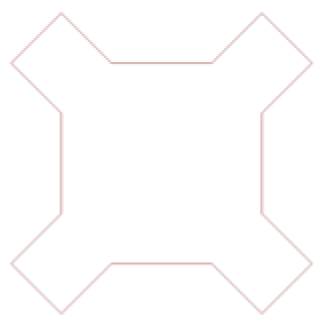
# Sierpiński curve

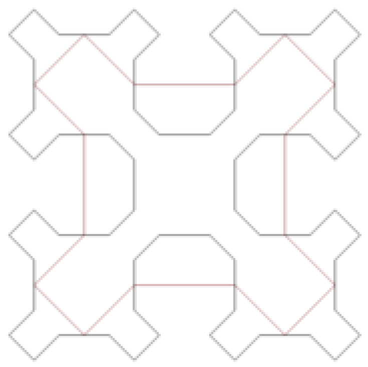
**Sierpiński curves** are a recursively defined sequence of continuous closed plane fractal curves discovered by Wacław Sierpiński, which in the limit $n \to \infty$ completely fill the unit square: thus their limit curve, also called **the Sierpiński curve**, is an example of a space-filling curve.

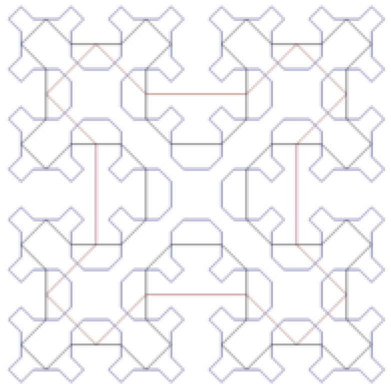
Because the Sierpiński curve is space-filling, its Hausdorff dimension (in the limit $n \to \infty$) is $2$.
The Euclidean length of

$$S_n \text{ is } l_n = \frac{2}{3}(1+\sqrt{2})2^n - \frac{1}{3}(2-\sqrt{2})\frac{1}{2^n},$$

i.e., it grows *exponentially* with $n$ beyond any limit, whereas the limit for $n \to \infty$ of the area enclosed by $S_n$ is $5/12$ that of the square (in Euclidean metric).


Sierpiński curve of first order


Sierpiński curves of orders 1 and 2


Sierpiński curves of orders 1 to 3

## Uses of the curve

The Sierpiński curve is useful in several practical applications because it is more symmetrical than other commonly studied space-filling curves. For example, it has been used as a basis for the rapid construction of an approximate solution to the Travelling Salesman Problem (which asks for the shortest sequence of a given set of points): The heuristic is simply to visit the points in the same sequence as they appear on the Sierpiński curve. To do this requires two steps: First compute an inverse image of each point to be visited; then sort the values. This idea has been used to build routing systems for commercial vehicles based only on Rolodex card files.

A space-filling curve is a continuous map of the unit interval onto a unit square and so a (pseudo) inverse maps the unit square to the unit interval. One way of constructing a pseudo-inverse is as follows. Let the lower-left corner (0, 0) of the unit square correspond to 0.0 (and 1.0). Then the upper-left corner (0, 1) must correspond to 0.25, the upper-right corner (1, 1) to 0.50, and the lower-right corner (1, 0) to 0.75. The inverse map of interior points are computed by taking advantage of the recursive structure of the curve. Here is a function coded in Java that will compute the relative position of any point on the Sierpiński curve (that is, a pseudo-inverse value). It takes as input the coordinates of the point (x,y) to be inverted, and the corners of an enclosing right isosceles triangle (ax, ay), (bx, by), and (cx, cy). (Note that the unit square is the union of two such triangles.) The remaining parameters specify the level of accuracy to which the inverse should be computed.

```
    static long sierp_pt2code( double ax, double ay, double bx, double by, double cx, double cy,
        int currentLevel, int maxLevel, long code, double x, double y )
    {
        if (currentLevel <= maxLevel) {
            currentLevel++;
            if ((sqr(x-ax) + sqr(y-ay)) < (sqr(x-cx) + sqr(y-cy))) {
                code = sierp_pt2code( ax, ay, (ax+cx)/2.0, (ay+cy)/2.0, bx, by,
                    currentLevel, maxLevel, 2 * code + 0, x, y );
            }
            else {
                code = sierp_pt2code( bx, by, (ax+cx)/2.0, (ay+cy)/2.0, cx, cy,
                    currentLevel, maxLevel, 2 * code + 1, x, y );
            }
        }
        return code;
    }
```

## Drawing the curve

The following Java applet draws a Sierpiński curve by means of four mutually recursive methods (methods that call one another):

```
import java.applet.Applet;
import java.awt.Graphics;
import java.awt.Image;

public class SierpinskyCurve extends Applet {

    private SimpleGraphics sg = null;
    private int dist0 = 128, dist;
    private Image offscrBuf;
    private Graphics offscrGfx;

    public void init() {
        sg = new SimpleGraphics(getGraphics());
        dist0 = 100;
        resize(4 * dist0, 4 * dist0);
    }
    public void update(Graphics g) {
        paint(g);
    }

    public void paint(Graphics g) {

        if (g == null)
            throw new NullPointerException();

        if (offscrBuf == null) {
            offscrBuf = createImage(this.getWidth(), this.getHeight());
            offscrGfx = offscrBuf.getGraphics();
            sg.setGraphics(offscrGfx);
        }

        int level = 3;
        dist = dist0;
        for (int i = level; i > 0; i--)
            dist /= 2;
        sg.goToXY(2 * dist, dist);
        sierpA(level); // start recursion
        sg.lineRel('X', +dist, +dist);
        sierpB(level); // start recursion
        sg.lineRel('X', -dist, +dist);
        sierpC(level); // start recursion
        sg.lineRel('X', -dist, -dist);
        sierpD(level); // start recursion
        sg.lineRel('X', +dist, -dist);

        g.drawImage(offscrBuf, 0, 0, this);

    }

    private void sierpA(int level) {
        if (level > 0) {
            sierpA(level - 1);
            sg.lineRel('A', +dist, +dist);
            sierpB(level - 1);
            sg.lineRel('A', +2 * dist, 0);
            sierpD(level - 1);
            sg.lineRel('A', +dist, -dist);
            sierpA(level - 1);
        }
    }

    private void sierpB(int level) {
        if (level > 0) {
            sierpB(level - 1);
            sg.lineRel('B', -dist, +dist);
            sierpC(level - 1);
            sg.lineRel('B', 0, +2 * dist);
            sierpA(level - 1);
            sg.lineRel('B', +dist, +dist);
            sierpB(level - 1);
        }
    }
    private void sierpC(int level) {
        if (level > 0) {
            sierpC(level - 1);
            sg.lineRel('C', -dist, -dist);
            sierpD(level - 1);
            sg.lineRel('C', -2 * dist, 0);
            sierpB(level - 1);
            sg.lineRel('C', -dist, +dist);
            sierpC(level - 1);
        }
    }

    private void sierpD(int level) {
        if (level > 0) {
            sierpD(level - 1);
            sg.lineRel('D', +dist, -dist);
            sierpA(level - 1);
            sg.lineRel('D', 0, -2 * dist);
            sierpC(level - 1);
            sg.lineRel('D', -dist, -dist);
            sierpD(level - 1);
        }
    }
}

class SimpleGraphics {
    private Graphics g = null;
    private int x = 0, y = 0;

    public SimpleGraphics(Graphics g) {
        setGraphics(g);
    }

    public void setGraphics(Graphics g) {
        this.g = g;
    }

    public void goToXY(int x, int y) {
        this.x = x;
        this.y = y;
    }

    public void lineRel(char s, int deltaX, int deltaY) {
        g.drawLine(x, y, x + deltaX, y + deltaY);
        x += deltaX;
        y += deltaY;
    }
}
```

The following Logo program draws a Sierpiński curve by means of recursion.

```
to half.sierpinski :size :level
 if :level = 0 [forward :size stop]
 half.sierpinski :size :level - 1
 left 45
 forward :size * sqrt 2
 left 45
 half.sierpinski :size :level - 1
 right 90
 forward :size
 right 90
 half.sierpinski :size :level - 1
 left 45
 forward :size * sqrt 2
 left 45
 half.sierpinski :size :level - 1
end
```

```
to sierpinski :size :level
 half.sierpinski :size :level
 right 90
 forward :size
 right 90
 half.sierpinski :size :level
 right 90
 forward :size
 right 90
end
```

## References

1. ^ Platzman, Loren K.; Bartholdi, John J., III (1989). "Spacefilling curves and the planar traveling salesman problem". *Journal of the Association of Computing Machinery*. **36** (4): 719–737.
2. ^ Bartholdi, John J., III. "Some combinatorial applications of spacefilling curves". Georgia Institute of Technology.

## See also

- Hilbert curve
- Koch snowflake
- Moore curve
- Peano curve
- Sierpiński arrowhead curve
- List of fractals by Hausdorff dimension
- Recursion (computer science)
- Sierpinski triangle

Wikimedia Commons has media related to ***Sierpiński curve***.