverse M^{-1} is a *contractive mapping*.

$$M = \begin{bmatrix} n & 0 \\ 0 & n \end{bmatrix} \tag{1}$$

Definition 15. *A transformation $f : X \to X$ on a metric space (X, d) is called* ***contractive mapping*** *or a* ***contraction mapping*** *if there is a constant $0 \le s < 1$ such that*

$$d(f(x), f(y)) \le s * d(x, y) \quad \forall x, y \in X.$$

The number s is called a ***contractivity factor*** *of f.*

Then for $j = 1, \dots n^2$ J. Palagallo & M. Salcedo then define the mappings

$$f_j \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{bmatrix} 1/n & 0 \\ 0 & 1/n \end{bmatrix} * \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + r_j$$

so that the square is scaled to 1/9 the original size of both the x and y components. The addition of r_j serves to translate the image. Its initial integer coordinates (x, y) given as the lower left corner of each of the n^2 squares in the selected pattern. The set of functions $\{f_j\}$ and the set of vectors $\{r_j\}$ is special because together they satisfy the requirements such that the *iterated function system* will converge to a compact set, called the *attractor*. In other words the functions will converge to our desired tiling.

Definition 16. *An* ***iterated function system*** *consists of a complete metric space* (X, d) *together with a finite set of contractive mappings* $w_n : X \to X$*, with respect to contractivity factors* s_n*, for* $n = 1, 2, \dots, N$*.*

Theorem 1. *Let* $\{X; w_n, n = 1, 2, ..., N\}$ *be a hyperbolic iterated function system with contractivity factor* s*. Then the transformation* $W : \mathscr{H}(X) \to \mathscr{H}(X)$ *defined by*

$$W(B) = \bigcup_{n=1}^{N} w_n(B)$$

for all $B \in \mathscr{H}(X)$ *is a contraction mapping on the complete metric space* $\mathscr{H}(X, h(d))$ *with contractivity factor* s*. That is*

$$h(W(B), W(C)) \leq s * h(B, C)$$

for all $B, C \in \mathscr{H}(X)$*. Its unique fixed point,* $A \in \mathscr{H}(X)$*, obeys*

$$A = W(A) = \bigcup_{n=1}^{N} w_n(A),$$

and is given by $A = \lim_{n\to\infty} W^{on}(B)$ *for any* $B \in \mathscr{H}(X)$

Definition 17. *The fixed point* $A \in \mathscr{H}(X)$ *described in Theorem 1 is called the* ***attractor*** *of the iterated function system.*

4 Julia Sets (Fractals and Complex Analysis)

Let $f(x)$ be an analytic function that maps the extended complex plane $\mathbb{C}^*$ onto itself, and let $R(z) = P(z)/Q(z)$ where $x \in \mathbb{C}^*$. The *Julia set* is the set of points of the iteration of $f(z)$, $f(f \cdots f(z) \cdots))$, n times, $n = 1, 2, 3$,... The *Fatou set* of $f(z)$, denoted by $\mathcal{F} = \mathcal{F}(z)$, is all the points in the extended complex plane that have an open neighborhood U such that the iterations of $f(z)$ to U form a normal family of analytic functions on U. The Julia set is the complement of the Fatou set, and is closed. Since these two sets are complementary of each other on the extended plane, we have the following result

Theorem 2. *The Fatou set and and the Julia set of a rational function* $f(z)$ *are invariant, that is,* $f(\mathcal{F}) \subseteq \mathcal{F}$ *and* $f(\mathcal{J}) \subseteq \mathcal{J}$

Fundamental Properties of Julia Sets

- $J_R \neq \emptyset$ and contains more than countable many points.
- The Julia sets of R and R^k, $k = 1, 2, 3, ...$, are identical.
- $R(J_R) = J_R = R^{-1}(J_R)$.
- $\forall\ x \in J_R$ the inverse orbit $O_r^{-1}(x)$ is dense in J_R.
- If γ is an attractive cycle of R, then $A(\gamma) \subset F_R = \{\mathbb{C} \cup \infty\} - J_R$ and $\partial A(\gamma) = J_R$

5 Conclusion

Palagallo and Salcedo's paper on fractal tiling of the plane gives an artistic and colorful example of one of the areas of application for fractals. In building the code for the Lévy Dragon and other fractals, I was amazed at how a recursive algorithm of such a simple pattern could produce such complex figures that explain very different phenomena. Fractals also play a big role in complex analysis, which has numerous applications in physics and engineering. Fractal geometry has potential to expand and take foot hold into many areas of mathematics and the life sciences.

References

[1] Michael Barnsley. *Fractals everywhere.* Harcourt Brace Jovanovich, San Diego, 1988.

[2] J. L. Kelley and Isaac Namioka. *Linear Topological Spaces*, chapter 2, page 27. D. VAN NOSTRAND COMAPNY, INC., Princeton, New Jersey, 1963.

[3] Benoit B. Mandelbrot. *The fractal geometry of nature.* W. H. Freeman and Company, New York, 1977.

[4] Judith Palagallo and Maria Selcedo. Symmetries of fractal tilings. *World Scientific*, 16(1):69–78, 2008.

[5] Dr. Heinz-Otto Peitgen and Dr. Peter H. Richter. *The beauty of fractals*, chapter 2, pages 27–29. Springer-Verlag, Berlin, 1986. Images of Complex Dynamical Systems.

[6] Stuart Reges and Marty Stepp. *Building Java programs.* Pearson Addison Wesley, Boston, 2008. A Back to Basics Approach.

Appendix A: JAVA Fractal Generators

Lévy Dragon

```java
/*
 * Crista Moreno 05/21/09
 * Produces Levy Dragon Fractal.
 */
import java.awt.*;
import java.util.*;

public class LevyDragon {
    public static void main(String[] args) {
        DrawingPanel panel = new DrawingPanel(10000/3, 6000/3);
        Graphics g = panel.getGraphics();
        drawInstructions(buildInstructions(15), g);
        panel.save("LevyDragon.png");
    }

    public static String buildInstructions(int numberOfItr) {
        String axiom = "F";
        for (int i = 0; i < numberOfItr; i++) {
            axiom = "+" + axiom + "--" + axiom + "+";
        }
        return axiom;
    }

```

```java
    public static void drawInstructions(String instructions, Graphics g) {
        Random r = new Random();
        Color custom = new Color(0, 0, 0);
        double lLength = 25/3;
        double currentAngle = 0;
        double x = 2700.0/3;
        double y = 4700.0/3;
        for (int i = 0; i < instructions.length(); i++) {
            char step = instructions.charAt(i);
            switch (step) {
                case 'F':
                    double x2 = x + (lLength * Math.cos(currentAngle));
                    double y2 = y + (lLength * Math.sin(currentAngle));
                    if (i%40 == 0)
                        custom = new Color(r.nextInt(254), r.nextInt(254),
                                r.nextInt(254));
                    g.setColor(custom);
                    g.drawLine((int) Math.round(x), (int) Math.round(y),
                            (int) Math.round(x2), (int) Math.round(y2));
                    x = x2;
                    y = y2;
                    break;
                case '+':
                    currentAngle += -Math.PI/4;
                    break;
                case '-':
```

```
                    currentAngle += Math.PI/4;
                    break;
                }
            }
        }
    }
```

Square Tiling

```
/*
 * Crista Moreno 05/21/09
 * Produces Square Tiling Pinwheel Fractal.
 */
import java.awt.*;
import java.util.*;

public class pinwheel {
    public static void main(String[] args) {
        DrawingPanel panel = new DrawingPanel(2000, 2000);
        Graphics g = panel.getGraphics();
        drawInstructions(g);
        // panel.save("Pinwheel.png");
    }

    public static void drawInstructions(Graphics g) {
        double x = 650;
        double y = 650;
```

```java
        double length = 700.0;
        Random r = new Random();
        drawInstructions(g, x, y, length, 3, r);
    }

    public static void drawInstructions(Graphics g, double x, double y,
            double length, int itr, Random r) {
        if (itr == 3) {
            g.setColor(new Color(r.nextInt(254), r.nextInt(254), r.nextInt(25
        }
        if (itr == 1) {
            g.fillRect((int) Math.floor(x), (int) Math.floor(y),
                    (int) Math.ceil(length), (int) Math.ceil(length));
        } else {
            length = length/3;
            itr -= 1;
            x = x + length;
            y = y + length;
            drawInstructions(g, x, y, length, itr, r);
            drawInstructions(g, x, y - length, length, itr, r);
            drawInstructions(g, x + length, y, length, itr, r);
            drawInstructions(g, x, y + length, length, itr, r);
            drawInstructions(g, x - length, y, length, itr, r);
            drawInstructions(g, x + 2*length, y - length, length, itr, r);
            drawInstructions(g, x - length, y - 2*length, length, itr, r);
            drawInstructions(g, x - 2*length, y + length, length, itr, r);
```

```
            drawInstructions(g, x + length, y + 2*length, length, itr, r);
        }
    }
}
```

Triangle Tiling

```
/*
 * Crista Moreno 05/24/09
 * Produces Example Triangle Tiling Fractal.
 */
import java.awt.*;
import java.util.*;

public class triangleFractal {
    public static final int SIZE = 500;
    public static final int LEVEL = 3;

    public static void main(String[] args) {
        DrawingPanel panel = new DrawingPanel(1000, 750);
        Graphics g = panel.getGraphics();
        drawInstructions(g);
        panel.save("Triangle_Fractal_level3.png");
    }

    public static void drawInstructions(Graphics g) {
        double[] x = new double[] {100+250, SIZE-100+250, SIZE/2+250};
```

```
        double[] y = new double[] {120+230, 120+230, SIZE-120+230};
        g.setColor(Color.BLACK);
        Random r = new Random();
        drawInverted(g, x, y, LEVEL, r);
    }

    public static void drawInverted(Graphics g, double[] x, double[] y,
            int itr, Random r) {
        if (itr == 2) {
            g.setColor(new Color(r.nextInt(254), r.nextInt(254),
                r.nextInt(254)));
        }
        if (itr == 0) {
            g.fillPolygon(new int[] {(int) Math.floor(x[0]),
                (int) Math.ceil(x[1]),
            (int) Math.round(x[2])}, new int[] {(int) Math.floor(y[0]),
            (int) Math.floor(y[1]), (int) Math.ceil(y[2])}, 3);
        } else {
            itr -= 1;

            double x2[];
            double y2[];

            double nBase = (x[1]-x[0])/3;
            double nHeight = (nBase/2)*Math.sqrt(3);

```

```
        // upRight triangles

        // triangle 4
        x2 = new double[] {x[0], x[0] + nBase/2, x[0] + nBase};
        y2 = new double[] {y[0] + 2*nHeight, y[0] + nHeight,
            y[0] + 2*nHeight};
        drawUpRight(g, x2, y2, itr, r);

        // triangle 2
        x2 = new double[] {x[1] - nBase, x[1] - nBase/2, x[1]};
        y2 = new double[] {y[1] + 2*nHeight, y[1] + nHeight,
            y[1] + 2*nHeight};
        drawUpRight(g, x2, y2, itr, r);

        // triangle 1
        x2 = new double[] {x[2] - nBase/2, x[2], x[2] + nBase/2};
        y2 = new double[] {y[0] - 2*nHeight, y[0] - 3*nHeight,
            y[0] - 2*nHeight};
        drawUpRight(g, x2, y2, itr, r);

        // triangle 7
        x2 = new double[] {x[2] - nBase/2, x[2], x[2] + nBase/2};
        y2 = new double[] {y[0], y[0] - nHeight, y[0]};
        drawUpRight(g, x2, y2, itr, r);

        // triangle 8
```

```
x2 = new double[] {x[0] + nBase/2, x[0] + nBase, x[2]};
y2 = new double[] {y[0] + nHeight, y[0], y[0] + nHeight};
drawUpRight(g, x2, y2, itr, r);

// triangle 6
x2 = new double[] {x[2], x[2] + nBase/2, x[2] + nBase};
y2 = new double[] {y[0] + nHeight, y[0], y[0] + nHeight};
drawUpRight(g, x2, y2, itr, r);

// triangle 3
x2 = new double[] {x[2] - nBase/2, x[2], x[2] + nBase/2};
y2 = new double[] {y[2] - nHeight, y[2] - 2*nHeight,
    y[2] - nHeight};
drawUpRight(g, x2, y2, itr, r);

// triangle 9
x2 = new double[] {x[2] + 2*nBase, x[2] + 2*nBase +
    nBase/2, x[2] + 3*nBase};
y2 = new double[] {y[2], y[2] - nHeight, y[2]};
drawUpRight(g, x2, y2, itr, r);

// triangle 5
x2 = new double[] {x[2] - 3*nBase, x[2] - 3*nBase +
    nBase/2, x[2] - 2*nBase};
y2 = new double[] {y[2], y[2] - nHeight, y[2]};
drawUpRight(g, x2, y2, itr, r);
```

```
        }
    }

    public static void drawUpRight(Graphics g, double[] x, double[] y,
            int itr, Random r) {
        if (itr == 2) {
            g.setColor(new Color(r.nextInt(254), r.nextInt(254),
                r.nextInt(254)));
        }
        if (itr == 0) {
            g.fillPolygon(new int[] {(int) Math.floor(x[0]),
                (int) Math.round(x[1]), (int) Math.ceil(x[2])},
                new int[] {(int) Math.ceil(y[0]), (int) Math.floor(y[1]),
                (int) Math.ceil(y[2])}, 3);
        } else {
            itr -= 1;

            double x2[];
            double y2[];

            // inverted triangles

            double nBase = (x[2]-x[0])/3;
            double nHeight = (nBase/2)*Math.sqrt(3);

            // triangle 5
```

```
            x2 = new double[] {x[1] + 2*nBase, x[1] + 3*nBase,
                x[1] + 3*nBase - nBase/2};
            y2 = new double[] {y[1], y[1], y[1] + nHeight};
            drawInverted(g, x2, y2, itr, r);

            // triangle 9
            x2 = new double[] {x[1] - 3*nBase, x[1] - 2*nBase,
                x[1] - 2*nBase - nBase/2};
            y2 = new double[] {y[1], y[1], y[1] + nHeight};
            drawInverted(g, x2, y2, itr, r);

            // triangle 3
            x2 = new double[] {x[1] - nBase/2, x[1] + nBase/2, x[1]};
            y2 = new double[] {y[1] + nHeight, y[1] + nHeight,
                y[1] + 2*nHeight};
            drawInverted(g, x2, y2, itr, r);

            // triangle 6
            x2 = new double[] {x[0] + nBase/2, x[1], x[0] + nBase};
            y2 = new double[] {y[0] - nHeight, y[0] - nHeight, y[0]};
            drawInverted(g, x2, y2, itr, r);

            // triangle 8
            x2 = new double[] {x[1], x[1] + nBase, x[1] + nBase/2};
            y2 = new double[] {y[0] - nHeight, y[0] - nHeight, y[0]};
            drawInverted(g, x2, y2, itr, r);
```

```
            // triangle 7
            x2 = new double[] {x[1] - nBase/2, x[1] + nBase/2, x[1]};
            y2 = new double[] {y[0], y[0], y[0] + nHeight};
            drawInverted(g, x2, y2, itr, r);

            // triangle 2
            x2 = new double[] {x[0], x[0] + nBase, x[0] + nBase/2};
            y2 = new double[] {y[1] + nHeight, y[1] + nHeight,
                y[1] + 2*nHeight};
            drawInverted(g, x2, y2, itr, r);

            // triangle 4
            x2 = new double[] {x[1] + nBase/2, x[2], x[2] - nBase/2};
            y2 = new double[] {y[1] + nHeight, y[1] + nHeight,
                y[1] + 2*nHeight};
            drawInverted(g, x2, y2, itr, r);

            // triangle 1
            x2 = new double[] {x[1] - nBase/2, x[1] + nBase/2,  x[1]};
            y2 = new double[] {y[0] + 2*nHeight, y[0] + 2*nHeight,
                y[0] + 3*nHeight};
            drawInverted(g, x2, y2, itr, r);
        }
    }
}
```

DrawingPanel [6]

```java
/*
Stuart Reges and Marty Stepp

07/01/2005

The DrawingPanel class provides a simple interface for drawing persistent
images using a Graphics object. An internal BufferedImage object is used
to keep track of what has been drawn. A client of the class simply
constructs a DrawingPanel of a particular size and then draws on it with
the Graphics object, setting the background color if they so choose.

To ensure that the image is always displayed, a timer calls repaint at
regular intervals.
*/

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

public class DrawingPanel implements ActionListener {
    public static final int DELAY = 250;  // delay between repaints in millis
```

```
    private static final String DUMP_IMAGE_PROPERTY_NAME = "drawingpanel.save
    private static String TARGET_IMAGE_FILE_NAME = null;
    private static final boolean PRETTY = true;  // true to anti-alias
    private static boolean DUMP_IMAGE = false;  // true to write DrawingPanel
                                                // to file
    private int width, height;    // dimensions of window frame
    private JFrame frame;         // overall window frame
    private JPanel panel;         // overall drawing surface
    private BufferedImage image;  // remembers drawing commands
    private Graphics2D g2;        // graphics context for painting
    private JLabel statusBar;     // status bar showing mouse position
    private long createTime;

    static {
        TARGET_IMAGE_FILE_NAME = System.getProperty(DUMP_IMAGE_PROPERTY_NAME)
        DUMP_IMAGE = (TARGET_IMAGE_FILE_NAME != null);
    }

    // construct a drawing panel of given width and height enclosed in a wind
    public DrawingPanel(int width, int height) {
        this.width = width;
        this.height = height;
        this.image = new BufferedImage(width, height,
            BufferedImage.TYPE_INT_ARGB);
        this.statusBar = new JLabel(" ");
        this.statusBar.setBorder(BorderFactory.createLineBorder(Color.BLACK))
```

```
        this.panel = new JPanel(new FlowLayout(FlowLayout.CENTER, 0, 0));
        this.panel.setBackground(Color.WHITE);
        this.panel.setPreferredSize(new Dimension(width, height));
        this.panel.add(new JLabel(new ImageIcon(image)));

        // listen to mouse movement
        MouseInputAdapter listener = new MouseInputAdapter() {
            public void mouseMoved(MouseEvent e) {
                DrawingPanel.this.statusBar.setText("(" + e.getX() + ", "
                    + e.getY() + ")");
            }

            public void mouseExited(MouseEvent e) {
                DrawingPanel.this.statusBar.setText(" ");
            }
        };

        this.panel.addMouseListener(listener);
        this.panel.addMouseMotionListener(listener);
        this.g2 = (Graphics2D)image.getGraphics();
        this.g2.setColor(Color.BLACK);

        if (PRETTY) {
            this.g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
                RenderingHints.VALUE_ANTIALIAS_ON);
            this.g2.setStroke(new BasicStroke(1.1f));
```

```
        }

        this.frame = new JFrame("Drawing Panel");
        this.frame.setResizable(false);
        this.frame.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                if (DUMP_IMAGE) {
                    DrawingPanel.this.save(TARGET_IMAGE_FILE_NAME);
                }
                System.exit(0);
            }
        });

        this.frame.getContentPane().add(panel);
        this.frame.getContentPane().add(statusBar, "South");
        this.frame.pack();
        this.frame.setVisible(true);
        if (DUMP_IMAGE) {
            createTime = System.currentTimeMillis();
            this.frame.toBack();
        } else {
            this.toFront();
        }

        // repaint timer so that the screen will update
        new Timer(DELAY, this).start();
```

```
    }

    // used for an internal timer that keeps repainting
    public void actionPerformed(ActionEvent e) {
        this.panel.repaint();
        if (DUMP_IMAGE && System.currentTimeMillis() >
                createTime + 4 * DELAY) {
            this.frame.setVisible(false);
            this.frame.dispose();
            this.save(TARGET_IMAGE_FILE_NAME);
            System.exit(0);
        }
    }

    // obtain the Graphics object to draw on the panel
    public Graphics2D getGraphics() {
        return this.g2;
    }

    // set the background color of the drawing panel
    public void setBackground(Color c) {
        this.panel.setBackground(c);
    }

    // show or hide the drawing panel on the screen
    public void setVisible(boolean visible) {
```

```
        this.frame.setVisible(visible);
    }

    // makes the program pause for the given amount of time,
    // allowing for animation
    public void sleep(int millis) {
        try {
            Thread.sleep(millis);
        } catch (InterruptedException e) {}
    }

    // take the current contents of the panel and write them to a file
    public void save(String filename) {
        String extension = filename.substring(filename.lastIndexOf(".") + 1);

        // create second image so we get the background color
        BufferedImage image2 = new BufferedImage(this.width, this.height,
            BufferedImage.TYPE_INT_RGB);
        Graphics g = image2.getGraphics();
        g.setColor(panel.getBackground());
        g.fillRect(0, 0, this.width, this.height);
        g.drawImage(this.image, 0, 0, panel);

        // write file
        try {
            ImageIO.write(image2, extension, new java.io.File(filename));
```

```
            } catch (java.io.IOException e) {
                System.err.println("Unable to save image:\n" + e);
            }
        }

        // makes drawing panel become the frontmost window on the screen
        public void toFront() {
            this.frame.toFront();
        }
}
```