Generating Fractals from Voronoi Diagrams

Ken Shirriff
Computer Science Division
571 Evans Hall
University of California, Berkeley
Berkeley, CA 94720

ABSTRACT

This paper describes how to generate fractal patterns by recursively creating Voronoi diagrams on a set of points. These patterns resemble such things as leaf veins and roadmaps. By varying the degree of subdivision and the distribution of points, different output patterns can be obtained.

August 23, 1995

Generating Fractals from Voronoi Diagrams

Ken Shirriff
Computer Science Division
571 Evans Hall
University of California, Berkeley
Berkeley, CA 94720

Voronoi diagrams can be used to generate interesting fractal patterns which resemble leaf veins, roadmaps, and cracked pottery glaze. The patterns are generated by recursively creating a Voronoi diagram inside each Voronoi polygon. (A Voronoi diagram consists of nearest-neighbor polygons. Given a set of points, the Voronoi polygon around each point is the region of the plane closer to the selected point than to any other point [PrS85].)

To generate the fractal images, we start with a small set of points and draw the Voronoi diagram of these points. Then, we use a denser set of points and compute a new Voronoi diagram inside each region of the first diagram. That is, for each region, we consider only the points inside that region, and draw a Voronoi diagram clipped to that region. We repeat this process recursively, building a new Voronoi diagram inside each region of the higher level diagram. Listing 1 gives a pseudocode description of the algorithm. Figure 1 demonstrates execution of the algorithm through four levels. The first point set consists of 5 points; the next sets are 25, 125, and 625 points.

The output can be varied in several ways. The most obvious parameters are the number of points at each level and the total recursion depth. Figure 2 uses 10, 100, 1000, and 10000 points in four levels of recursion. Thus, each polygon is subdivided into an average of 10 subpolygons. The line thickness is halved at each level to emphasize the higher levels. In Figure 3, the number of points at each level is computed with factorials: the levels have 2!, 3!, 4!, 5!, 6!, 7!, and 8! points. This results in more subdivision in deeper levels. Finally, a small degree of subdivision at each level, for instance by using

3, 9, 27, and 81 points, gives a more angular result than larger subdivisions.

The distribution of points can be changed to affect the output. Figure 4 uses a distribution of points in a circle, weighted towards the edge. The result is somewhat reminiscent of Escher. Figure 5 uses a nonuniform distribution in the plane; the first level is weighted to the left, the second to the bottom, the third to the right, and the fourth to the top. As a result, polygons on different scales shrink in each of the four directions.

A final way to vary the output is, for each polygon, randomly decide if it is to be subdivided or not subdivided. This creates an image with voids of various sizes, and with a smaller fractal dimension. However, the result is not particularly pleasing, and a sample image is not included.

Acknowledgements

This research was supported by an IBM Graduate Fellowship. The initial Voronoi diagram code was written by Steven Fortune and is described in [For87]. I wish to thank Clifford Pickover for his helpful comments.

References

[For87] S. Fortune, A Sweepline Algorithm for Voronoi Diagrams, *Algorithmica 2* (1987), 153-174.

[PrS85] F. P. Preparata and M. I. Shamos, *Computational Geometry*, Springer-Verlag, New York, 1985.

Figure captions:

Figure 1: Iterations of the algorithm over four levels, showing results after 5, 25, 125, and 625 points.

Figure 2: A Voronoi fractal generated by subdivision by 10. The four levels have 10, 100, 1000, and 10000 points.

Figure 3: A Voronoi fractal with more subdivision at lower levels. The levels have 2!, 3!, 4!, 5!, 6!, 7!, and 8! points.

Figure 4: Nonuniform distribution on a disk. The levels have 10, 100, 1000, and 10000 points.

Figure 5: Nonuniform distribution on a square. Successive levels are weighted towards the left, bottom, right, and top, and have 10, 100, 1000, and 10000 points respectively.

Listing 1: Pseudocode for the Voronoi fractal algorithm.

"max_level" is the number of levels to do.
"num_points" is an array of the number of points at each level.
"pointset" is an array of sets of points.
"drawing_area" is the initial screen region.
"vor_polygons" is a set of Voronoi polygons.
"generate_points" generates a set of points.
"intersect" intersects a set of points with a region, returning a set of points.
"voronoi_diagram" generates the Voronoi polygons of a set of points.
"draw" draws polygons clipped to a region.

```
function main(max_level, num_points)
    pointset[max_level] = generate_points(num_points[max_level])
    for i = 1 to max_level-1
        pointset[i] = first num_points[i] points
                of pointset[max_level]
    end for
    draw_fractal(1, drawing_area)

function draw_fractal(level, region)
    points = intersect(pointset[level], region)
    vor_polygons = voronoi_diagram(points)
    draw(vor_polygons, clipped to region)
    if (level < max_level)
        for i in points
            draw_fractal(level+1, vor_polygons[i])
        end for
    end if
```

Listing 1.