The Craft of Coding

Musings on programming

Recursive patterns – the Sierpinski curve

08/05/2018

Processing can be used to show any number of different types of visualization, and one of the more interesting ones is recursive in nature. The Sierpinski curve is an exceptional example of mutual recursion. An exceptional algorithm can be found in Rohl [1], which is derived from Wirth[2].

The Sierpinski curve is what Wirth called aesthetically sophisticated. What seems like a curve with a simple building block is much more. This is because the difference between Sierpinski, and "simpler" curves like Hilbert is that Sierpinski curves are **closed,** i.e. without crossovers. The basic recursion schema is an open curve, and consists of four components which are connected by links which do not belong to the recursion pattern. The curve is drawn in a clockwise direction starting from the bottom left-hand corner. The components are marked N, E, S, W reflecting the direction in which they are drawn.

The components of a curve of order **k** are then constructed from the components of the curve of order **k-1**, and joined by diagonal, and horizontal or vertical lines. For example the **N** component of a curve of order k, is composed of curves of order k-1 in the sequence N→E→W→N, and is joined with lines in the NE, N, and NW directions respectively.

In Processing we first create a function **lineTo(x,y)** which draws a line from the current position to a new position **(x,y)**, and then updates the current position.

```
void lineTo(float newX, float newY) {
   line(cx, cy, newX, newY);
   fill(0);
   cx = newX;
   cy = newY;
}
```

it uses two global variables, **cx**, and **cy** to maintain the current drawing position. Next we derive the eight direction drawing functions:

```
void lineN(){ lineTo(cx,cy-2*h);}
void lineS(){ lineTo(cx,cy+2*h);}
void lineE(){ lineTo(cx+2*h,cy);}
void lineW(){ lineTo(cx-2*h,cy);}

void lineNW(){ lineTo(cx-h,cy-h);}
void lineNE(){ lineTo(cx+h,cy-h);}
void lineSE(){ lineTo(cx+h,cy+h);}
void lineSW(){ lineTo(cx-h,cy+h);}
```

Here is the main function, **sierpinskiCurve()**:

```
void sierpinskiCurve(int level) {
   sierN(level);
   lineNE();
   sierE(level);
   lineSE();
   sierS(level);
   lineSW();
   sierW(level);
   lineNW();
}
```

This draws the four separate curves, and joins them together. Here is the setup, and invoking code:

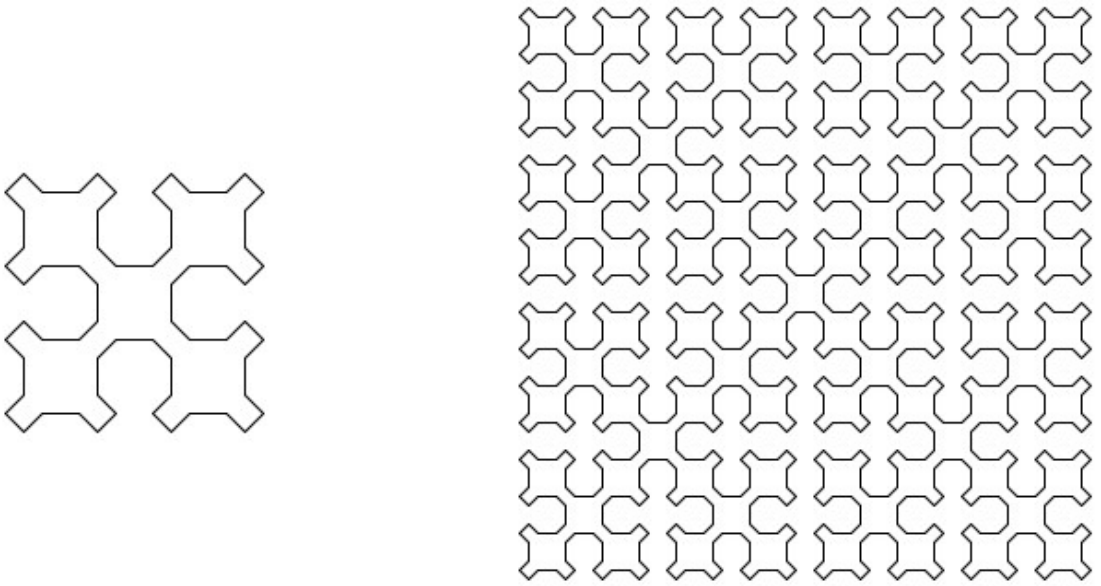
```
float cx;
float cy;
int h;

void setup() {
   size(800, 800);
   cx = width/2;
   cy = height;
}

void draw() {
   background(255);
   stroke(0);
   h = 3;
   sierpinskiCurve(4);
   noLoop();
}
```

And finally the four functions N, E, S, and W:

```
void sierN(int i){
   if (i == 1) {
      lineNE(); lineN();
      lineNW();
   }
   else {
      sierN(i-1); lineNE();
      sierE(i-1); lineN();
      sierW(i-1); lineNW();
      sierN(i-1);
   }
}

void sierE(int i){
   if (i == 1) {
      lineSE(); lineE();
      lineNE();
   }
   else {
      sierE(i-1); lineSE();
      sierS(i-1); lineE();
      sierN(i-1); lineNE();
      sierE(i-1);
```

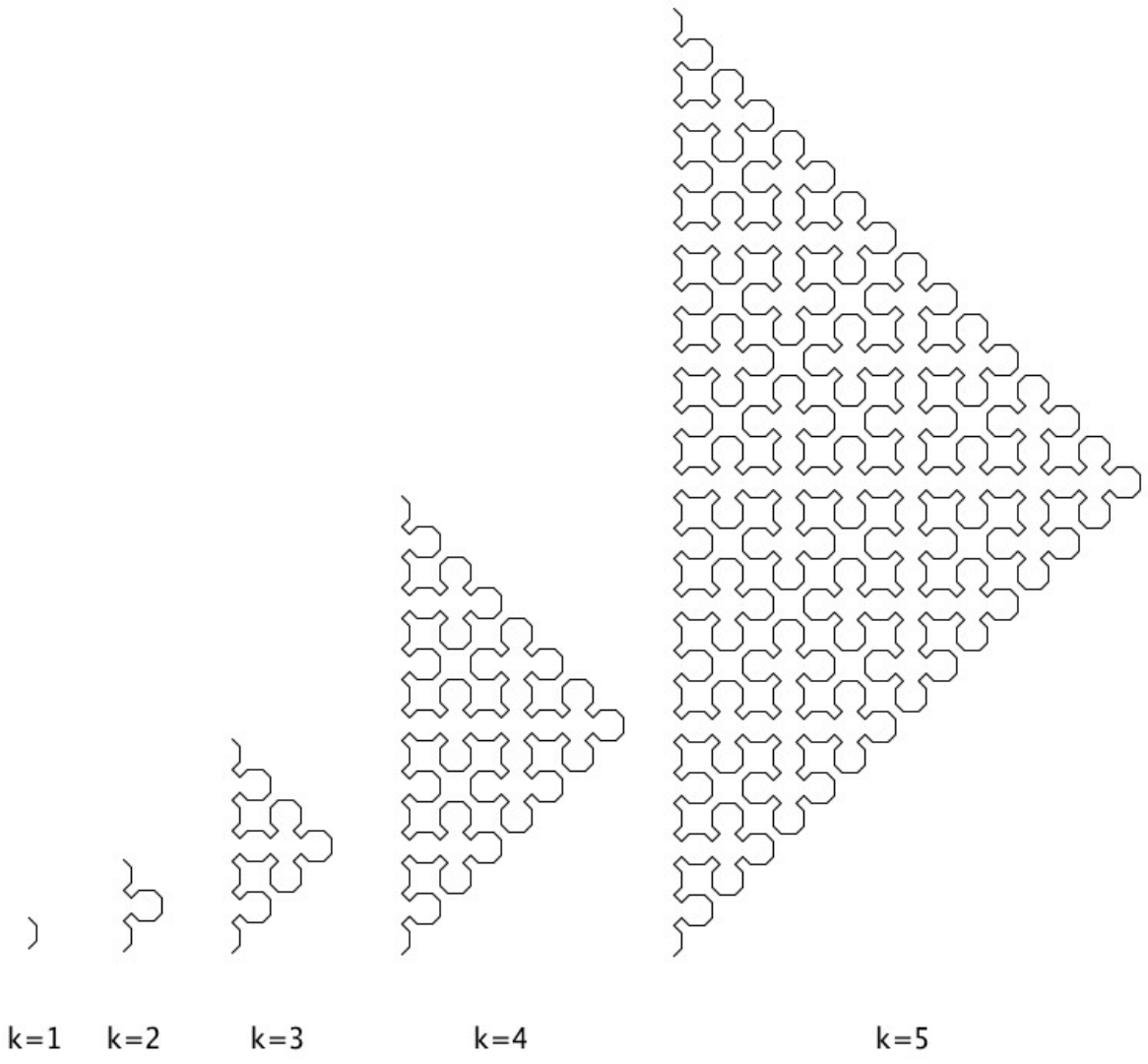
```
    }
}

void sierS(int i){
    if (i == 1) {
        lineSW(); lineS();
        lineSE();
    }
    else {
        sierS(i-1); lineSW();
        sierW(i-1); lineS();
        sierE(i-1); lineSE();
        sierS(i-1);
    }
}

void sierW(int i){
    if (i == 1) {
        lineNW(); lineW();
        lineSW();
    }
    else {
        sierW(i-1); lineNW();
        sierN(i-1); lineW();
        sierS(i-1); lineSW();
        sierW(i-1);
    }
}
```

How does it work? Below are two examples: On the left, is a Sierpinski curve with **h**=10, and **k**=2, or the right, a denser curve with **h**=5, and **k**=4.



(https://craftofcoding.files.wordpress.com/2018/05/sierpinskieg.jpg)

Here is an example of generating just **sierN(k)**, where k=1,2,3,4, and 5.



(https://craftofcoding.files.wordpress.com/2018/05/sierpinskieg2.jpg)

Looking closely at the code of course, you will notice the amount of mutual recursion going on to generate these complex shapes.

[1] Rohl, JS., Recursion via Pascal, Cambridge University Press (1984)
[2] Wirth, N., Algorithms + Data Structures = Programs, Prentice Hall (1976)

Posted in: recursion | *Tagged: mutual recursion, Processing language, recursive algorithm, Sierpinski curve*

One thought on “Recursive patterns – the Sierpinski curve”

1. **GIFGUIDE2CODE** says: 09/05/2018 at 11:35 pm REPLY
   Such lovely patterns. Nice post!