CHAPTER-7
CHAPTER-7

FRACTAL TREE GENERATION USING RECURSIVE METHOD

Mandelbrot [16] gives examples of the recursive branching structures of trees and flowers, analyzes their Hausdorff-Besicovitch dimension and writes inconclusively trees may be called fractals in part. Smith [10] recognizes similarities between algorithms yielding Koch curves and branching plant-like structures, but does not qualify plant models as fractals. These structures are produced in a finite number of steps and consist of a finite number of line segments, while the notion of fractal is defined only in the limit. Oppenheimer [71] uses the term fractal more freely, exchanging it with self-similarity, and comments: The geometric notion of self-similarity became a paradigm for structure in the natural world. Nowhere is this principle more evident than in the world of botany. The approach presented in this chapter, which considers fractals as simplified abstract representations of real plant structures, seems to reconcile these previous opinions.

7.1 Symmetry and Self-Similarity

The notion of symmetry is generally defined as the invariance of a configuration of elements under a group of automorphic transformations. Commonly considered transformations are congruence’s, which can be obtained by composing rotations, reflections and translations. Could we extend this list of transformations to similarities, and consider self-similarity as a special case of symmetry involving scaling operations? On the surface, this seems possible. For example, Weyl suggests: “In dealing with potentially infinite patterns like band ornaments or with infinite groups, the operation
under which a pattern is invariant is not of necessity congruence but could be a similarity.” The spiral shapes of the shells Turritella duplicata and Nautilus are given as examples. However, all similarities involved have the same fixed point. The situation changes dramatically when similarities with different fixed points are considered. For example [96], the Sierpinski gasket is mapped onto itself by a set of three contractions $T_1$, $T_2$ and $T_3$. Each contraction takes the entire figure into one of its three main components. Thus, if $A$ is an arbitrary point of the gasket, and $T = T_1T_2 \ldots T_n$ is an arbitrary composition of transformations $T_1$, $T_2$ and $T_3$, the image $T(A)$ will belong to the set $A$. On the other hand, if the inverses of transformations $T_1$, $T_2$ and $T_3$ can also be included in the composition, one obtains points that do not belong to the set $A$ nor its infinite extension. This indicates that the set of transformations that maps $A$ into itself forms a semi group generated by $T_1$, $T_2$ and $T_3$, but does not form a group. Thus, self-similarity is a weaker property than symmetry, yet it still provides a valuable insight into the relationships between the elements of a structure.

Figure 7.1.1 fern
7.2 Plant Models and Iterated Function Systems

Barnsley [55] presents a model of a fern leaf (Figure 7.1.1), generated using an iterated function system, or IFS. This raises a question regarding the relationship between developmental plant models expressed using L-systems and plant-like structures captured by IFSes. This section briefly describes IFSes and introduces a method for constructing those which approximate structures generated by a certain type of parametric L-system. The restrictions of this method are analyzed, shedding light on the role of IFSes in the modeling of biological structures. By definition [78], a planar iterated function system is a finite set of contractive affine mappings \( T = \{T_1, T_2, \ldots, T_n\} \) which map the plane \( \mathbb{R} \times \mathbb{R} \) into itself. The set defined by \( T \) is the smallest nonempty set \( A \), closed in the topological sense, such that the image \( y \) of any point \( x \in A \) under any of the mappings \( T_i \in T \) also belongs to \( A \). It can be shown that such a set always exists and is unique [78] (see also [118] for an elementary presentation of the proof). Thus, starting from an arbitrary point \( x \in A \), one can approximate \( A \) as a set of images of \( x \) under compositions of the transformations from \( T \).

7.3 Modeling and Generating Fractal Tree

Fractal recursion algorithm is a render method of fractal graphics[9]. At first, setting basic graphics generate block, then render it repeatedly on each level depend on render formulae, until switch condition is false. In our fractal tree model, graphic generate block is planar furcated tree. In figure 7.3.1, segment DA denotes trunk, segment AB
denotes branch, \( L \) denotes length of \( AB \), angle \( \theta \) denotes transfer angle both branch and horizontal axis. \( \beta \) is an angle between branches.

The following is the process of modeling:

i. Render trunk \( DA \);

ii. Calculate coordinates of \( B \), render \( AB \);

iii. Calculate coordinates of \( C \), render \( AC \);

iv. Move \( A \) to \( B \), and \( B,C \) as new branch of \( A \);

v. Running (i) and (ii) repeatedly, until switch conditions is false.

![Figure 7.3.1 Fractal tree model](image)

### 7.4 Transform of Tree Parameter

Fractal tree rendered need many parameters which would be transformed, for it we used a method that are controlled on a certain scope, so that acquire changed shape and multicolored fractal tree. The following are definitions of parameters of tree.

The start coordinates of each branch is \( (x_0, y_0) \), the end is \( (x_1, y_1) \); the furcated numbers is \( B \) every tree node; angle \( \theta \) is both trunk and branch; increment angle \( \beta \) is between
branches; W is the thickness of branches; L is the length of branches; level is node numbers from root to leaf each branch, which is the most important control part; sc is pixel size of branch rendered. Transform randomly including coordinates, branch length, branch angle, trunk thickness, render color.

Transform formula of Branch length is

\[ L = \text{Int}(\text{Rnd} \times (\text{level} \times sc \times 2)) \] …1

Rnd is a function to get random value from 0 to 1. Int denotes take an i is an integer from 0 to B.

Coordinates transform formula is

\[
\begin{align*}
x_1 &= x_0 + L \times \cos(\text{Rnd} \times \theta) + \beta \\
y_1 &= y_0 - L \times \sin(\text{Rnd} \times \theta) + \beta
\end{align*}
\] ...2

The branch thickness is equal to nodes numbers

\[ W = \text{level} \] …3

W value denotes line scale when render on program. Owing to level value would reduce gradually in running; the branch would also become thin, which tally with living characteristic of natural scenery. In color transform gradually, we use RGB mode, the principium is that the color would be more saturated a near root, would be more thin a near leaf. We should make full use of the value of level to control the shadow of color.
7.5 Generating Algorithm

Generating algorithm is a recursion call repeatedly. In generating, first initial parameters then render a trunk; last render a branch start from one end of trunk. Once function call would render a trunk and a branch, function call time after time would render entire fractal tree. In our function designed, there are five formal parameters, B would control furcated numbers of tree; level would control node numbers of branch, it would be decrease 1 after render a branch; sc is a pixel size; (x1,y1) is a start coordinates, after rendered a branch, it would be replace with coordinates of leaf end, and regard as a new coordinates to render a new branch. Our algorithm guarantees the worst-case cost of $O(B \times \text{level})$.

Program to generate tree using recursive algorithm

```java
import java.awt.*;
import java.awt.geom.Line2D;
import javax.swing.JFrame;
import javax.swing.JPanel;

public class Fractal extends JPanel {
    public Graphics2D g1;
    public static final int maxAngle = 360; //Maximum number of degrees in a period = 360
    public static final int startX = 600; //Origin position X
    public static final int startY = 800; //Origin position Y
    public static final int numOfRecursions = 9; //Number of recursive levels
```
public static final int startAngle = 0; //Origin angle in degrees (0 = vertical)
public static final double treeSize = 2; //Size of the tree. (Default = 2)
public static final int Detail = 1; //Detail level (Max = 1)
public static final int randFact = 30; //Random factor on branch angles
public static final int[] constFact = {-60, 05, -50, 45}; //Constant factor on branch angles
public static int[] red =   {0, 0, 0, 0, 7, 15, 23, 31, 39, 47, 55, 43}; //Array representation
public static int[] green = {171, 159, 147, 135, 123, 111, 99, 87, 75, 63, 51, 43}; //of a color scheme
public static int[] blue = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
public static double degToRad(int deg) { //degree to radian conversion
    return deg * Math.PI / 180;
}

//Recursive method to create fractal pattern, using n levels of recursion
public static void drawFractal(Graphics2D g1, int x, int y, int n, int angle) {

    if (n == Detail) return;
    int len = (int) Math.round(Math.pow(treeSize, n - 1)); //Calculates the length of branch
int xn1 = (int) Math.round(x - (2 * len * Math.sin(degToRad(angle))));

//Length and angle
int yn1 = (int) Math.round(y - (2 * len * Math.cos(degToRad(angle))));

//of the branches
int mid1x = (x + xn1) / 2; //Positions of the branches
int mid1y = (y + yn1) / 2;
int mid2x = (mid1x + xn1) / 2;
int mid2y = (mid1y + yn1) / 2;
int mid3x = (x + mid1x) / 2;
int mid3y = (y + mid1y) / 2;
int mid4x = (mid3x + mid1x) / 2;
int mid4y = (mid3y + mid1y) / 2;

java.util.Random r = new java.util.Random(); //Creates a random number generator
drawFractal(g1, mid1x, mid1y, n - 1, (angle + r.nextInt(randFact) +
 constFact[0]) % maxAngle);
drawFractal(g1, mid2x, mid2y, n - 1, (angle + r.nextInt(randFact) +
 constFact[1]) % maxAngle);
drawFractal(g1, mid3x, mid3y, n - 1, (angle + r.nextInt(randFact) +
 constFact[2]) % maxAngle);
drawFractal(g1, mid4x, mid4y, n - 1, (angle + r.nextInt(randFact) +
 constFact[3]) % maxAngle);
Color c = new Color(red[(r.nextInt() % 3) + n], green[(r.nextInt() % 3) + n], blue[(r.nextInt() % 3) + n]); // Creates a color using rgb-values

g1.setColor(c); // Sets the color
Line2D L1 = new Line2D.Double(x, y, xn1, yn1); // Creates a line

g1.draw(L1); // Draws the line
return;

}

public void paint(final Graphics g) {

g1 = (Graphics2D) g;
drawFractal(g1, startX, startY, numOfRecursions, startAngle);

}

public static void main(String args[]) {
JFrame FF = new JFrame("Drawing a recursive tree");
FF.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
Fractal F = new Fractal();
FF.setBackground(Color.BLACK);
FF.add(F);
FF.pack();
FF.setVisible(true);
FF.setSize(1200, 1000);
}
The method fractal immediately returns if the level of the recursion is equal to the one (i.e. n ==1). Otherwise it computes the length of the branch and then compute coordinates of the end point using the trigonometric class functions Math.cos and Math.sin (please note that in the Java coordinate system a larger value for the second coordinate means lower points in the plane). After that set the position of the branches and the angle are rotated by different degrees in both directions. The recursive calls then draw the self-replicated tree structures. The set color method is used to set the color of the tree.

By changing the recursive level, tree size and branch angle we generate the different types of tree structure that shown in fig7.5.1, fig7.5.2, fig7.5.3, fig7.5.4 and fig7.5.5 respectively.

Figure -7.5.1 Recursive Fractal tree (recursive level=8, tree size=2, branch angle=45)
Figure 7.5.2 Recursive Fractal tree (recursive level= 6, tree size=3, branch angle=60)

Figure 7.5.3 Recursive Fractal tree (recursive level= 8, tree size=2, branch angle=90)
Figure 7.5.4 Recursive Fractal tree (recursive level= 5, tree size=3, branch angle=30)

Figure 7.5.5 Recursive Fractal tree (recursive level= 8, tree size=2, branch angle=30)
7.6 Conclusion

Since transformation is an important method in Computer Graphics and they are helpful in generating and analyzing graphical objects from different perspective. In a recursive method we had given starting information and a rule for how to use it to get new information. Then we repeat the rule using the new information as it was the starting information. Chapter concluded that helps of basic transformation we can generate different fractal objects.