Ventrella's terdragon in METAPOST

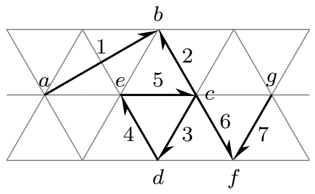
Linus Romer

Abstract

This article shows how to create a vector graphic similar to the terdragon described by Ventrella (2019) in METAPOST.

1 Generating the fractal path

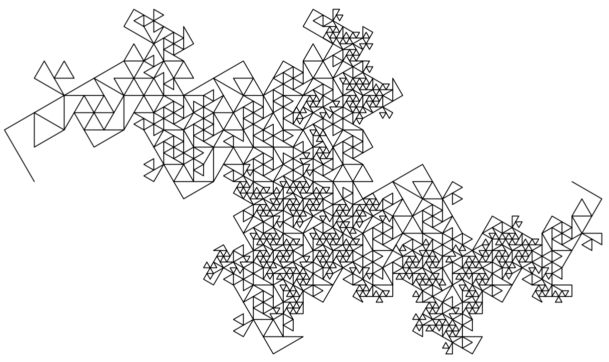
The fractal used for the terdragon is based on the following recursive replacement pattern on a triangular grid. Reverse arrows indicate segments that are rotated by 180°:



This pattern can be implemented as a recursive macro with recursion depth n:

```
vardef dragon(expr a,g,n) =
 save p,b,c,d,e,f; path p;
 if n > 0:
  pair b,c,d,e,f;
  e = 1/3[a,g]; c = 2/3[a,g];
  b = .5[a,g]+sqrt(3)/2*((c-e) rotated 90);
  d = c+e-b; f = g+e-b;
  p = dragon(a,b,n-1)
      & reverse dragon(c,b,n-1)
      & dragon(c,d,n-1) & dragon(d,e,n-1)
      & dragon(e,c,n-1) & dragon(c,f,n-1)
      & reverse dragon(g,f,n-1);
 else:
  p = a -- g;
 fi
 p % the returned path
enddef;
```

Using `draw dragon((0,0),(300,0),4);` you will get the following figure:



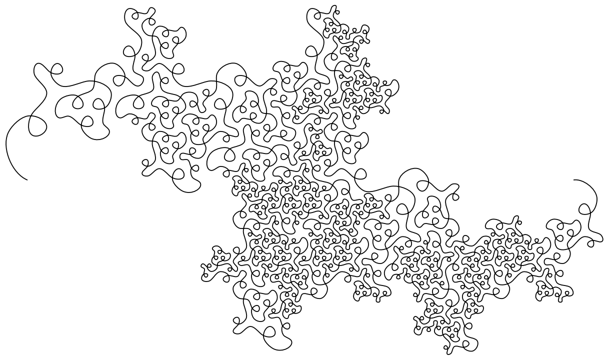
2 Rounding the vertices to curves

In the previous picture many of the vertices are revisited at different travel times on the path. This self-contacting behaviour can be avoided in different ways. In order to keep the number of path points small, weighted averages between neighbour vertices are used to draw smooth Bézier curves through them:

```
def roundcorners expr p =
  point 0 of p
  for i = 1 upto length(p)-1:
   .. tension 1.2
   .. ( .64*(point i of p)
       + .18*(point i-1 of p)
       + .18*(point i+1 of p) )
  endfor
  .. tension 1.2 .. point length(p) of p
enddef;
```

The tension 1.2 and the weights 0.64 and 0.18 are somewhat arbitrary. The following figure is produced by:

```
draw roundcorners dragon((0,0),(300,0),4);
```



It can now be observed that this terdragon variation is self-crossing.

3 Stroking the path dynamically

A dynamic stroking of the path is achieved by making the stroke width proportional to the distance between the points:

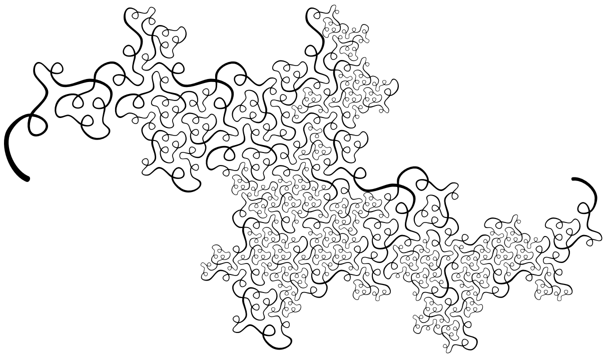
```
vardef dynamicdraw expr p =
 save n,l,r,s;
 pair l[],r[];
 numeric n; n = length(p);
 for i = 0 upto n:
  l[i] - r[i]
  = unitvector(direction i of p rotated 90)
    * ( length(point max(1,i) of p
             - point max(i-1,0) of p)
      + length(point min(n,i+1) of p
      - point min(i,n-1) of p) )
    * .08;
  r[i] + l[i] = 2*point i of p;
```

doi.org/10.47397/tb/43-3/tb135romer-terdragon

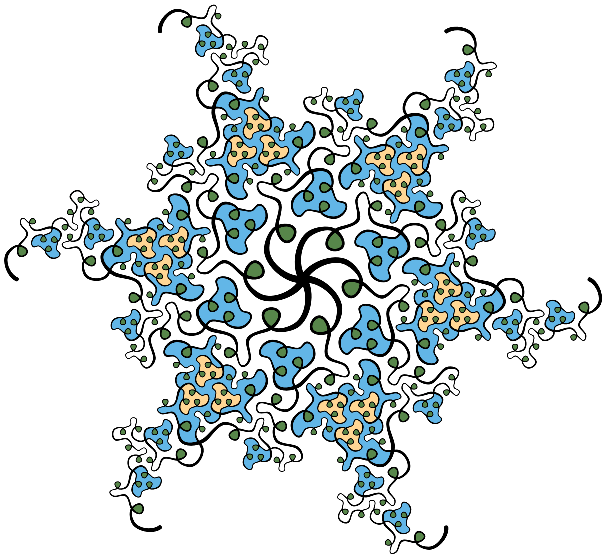
```
 endfor
 fill l[0]{direction 0 of p}
 for i=1 upto n:
  .. tension 1.2 .. l[i]{direction i of p}
 endfor
 for i=n downto 0:
  .. tension 1.2 .. r[i]{-direction i of p}
 endfor .. tension 1.2 .. cycle;
enddef;
```

The width scale 0.08 may be changed according to one's personal taste.

Replacing `draw` by `dynamicdraw` changes the last figure to the following:



If desired, a vector drawing program like *Inkscape* may be used to fill the enclosed areas with colour. After rotating a coloured (though grayscaled for the printed *TUGboat*) terdragon with recursion depth 3 six times, with respect to its left end, you will get the following figure:



For more information about the terdragon, please see the references, among numerous other books and articles.

References

Ventrella, Jeffrey. *Brainfilling Curves — A Fractal Bestiary*. Eyebrain Books, 2012.

Ventrella, Jeffrey. “Portraits from the Family Tree of Plane-filling Curves”. In *Proceedings of Bridges 2019: Mathematics, Art, Music, Architecture, Education, Culture*, edited by S. Goldstine, D. McKenna, and K. Fenyvesi, pages 123–130, Phoenix, Arizona. Tessellations Publishing, 2019. Available online at `archive.bridgesmathart.org/2019/bridges2019-123.pdf`.

◇ Linus Romer
Ahornstrasse 8
Uznach, 8730
Switzerland