Fractals with METAPOST

Martin Geisler <gimpster@gimpster.com>

February 16, 2002

METAPOST is probably the best program to use, when it comes to generate graphics for inclusion in LaTeX and pdfLaTeX. There's several features of METAPOST that helps give it an edge:

- The graphics generated by METAPOST is a particular simple form of PostScript that is converted on-the-fly when inserted into pdfLaTeX.
- You can include labels in your figures, and the labels can come from LaTeX. It's no problem to label a figure with a complicated mathematical expression, as METAPOST knows the dimensions of the label and will position it correctly.
- As the METAPOST language is a programming language, the drawings you produce with it are precise. No more hand-drawn sketches of graphs — with METAPOST it's easy to render the correct graph for a given function.

I've used METAPOST to draw the fractals in the article. Fractals are interesting objects in their own right, but they're also interesting to draw using METAPOST, as they're defined in mathematical terms.

I've drawn three different fractals: Koch Curves in Figure 1 on the following page, Hilbert Curves in Figure 3 on page 3 and Sierpinski's Sieve in figure 4 on page 5.

You can find lot's of information about fractals at this very comprehensize webpage: `http://mathworld.wolfram.com/topics/Fractals.html`.

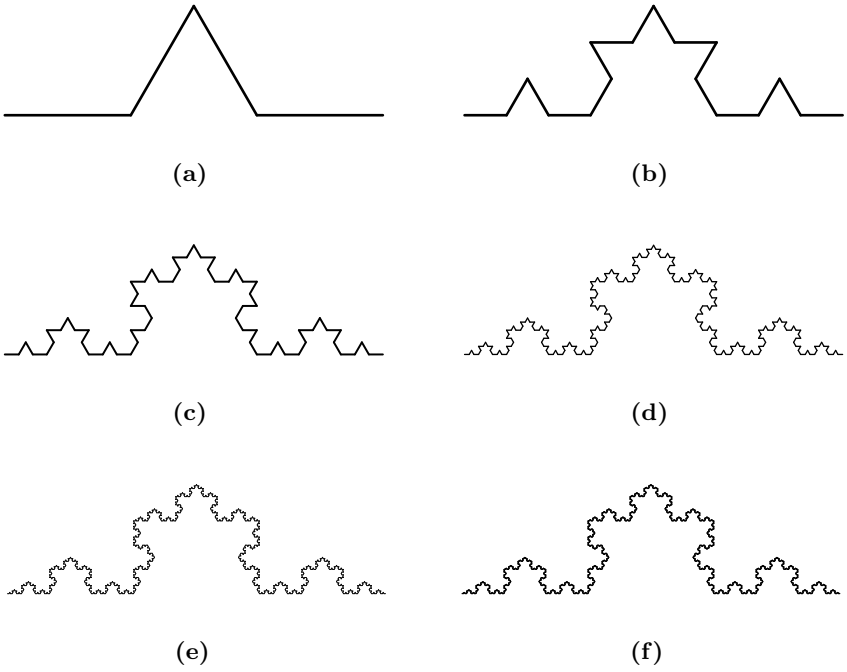

Figure 1: The first few iterations of a Koch-curve.

Koch Curve

```
def koch(expr a, b, n) =
  begingroup
    if n = 0:
      draw a--b;
    else:
      koch(a, 1/3[a,b], n-1);
      koch(1/3[a,b], 2/3[a,b] rotatedabout(1/3[a,b], 60), n-1);
      koch(2/3[a,b] rotatedabout(1/3[a,b], 60), 2/3[a,b], n-1);
      koch(2/3[a,b], b, n-1);
    fi
  endgroup
enddef;
```

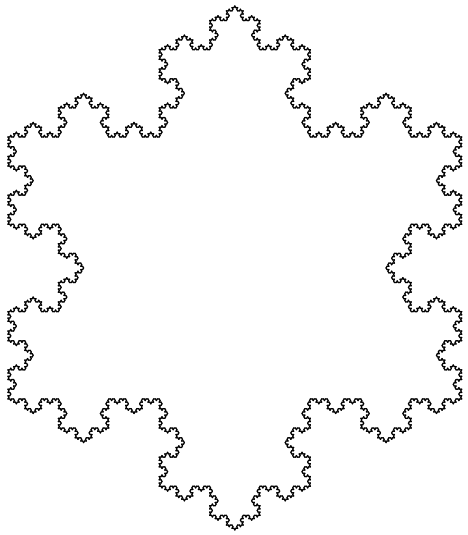

Figure 2: This is result of combining three Koch Curves as the sides of a are a Koch Snowflake.

Hilbert Curve

```
def hilbert(expr p, length, n) =
  begingroup
    save curve, T;
    path curve;
    transform T[];
    T[1] = identity scaled 1/2
                    rotated 90
                    reflectedabout((0, 0), (0, 1))
                    shifted (0.5*length, -1*length);
    T[2] = identity scaled 1/2;
    T[3] = identity scaled 1/2
                    shifted (0.5*length, 0*length);
    T[4] = identity scaled 1/2
                    reflectedabout((0, 0), (1, -1))
                    shifted (0.5*length, -0.5*length);
    curve = ((0.25, -0.75)*length)--((0.25, -0.25)*length)--
            ((0.75, -0.25)*length)--((0.75, -0.75)*length)
            shifted p;
    for i = 1 upto n:
      curve := (curve transformed T[1])--(curve transformed T[2])--
               (curve transformed T[3])--(curve transformed T[4]);
    endfor;
    draw curve scaled ((2**(n+1))/(2**(n+1) - 1));
  endgroup
enddef;
```

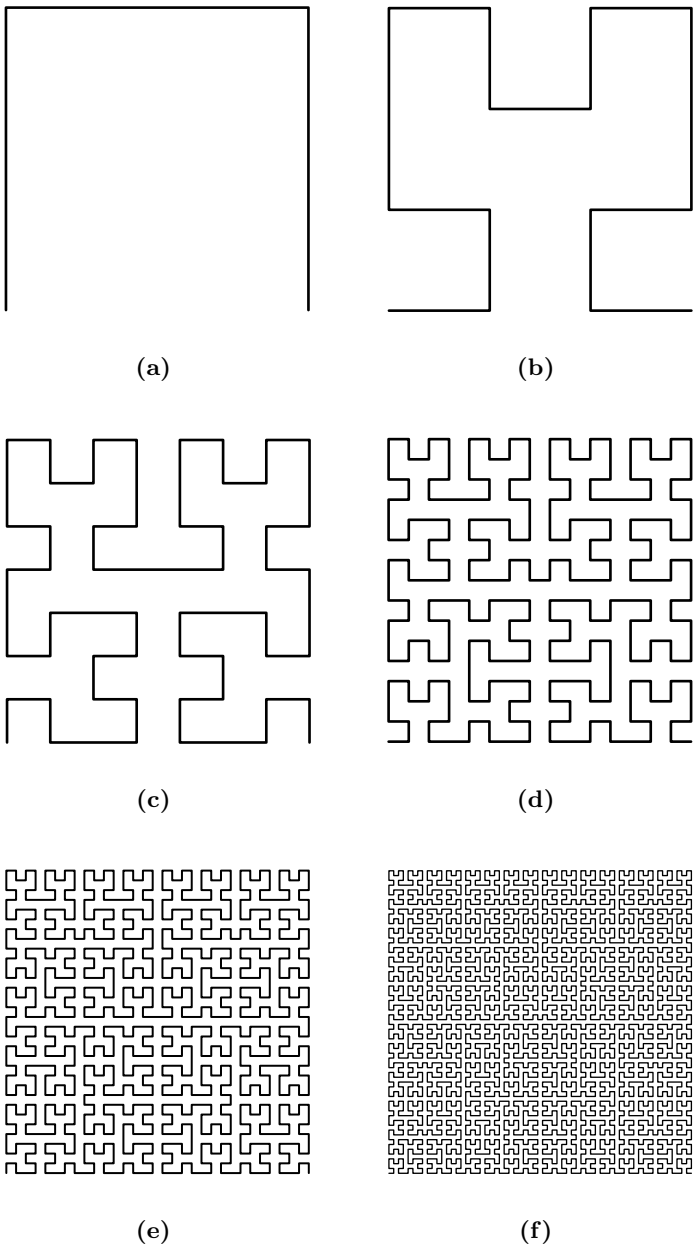

Figure 3: The first six iterations of a Hilbert curve.

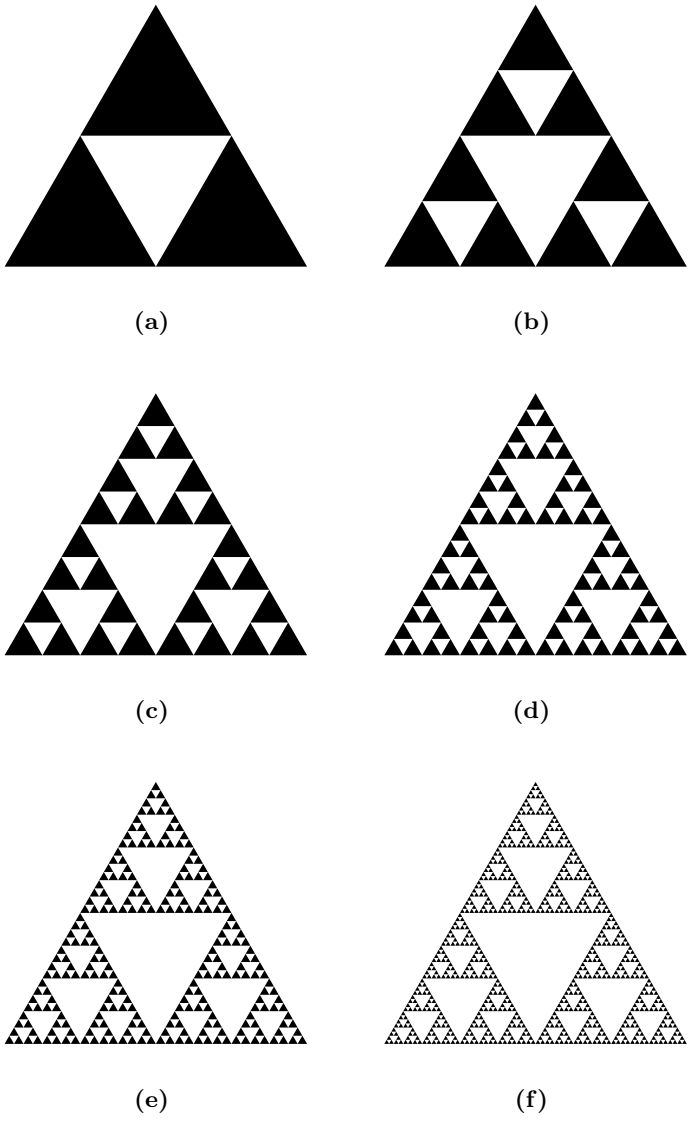

Figure 4: The first six iterations of Sierpinski's Sieve or Triangle as it's also called.

Sierpinski Sieve

```
def sierpinski(expr a, b, n) =
  if n = 0:
    fill a--(b rotatedabout(a, 60))--b--cycle;
  else:
    sierpinski(a, 0.5[a,b], n-1);
    sierpinski(0.5[a,b], b, n-1);
    sierpinski(0.5[a,b rotatedabout(a, 60)],
               0.5[a rotatedabout(b, -60),b], n-1);
  fi;
enddef;
```