Math Article Review Symmetries of Fractal Tilings

Crista Moreno

June 3, 2009

Contents

1 Introduction

This review explores tiling of the plane using square and triangle fractiles discussed in Palagallo & Salcedo (2008) [4].

2 Fractals

2.1 Fractal Fundamentals

The study of fractals is fairly new, introduced in the late 20^{th} century by Benoît Mandelbrot, a French mathematician. He developed the idea of *fractional dimension*, and coined the term *fractal*. A fractal is a complex geometric figure that continues to display self-similarity when viewed on all scales. One of the fundamental characteristics of fractals is that the length of its boundary is infinite, but its area is finite. Surprisingly enough, though the images we see seem strange and exotic, fractals are inherent in nature. In the formation of clouds, mountain ranges, and even trees, fractals are all around us [5]. 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 the definiton of the *topological dimension* we must return to set theory. Where the 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.

Example 1. *Looking at **Figure 1**, note that both diagrams* 5 *and* 6 *are crossed out because they are not topologies. In diagram* 5 *the union of the elements v and w is missing and in diagram* 6 *the intersection v is not included.*

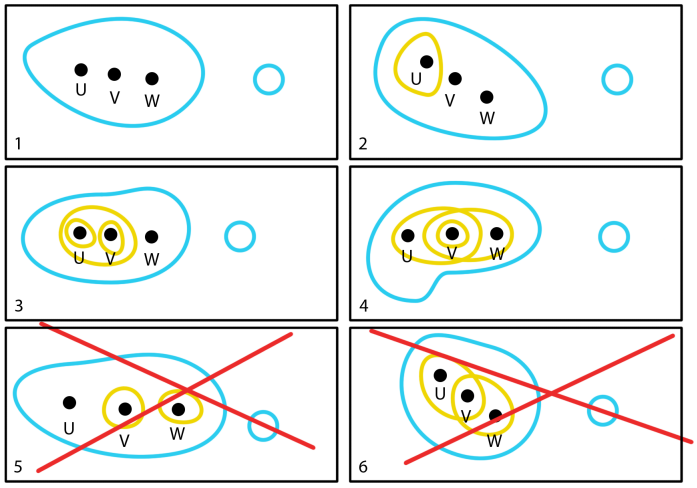


Figure 1: Topology

The set X with topology $\mathscr{T}$, is a topological space, denoted by $(X, \mathscr{T})$ [2]. The topological space X 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 2. *A covering of a subset S is is a set of open sets $\mathcal{C} = \{\mathcal{C}_1, \mathcal{C}_2, ..., \mathcal{C}_n\} \in X$ whose union $\mathcal{C}_1 \cup \mathcal{C}_2 \cup \cdots \cup \mathcal{C}_n \supset S$.*

Definition 3. *A refinement of a covering $\mathscr{C}$ of S is another covering $\mathscr{C}'$ of S such that each set B in $\mathscr{C}'$ is contained in some set belonging to $\mathscr{C}$.*

Example 2. ***Figure 2*** *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.*

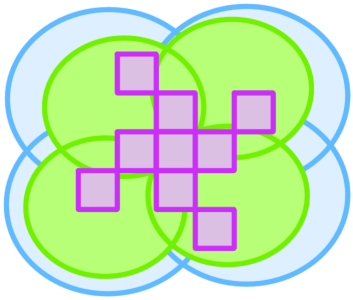


Figure 2: Refinement of a Covering

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 preseve is "irregular fragments" which very well suits the origin of this word. To study fractals we need a way in order to compare them to each other, thus we have the *fractal dimension* [1].

Definition 4. *Let $(\mathbb{X}, d)$ 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 A.

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

Definition 6. *A metric space $(\mathbb{X}, d)$ is complete if every Cauchy sequence $\{x_n\}_{n=1}^{\infty}$ in $\mathbb{X}$ has a limit $x \in \mathbb{X}$.*

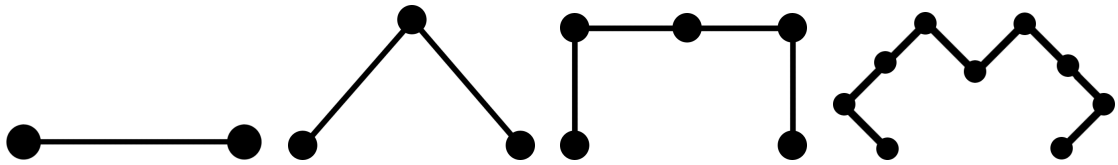
Definition 7. *A metric space (X, d) is a space X together with a real valued function $d : X \; x \; X \to \mathbb{R}$, which measures the distance between pairs of points x and y in X. The following are axioms for d:*

1. $d(x, y) = d(y, x) \quad \forall x, y \in X$
2. $0 < d(x, y) < \infty \quad \forall x, y \in X, x \neq y$
3. $d(x, x) = 0 \quad \forall x \in X$
4. $d(x, y) \leq d(x, z) + d(z, y) \quad \forall x, y, z \in X$

Definition 8. *A space X is a set. The points of the space are the elements of the set.*

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. *If we look at* ***Figure 3(b)*** *we notice that there are two breaks, and that the length of the two pieces are both* $\sqrt{1/2}$ *the size of the original length* ***Figure 3(a)****. Thus, the dimension of the Lévy Dragon fractal, displayed in* ***Figure 4****, 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 3: Lévy Dragon levels

2.2 Lévy Dragon

Figure 4 displays a Lévy Dragon Fractal, also known as the Lévy C Curve generated by a French mathematician Paul Pierre Lévy. The base pattern used to produce this fractal is *+F- -F+*. Where *+* means to turn 45°, *F* means to draw a line, and *-* means to turn -45°. This basic pattern expanded recursively is what produces this beautiful symmetric image. The coloring helps one see its natural progression.

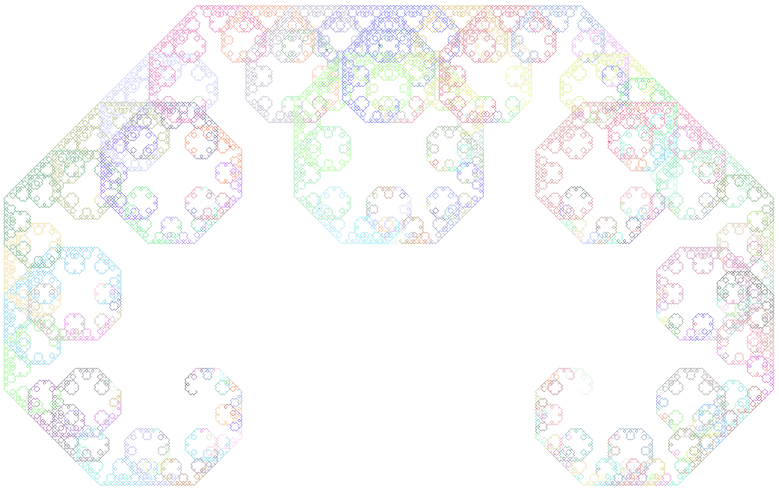


Figure 4: Lévy Dragon Fractal

3 Tiling

3.1 Square Tiling

To best introduce the concepts discussed in [4] I will give two examples that demonstrate fractile tiling of the *Euclidean plane* $\mathbb{R}^2$, which is defined as

Definition 9. *A countable family* $\{A_i\}$ *of compact sets that cover the plane with* $intA_i \cap intA_j = \emptyset$ *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.

We will begin with a plane divide into squares. Each square has nine inner squares labeled $1 - 9$ starting from the lower left corner and ending at the top right corner. Now imagine that the square tiles 1, 3, 7, and 9 have all been translated to their corresponding position in the square adjacent to their original square as shown in **Figure 5**.

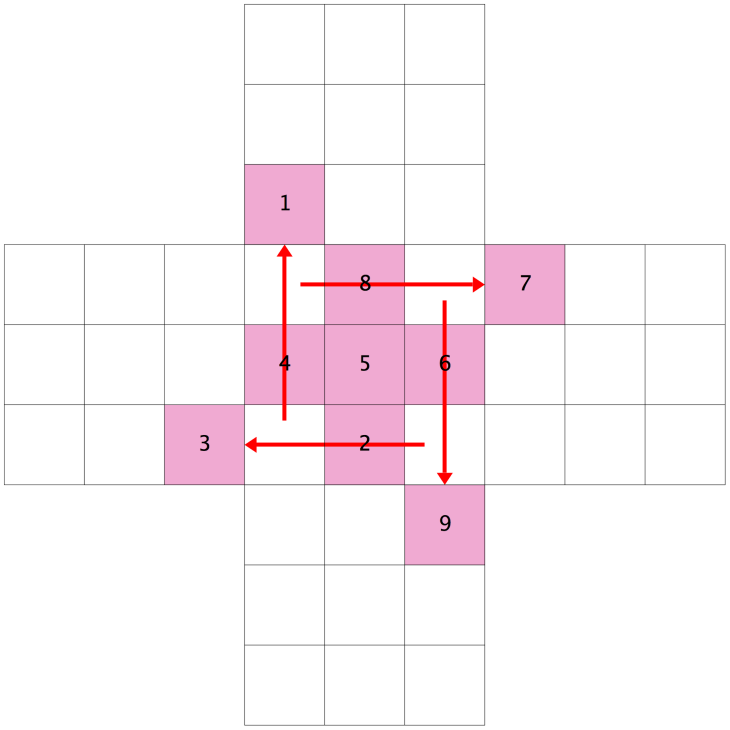
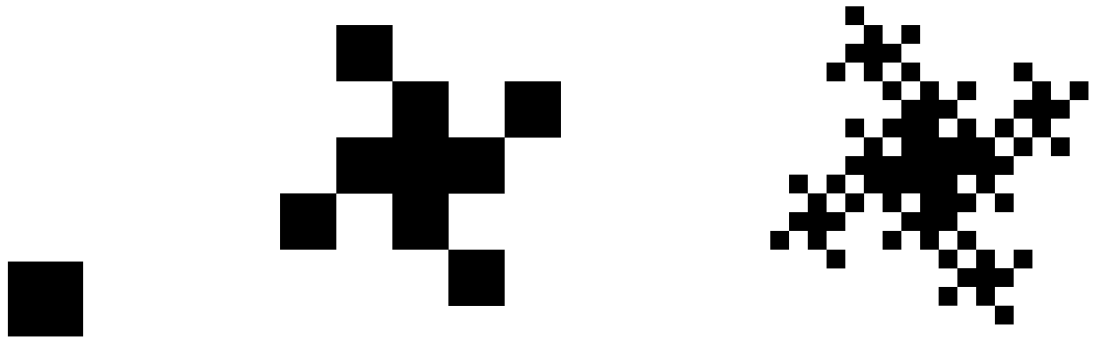


Figure 5: Pinwheel Fractal Pattern

Shown above is 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.

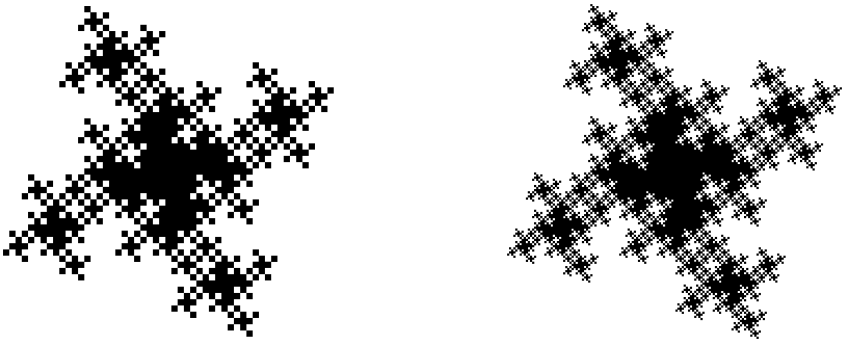
Definition 10. *An allowable pattern is a pattern that contains each tile, represented by a number, in the plane exactly once.*

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 6(c)**.



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

Figure 6: Pinwheel Fractal Stages



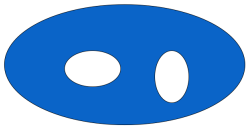
(a) Level 3 (b) Tile T

Figure 7: Pinwheel Fractal Stages

The process is repeated indefinitely, and with each iteration we notice that the area has a limit of the tile T, **Figure 7(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 11. *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.*

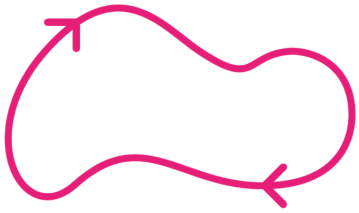
(a) Simply Connected



(b) Not Simply Connected

Example 4.

Definition 12. *A closed curve C is simple if it does not intersect itself.*

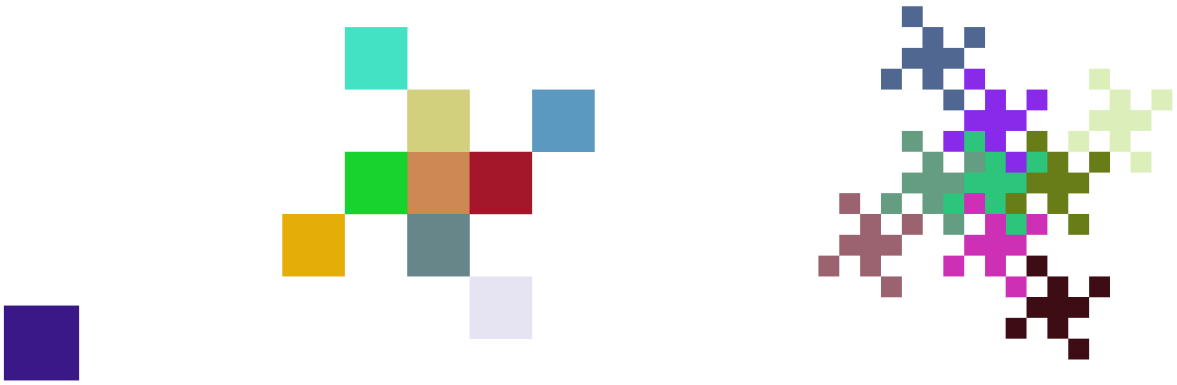


(c) Simple

(d) Not Simple

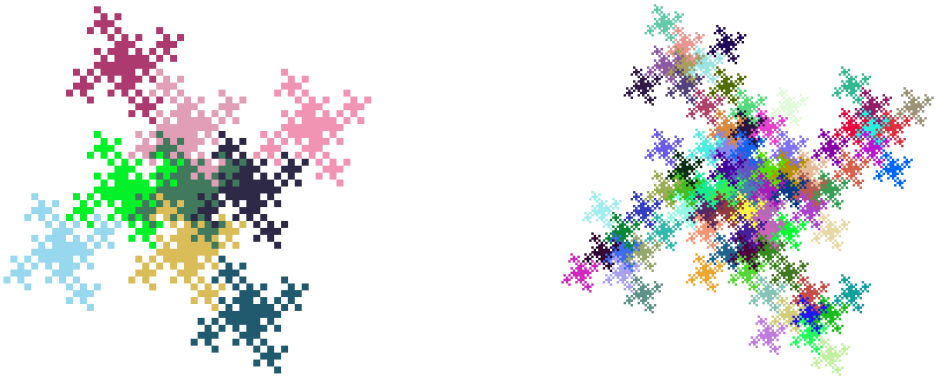
Example 5.

Coloring the fractal stages to define a single iteration helps show that no square coincides with another and the self-similarity of the fractal.



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

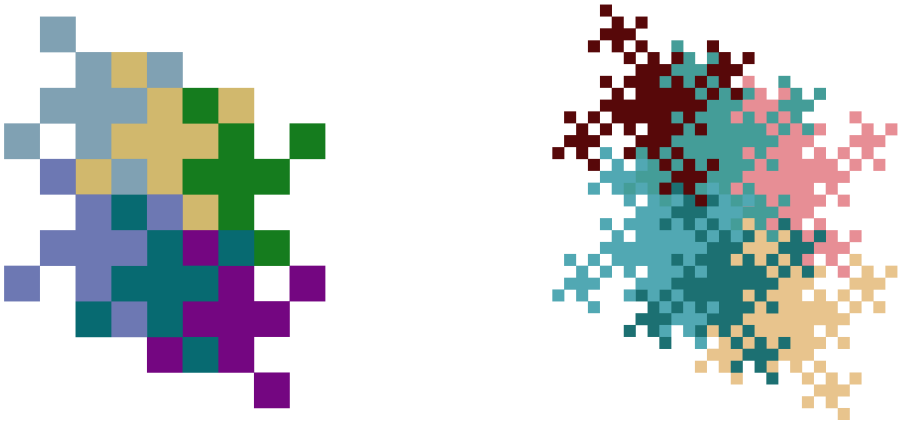
Figure 8: Pinwheel Fractal Stages



(a) Level 3 (b) Level 4

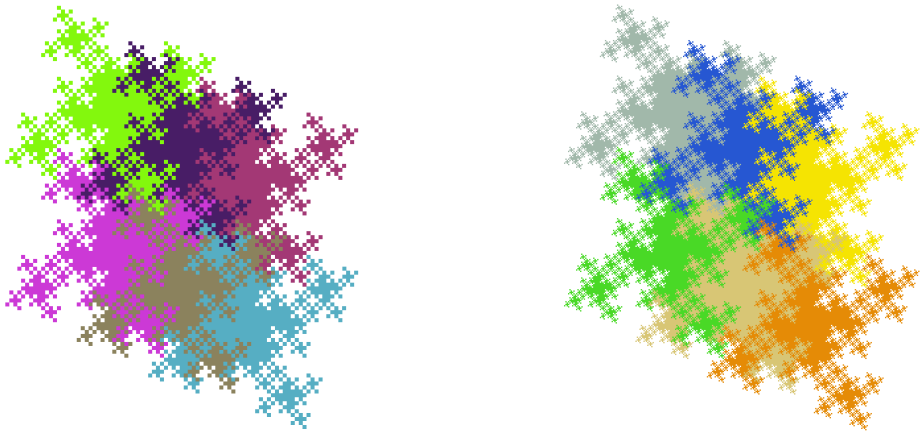
Figure 9: Pinwheel Fractal Stages

The objective in constructing these fractiles is to tile the Euclidean plane $\mathbb{R}^2$. After a finite number of iterations the fractile reaches its limiting area. We then fit four identical fractals onto its boundary as shown in **Figures 10(a), 10(b), 11(a), and 11(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 10: Pinwheel Fractal Tiling the Euclidean plane



(a) Level 4 (b) Level 5

Figure 11: Pinwheel Fractal Tiling the Euclidean Plane

3.2 Triangle Tiling

As with the sqare tiling, for triangle tiling we need an *allowable pattern*. In **Figure 12** the highlighted triangle is our base figure. In this triangle there are nine inner triangles. The triangles labeled 2, 4, and 7 are displaced outwardly onto their corresponding positions in the triangles adjacent to the main triangle. The triangles labeled 1, 5, and 9 are also relocated to their corresponding positions in the adjacent triganles.

The process is then repeated for each of those pink triangles. The second iteration produces the image in **Figure 13(c)** and so on.

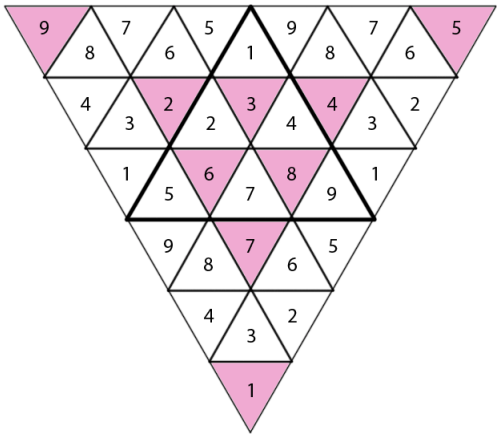
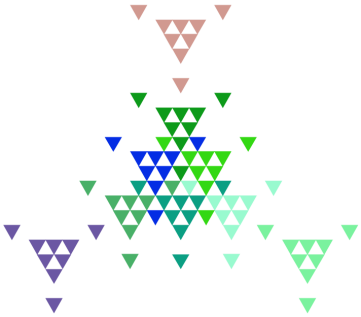


Figure 12: Triangle Pattern

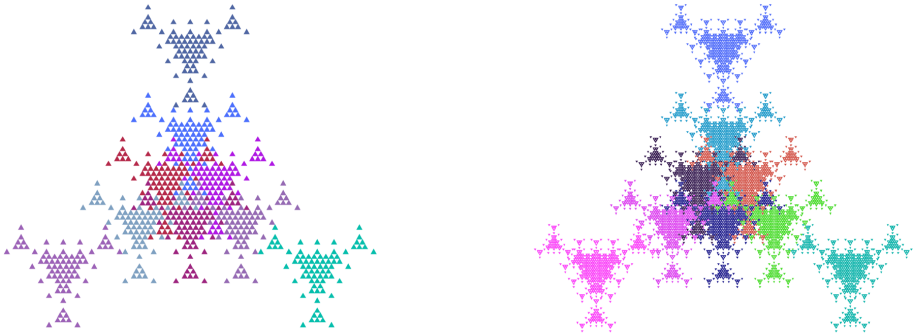
(a) Level 0

(b) Level 1



(c) Level 2

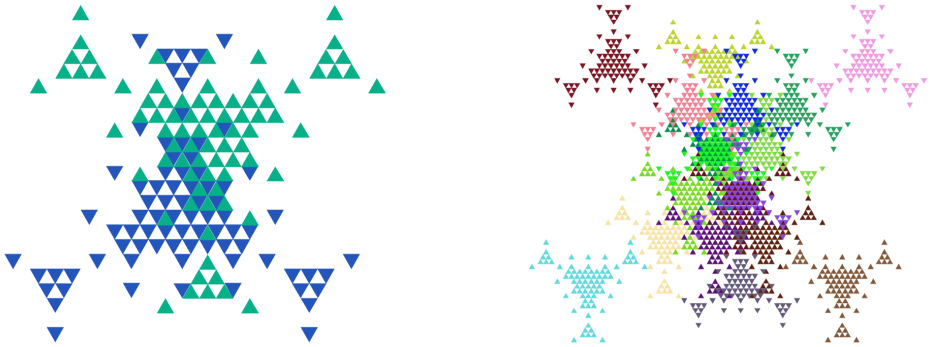
Figure 13: Triangle Fractal Stages



(a) Level 3 (b) Level 4

Figure 14: Triangle Fractal Stages

Similarily with the square tiling, we notice that by the fourth iteration **Figure 14(b)** there is 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 15(a), 15(b), and 16** notice that there are two triangle fractiles fitted together without any overlap.



(a) Level 2 (b) Level 2

Figure 15: Triangle Fractal Tiling of $\mathbb{R}^2$

Figure 16: 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 us consider a $(2x2)$ square matrix where the columns will serve as the scaling factors of our tile, and the inverse M^{-1} is a *contractive mapping.*

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

Definition 13. *A transformations $f : \mathbb{X} \to \mathbb{X}$ on a metric space $(\mathbb{X}, d)$ is called contractive or a contraction mapping if there is a constant $0 \leq s < 1$ such that*

$$d(f(x), f(y)) \leq s * d(x, y) \quad \forall x, y \in \mathbb{X}.$$

The number s is called a contractivity factor of f.

Then for $j = 1, \ldots n^2$ 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 the 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 **IFS** will converge to a compact set, the attractor. In other words the functions will converge to our desired tiling.

Definition 14. *An iterated function system consists of a complete metric space $(\mathbb{X}, d)$ together with a finite set of contrative mappings $w_n : \mathbb{X} \to \mathbb{X}$, with respect to contractivity factors s_n, for $n = 1, 2, ..., N$.*

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

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

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

Definition 15. *The fixed point $A \in \mathscr{H}(\mathbb{X})$ described in **Theorem 1** is called the attractor of the **IFS** (iterated function system).*

4 Julia Sets (Fractals and Complex Analysis)

First, let us begin with something we are more familiar with. 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 itterations 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 just 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 even have a strong presence in complex analysis, which has numerous applications in physics and engineering. This field of study certainly 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

```
1  /*
2   * Crista Moreno 05/21/09
3   * Produces Levy Dragon Fractal.
4   */
5  import java.awt.*;
6  import java.util.*;
7
8  public class LevyDragon {
9      public static void main(String[] args) {
10         DrawingPanel panel = new DrawingPanel(10000/3, 6000/3);
11         Graphics g = panel.getGraphics();
12         drawInstructions(buildInstructions(15), g);
13         panel.save("LevyDragon.png");
14     }
15
16     public static String buildInstructions(int numberOfItr) {
17         String axiom = "F";
18         for (int i = 0; i < numberOfItr; i++) {
19             axiom = "+" + axiom + "--" + axiom + "+";
20         }
21         return axiom;
22     }
23
24     public static void drawInstructions(String instructions, Graphics g) {
25         Random r = new Random();
26         Color custom = new Color(0, 0, 0);
27         double lLength = 25/3;
28         double currentAngle = 0;
29         double x = 2700.0/3;
30         double y = 4700.0/3;
31         for (int i = 0; i < instructions.length(); i++) {
32             char step = instructions.charAt(i);
33             switch (step) {
34                 case 'F':
35                     double x2 = x + (lLength * Math.cos(currentAngle));
36                     double y2 = y + (lLength * Math.sin(currentAngle));
37                     if (i%40 == 0)
38                         custom = new Color(r.nextInt(254), r.nextInt(254),
39                                 r.nextInt(254));
40                     g.setColor(custom);
```

```
41                      g.drawLine((int) Math.round(x), (int) Math.round(y),
42                              (int) Math.round(x2), (int) Math.round(y2));
43                      x = x2;
44                      y = y2;
45                      break;
46                  case '+':
47                      currentAngle += -Math.PI/4;
48                      break;
49                  case '-':
50                      currentAngle += Math.PI/4;
51                      break;
52              }
53          }
54      }
55  }
```

Square Tiling

```
1   /*
2    * Crista Moreno 05/21/09
3    * Produces Square Tiling Pinwheel Fractal.
4    */
5   import java.awt.*;
6   import java.util.*;
7
8   public class pinwheel {
9       public static void main(String[] args) {
10          DrawingPanel panel = new DrawingPanel(2000, 2000);
11          Graphics g = panel.getGraphics();
12          drawInstructions(g);
13          // panel.save("Pinwheel.png");
14      }
15
16      public static void drawInstructions(Graphics g) {
17          double x = 650;
18          double y = 650;
19          double length = 700.0;
20          Random r = new Random();
21          drawInstructions(g, x, y, length, 3, r);
22      }
23
24      public static void drawInstructions(Graphics g, double x, double y,
25              double length, int itr, Random r) {
26          if (itr == 3) {
27              g.setColor(new Color(r.nextInt(254), r.nextInt(254), r.nextInt(254)));
28          }
29          if (itr == 1) {
30              g.fillRect((int) Math.floor(x), (int) Math.floor(y),
31                      (int) Math.ceil(length), (int) Math.ceil(length));
32          } else {
33              length = length/3;
34              itr -= 1;
```

```
35              x = x + length;
36              y = y + length;
37              drawInstructions(g, x, y, length, itr, r);
38              drawInstructions(g, x, y - length, length, itr, r);
39              drawInstructions(g, x + length, y, length, itr, r);
40              drawInstructions(g, x, y + length, length, itr, r);
41              drawInstructions(g, x - length, y, length, itr, r);
42              drawInstructions(g, x + 2*length, y - length, length, itr, r);
43              drawInstructions(g, x - length, y - 2*length, length, itr, r);
44              drawInstructions(g, x - 2*length, y + length, length, itr, r);
45              drawInstructions(g, x + length, y + 2*length, length, itr, r);
46          }
47      }
48  }
```

Triangle Tiling

```
1   /*
2    * Crista Moreno 05/24/09
3    * Produces Example Triangle Tiling Fractal.
4    */
5   import java.awt.*;
6   import java.util.*;
7
8   public class triangleFractal {
9       public static final int SIZE = 500;
10      public static final int LEVEL = 3;
11
12      public static void main(String[] args) {
13          DrawingPanel panel = new DrawingPanel(1000, 750);
14          Graphics g = panel.getGraphics();
15          drawInstructions(g);
16          panel.save("Triangle_Fractal_level3.png");
17      }
18
19      public static void drawInstructions(Graphics g) {
20          double[] x = new double[] {100+250, SIZE-100+250, SIZE/2+250};
21          double[] y = new double[] {120+230, 120+230, SIZE-120+230};
22          g.setColor(Color.BLACK);
23          Random r = new Random();
24          drawInverted(g, x, y, LEVEL, r);
25      }
26
27      public static void drawInverted(Graphics g, double[] x, double[] y,
28              int itr, Random r) {
29          if (itr == 2) {
30              g.setColor(new Color(r.nextInt(254), r.nextInt(254),
31                  r.nextInt(254)));
32          }
33          if (itr == 0) {
34              g.fillPolygon(new int[] {(int) Math.floor(x[0]),
35                  (int) Math.ceil(x[1]),
```

```
36              (int) Math.round(x[2])}, new int[] {(int) Math.floor(y[0]),
37              (int) Math.floor(y[1]), (int) Math.ceil(y[2])}, 3);
38          } else {
39              itr -= 1;
40
41              double x2[];
42              double y2[];
43
44              double nBase = (x[1]-x[0])/3;
45              double nHeight = (nBase/2)*Math.sqrt(3);
46
47              // upRight triangles
48
49              // triangle 4
50              x2 = new double[] {x[0], x[0] + nBase/2, x[0] + nBase};
51              y2 = new double[] {y[0] + 2*nHeight, y[0] + nHeight,
52                  y[0] + 2*nHeight};
53              drawUpRight(g, x2, y2, itr, r);
54
55              // triangle 2
56              x2 = new double[] {x[1] - nBase, x[1] - nBase/2, x[1]};
57              y2 = new double[] {y[1] + 2*nHeight, y[1] + nHeight,
58                  y[1] + 2*nHeight};
59              drawUpRight(g, x2, y2, itr, r);
60
61              // triangle 1
62              x2 = new double[] {x[2] - nBase/2, x[2], x[2] + nBase/2};
63              y2 = new double[] {y[0] - 2*nHeight, y[0] - 3*nHeight,
64                  y[0] - 2*nHeight};
65              drawUpRight(g, x2, y2, itr, r);
66
67              // triangle 7
68              x2 = new double[] {x[2] - nBase/2, x[2], x[2] + nBase/2};
69              y2 = new double[] {y[0], y[0] - nHeight, y[0]};
70              drawUpRight(g, x2, y2, itr, r);
71
72              // triangle 8
73              x2 = new double[] {x[0] + nBase/2, x[0] + nBase, x[2]};
74              y2 = new double[] {y[0] + nHeight, y[0], y[0] + nHeight};
75              drawUpRight(g, x2, y2, itr, r);
76
77              // triangle 6
78              x2 = new double[] {x[2], x[2] + nBase/2, x[2] + nBase};
79              y2 = new double[] {y[0] + nHeight, y[0], y[0] + nHeight};
80              drawUpRight(g, x2, y2, itr, r);
81
82              // triangle 3
83              x2 = new double[] {x[2] - nBase/2, x[2], x[2] + nBase/2};
84              y2 = new double[] {y[2] - nHeight, y[2] - 2*nHeight,
85                  y[2] - nHeight};
86              drawUpRight(g, x2, y2, itr, r);
```

```
87
88              // triangle 9
89              x2 = new double[] {x[2] + 2*nBase, x[2] + 2*nBase +
90                  nBase/2, x[2] + 3*nBase};
91              y2 = new double[] {y[2], y[2] - nHeight, y[2]};
92              drawUpRight(g, x2, y2, itr, r);
93
94              // triangle 5
95              x2 = new double[] {x[2] - 3*nBase, x[2] - 3*nBase +
96                  nBase/2, x[2] - 2*nBase};
97              y2 = new double[] {y[2], y[2] - nHeight, y[2]};
98              drawUpRight(g, x2, y2, itr, r);
99          }
100     }
101
102     public static void drawUpRight(Graphics g, double[] x, double[] y,
103             int itr, Random r) {
104         if (itr == 2) {
105             g.setColor(new Color(r.nextInt(254), r.nextInt(254),
106                 r.nextInt(254)));
107         }
108         if (itr == 0) {
109             g.fillPolygon(new int[] {(int) Math.floor(x[0]),
110                 (int) Math.round(x[1]), (int) Math.ceil(x[2])},
111                 new int[] {(int) Math.ceil(y[0]), (int) Math.floor(y[1]),
112                 (int) Math.ceil(y[2])}, 3);
113         } else {
114             itr -= 1;
115
116             double x2[];
117             double y2[];
118
119             // inverted triangles
120
121             double nBase = (x[2]-x[0])/3;
122             double nHeight = (nBase/2)*Math.sqrt(3);
123
124             // triangle 5
125             x2 = new double[] {x[1] + 2*nBase, x[1] + 3*nBase,
126                 x[1] + 3*nBase - nBase/2};
127             y2 = new double[] {y[1], y[1], y[1] + nHeight};
128             drawInverted(g, x2, y2, itr, r);
129
130             // triangle 9
131             x2 = new double[] {x[1] - 3*nBase, x[1] - 2*nBase,
132                 x[1] - 2*nBase - nBase/2};
133             y2 = new double[] {y[1], y[1], y[1] + nHeight};
134             drawInverted(g, x2, y2, itr, r);
135
136             // triangle 3
137             x2 = new double[] {x[1] - nBase/2, x[1] + nBase/2, x[1]};
```

```
138             y2 = new double[] {y[1] + nHeight, y[1] + nHeight,
139                 y[1] + 2*nHeight};
140             drawInverted(g, x2, y2, itr, r);
141
142             // triangle 6
143             x2 = new double[] {x[0] + nBase/2, x[1], x[0] + nBase};
144             y2 = new double[] {y[0] - nHeight, y[0] - nHeight, y[0]};
145             drawInverted(g, x2, y2, itr, r);
146
147             // triangle 8
148             x2 = new double[] {x[1], x[1] + nBase, x[1] + nBase/2};
149             y2 = new double[] {y[0] - nHeight, y[0] - nHeight, y[0]};
150             drawInverted(g, x2, y2, itr, r);
151
152             // triangle 7
153             x2 = new double[] {x[1] - nBase/2, x[1] + nBase/2, x[1]};
154             y2 = new double[] {y[0], y[0], y[0] + nHeight};
155             drawInverted(g, x2, y2, itr, r);
156
157             // triangle 2
158             x2 = new double[] {x[0], x[0] + nBase, x[0] + nBase/2};
159             y2 = new double[] {y[1] + nHeight, y[1] + nHeight,
160                 y[1] + 2*nHeight};
161             drawInverted(g, x2, y2, itr, r);
162
163             // triangle 4
164             x2 = new double[] {x[1] + nBase/2, x[2], x[2] - nBase/2};
165             y2 = new double[] {y[1] + nHeight, y[1] + nHeight,
166                 y[1] + 2*nHeight};
167             drawInverted(g, x2, y2, itr, r);
168
169             // triangle 1
170             x2 = new double[] {x[1] - nBase/2, x[1] + nBase/2,  x[1]};
171             y2 = new double[] {y[0] + 2*nHeight, y[0] + 2*nHeight,
172                 y[0] + 3*nHeight};
173             drawInverted(g, x2, y2, itr, r);
174         }
175     }
176 }
```

DrawingPanel [6]

```
1  /*
2  Stuart Reges and Marty Stepp
3
4  07/01/2005
5
6  The DrawingPanel class provides a simple interface for drawing persistent
7  images using a Graphics object.  An internal BufferedImage object is used
8  to keep track of what has been drawn.  A client of the class simply
9  constructs a DrawingPanel of a particular size and then draws on it with
10 the Graphics object, setting the background color if they so choose.
```

```
11
12 To ensure that the image is always displayed, a timer calls repaint at
13 regular intervals.
14 */
15
16 import java.awt.*;
17 import java.awt.event.*;
18 import java.awt.image.*;
19 import javax.imageio.*;
20 import javax.swing.*;
21 import javax.swing.event.*;
22
23 public class DrawingPanel implements ActionListener {
24     public static final int DELAY = 250;  // delay between repaints in millis
25     private static final String DUMP_IMAGE_PROPERTY_NAME = "drawingpanel.save";
26     private static String TARGET_IMAGE_FILE_NAME = null;
27     private static final boolean PRETTY = true;  // true to anti-alias
28     private static boolean DUMP_IMAGE = false;  // true to write DrawingPanel
29                                                 // to file
30     private int width, height;     // dimensions of window frame
31     private JFrame frame;          // overall window frame
32     private JPanel panel;          // overall drawing surface
33     private BufferedImage image;   // remembers drawing commands
34     private Graphics2D g2;         // graphics context for painting
35     private JLabel statusBar;      // status bar showing mouse position
36     private long createTime;
37
38     static {
39         TARGET_IMAGE_FILE_NAME = System.getProperty(DUMP_IMAGE_PROPERTY_NAME);
40         DUMP_IMAGE = (TARGET_IMAGE_FILE_NAME != null);
41     }
42
43     // construct a drawing panel of given width and height enclosed in a window
44     public DrawingPanel(int width, int height) {
45         this.width = width;
46         this.height = height;
47         this.image = new BufferedImage(width, height,
48             BufferedImage.TYPE_INT_ARGB);
49         this.statusBar = new JLabel(" ");
50         this.statusBar.setBorder(BorderFactory.createLineBorder(Color.BLACK));
51         this.panel = new JPanel(new FlowLayout(FlowLayout.CENTER, 0, 0));
52         this.panel.setBackground(Color.WHITE);
53         this.panel.setPreferredSize(new Dimension(width, height));
54         this.panel.add(new JLabel(new ImageIcon(image)));
55
56         // listen to mouse movement
57         MouseInputAdapter listener = new MouseInputAdapter() {
58             public void mouseMoved(MouseEvent e) {
59                 DrawingPanel.this.statusBar.setText("(" + e.getX() + ", "
60                     + e.getY() + ")");
61             }
```

```
62
63              public void mouseExited(MouseEvent e) {
64                  DrawingPanel.this.statusBar.setText(" ");
65              }
66          };
67
68          this.panel.addMouseListener(listener);
69          this.panel.addMouseMotionListener(listener);
70          this.g2 = (Graphics2D)image.getGraphics();
71          this.g2.setColor(Color.BLACK);
72
73          if (PRETTY) {
74              this.g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
75                  RenderingHints.VALUE_ANTIALIAS_ON);
76              this.g2.setStroke(new BasicStroke(1.1f));
77          }
78
79          this.frame = new JFrame("Drawing Panel");
80          this.frame.setResizable(false);
81          this.frame.addWindowListener(new WindowAdapter() {
82              public void windowClosing(WindowEvent e) {
83                  if (DUMP_IMAGE) {
84                      DrawingPanel.this.save(TARGET_IMAGE_FILE_NAME);
85                  }
86                  System.exit(0);
87              }
88          });
89
90          this.frame.getContentPane().add(panel);
91          this.frame.getContentPane().add(statusBar, "South");
92          this.frame.pack();
93          this.frame.setVisible(true);
94          if (DUMP_IMAGE) {
95              createTime = System.currentTimeMillis();
96              this.frame.toBack();
97          } else {
98              this.toFront();
99          }
100
101         // repaint timer so that the screen will update
102         new Timer(DELAY, this).start();
103     }
104
105     // used for an internal timer that keeps repainting
106     public void actionPerformed(ActionEvent e) {
107         this.panel.repaint();
108         if (DUMP_IMAGE && System.currentTimeMillis() >
109                 createTime + 4 * DELAY) {
110             this.frame.setVisible(false);
111             this.frame.dispose();
112             this.save(TARGET_IMAGE_FILE_NAME);
```

```
113             System.exit(0);
114         }
115     }
116
117     // obtain the Graphics object to draw on the panel
118     public Graphics2D getGraphics() {
119         return this.g2;
120     }
121
122     // set the background color of the drawing panel
123     public void setBackground(Color c) {
124         this.panel.setBackground(c);
125     }
126
127     // show or hide the drawing panel on the screen
128     public void setVisible(boolean visible) {
129         this.frame.setVisible(visible);
130     }
131
132     // makes the program pause for the given amount of time,
133     // allowing for animation
134     public void sleep(int millis) {
135         try {
136             Thread.sleep(millis);
137         } catch (InterruptedException e) {}
138     }
139
140     // take the current contents of the panel and write them to a file
141     public void save(String filename) {
142         String extension = filename.substring(filename.lastIndexOf(".") + 1);
143
144         // create second image so we get the background color
145         BufferedImage image2 = new BufferedImage(this.width, this.height,
146             BufferedImage.TYPE_INT_RGB);
147         Graphics g = image2.getGraphics();
148         g.setColor(panel.getBackground());
149         g.fillRect(0, 0, this.width, this.height);
150         g.drawImage(this.image, 0, 0, panel);
151
152         // write file
153         try {
154             ImageIO.write(image2, extension, new java.io.File(filename));
155         } catch (java.io.IOException e) {
156             System.err.println("Unable to save image:\n" + e);
157         }
158     }
159
160     // makes drawing panel become the frontmost window on the screen
161     public void toFront() {
162         this.frame.toFront();
163     }
```

```
164 }
```