Functional Geometry Description of Escher's Fish

Milton Mazzarri

Feb 1, 2017

Houston Elixir Meetup

Introduction

Square Limit

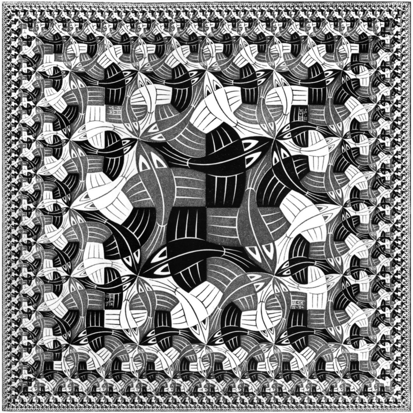

Figure 1: M.C. Escher's Square Limit

Source: `https://www.wikiart.org/en/m-c-escher/square-limit`

Functional Geometry

Functional Geometry is a paper by Peter Henderson[1, 2], which deconstructs the M.C. Escher woodcut “Square Limit”.

> *A picture is an example of a complex object that can be described in terms of its parts. Yet a picture needs to be rendered on a printer or a screen by a device that expects to be given a sequence of commands. Programming that sequence of commands directly is much harder than having an application generate the commands automatically from the simpler, denotational description.*

Denotational description v. Explicit Sequence of Commands

```elixir
Code.require_file("vector.ex", Path.join(__DIR__, "../../lib"))
Code.require_file("func_geo.ex", Path.join(__DIR__, "../../lib"))

alias FuncGeo, as: F

f = F.grid(
        7, 7,
        F.polygon([{2, 1}, {2, 6}, {5, 6}, {5, 5}, {3, 5},
                   {3, 4}, {4, 4}, {4, 3}, {3, 3}, {3, 1}]))

F.plot(f, "f.ps")
```

```postscript
500 500 scale
.1 .1 translate
0 setlinewidth
0 0 moveto 1 0 lineto 1 1 lineto 0 1 lineto 0 0 lineto
0.42857142857142855  0.14285714285714285 moveto 0.2857142857142857 0.14285714285714285 lineto
0.2857142857142857  0.14285714285714285 moveto 0.2857142857142857 0.8571428571428571 lineto
0.2857142857142857  0.8571428571428571 moveto 0.7142857142857143 0.8571428571428571 lineto
0.7142857142857143  0.8571428571428571 moveto 0.7142857142857143 0.7142857142857143 lineto
0.7142857142857143  0.7142857142857143 moveto 0.42857142857142855 0.7142857142857143 lineto
0.42857142857142855  0.7142857142857143 moveto 0.42857142857142855 0.5714285714285714 lineto
0.42857142857142855  0.5714285714285714 moveto 0.5714285714285714 0.5714285714285714 lineto
0.5714285714285714  0.5714285714285714 moveto 0.5714285714285714 0.42857142857142855 lineto
0.5714285714285714  0.42857142857142855 moveto 0.42857142857142855 0.42857142857142855 lineto
0.42857142857142855  0.42857142857142855 moveto 0.42857142857142855 0.14285714285714285 lineto
stroke
showpage
```

Figure 2: Denotational description v. explicit sequence of commands

Basic Operations

f

Note

The image it is located within a frame, but we do not consider the frame to be part of the picture.

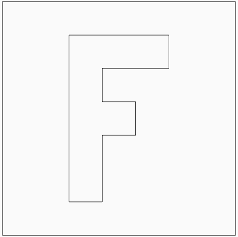

Figure 3: The value `f` denotes the picture of the letter F

Basic operations on pictures

- $rot(picture) :: picture$
- $flip(picture) :: picture$
- $rot45(picture) :: picture$
- $above(picture, picture) :: picture$
- $beside(picture, picture) :: picture$
- $over(picture, picture) :: picture$

Rotation

Rotates a picture 90° anticlockwise.

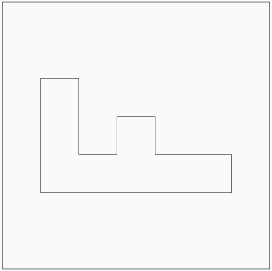

Figure 4: `rot(f)`

Flip

Flip a picture through its vertical centre axis.

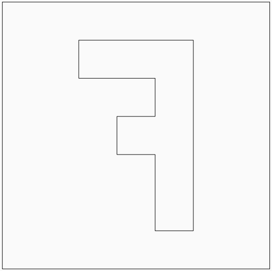

Figure 5: `flip(f)`

Rotation and Flip

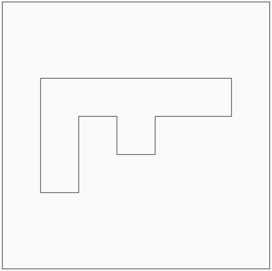

Figure 6: `rot(flip(f))`

Rotation 45°

Rotates a picture about its top left corner, through 45° anticlockwise.

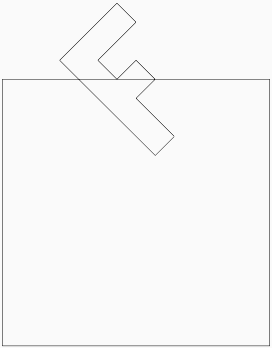

Figure 7: `rot45(f)`

Above

`above(p, q)` is the picture that has `p` in the upper half of its locating box and `q` in the lower half.

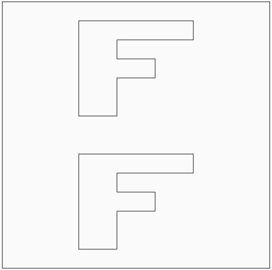

Figure 8: `above(f, f)`

Beside

`beside(p, q)` is the picture that has `p` in the left half of its locating box and `q` in the right half.

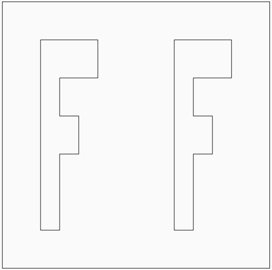

Figure 9: `beside(f, f)`

above/beside combination

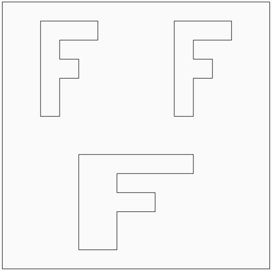

Figure 10: `above(beside(f, f) f)`

Superposition

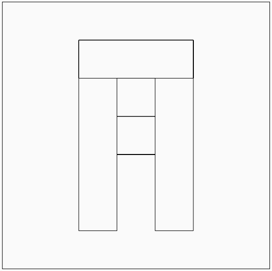

Figure 11: `over(f, flip(f))`

Laws

Laws

$$rot(rot(rot(rot(p)))) = p$$

Unit Test

```
test ``p is equal after fourth continuos rotations'' do
 p = p()
 p_rotated = p |> rot() |> rot() |> rot() |> rot()

 assert p_rotated.({0, 0}, {1, 0}, {0, 1}) == p.({0, 0}, {1, 0}, {0, 1})
end
```

Laws

$$rot(above(p, q)) = beside(rot(p), rot(q))$$

Unit Test

```
test ``rot(above(p, q)) must be equal to beside(rot(p), rot(q))'' do
 p = p()

 p_rotated = rot(above(p, p))
 p_beside = beside(rot(p), rot(p))

 assert
   p_rotated.({0, 0}, {1, 0}, {0, 1}) == p_beside.({0, 0}, {1, 0}, {0, 1})
end
```

Laws

$$rot(beside(p, q)) = above(rot(q), rot(p))$$

$$flip(beside(p, q)) = beside(flip(q), flip(p))$$

Square Limit

Basic patterns: p, q

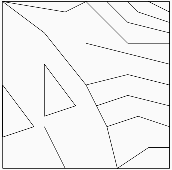

Figure 12: p

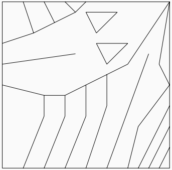

Figure 13: q

Basic patterns: r, s

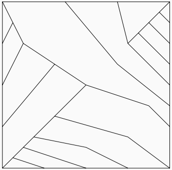

Figure 14: `r`

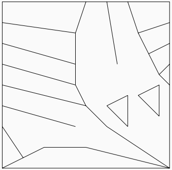

Figure 15: `s`

`t` shows how nicely the four basic fish tiles fit together

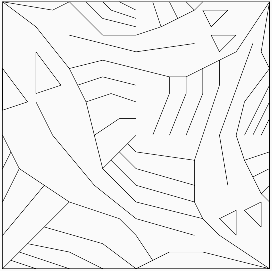

Figure 16: `t = quartet`p, q, r, s

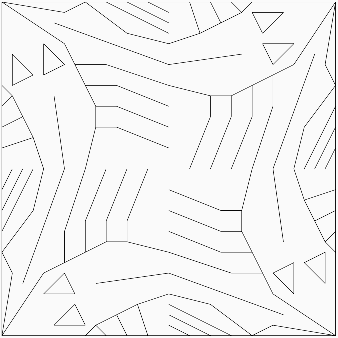

Figure 17: `u = cycle(rot(q))`

side1

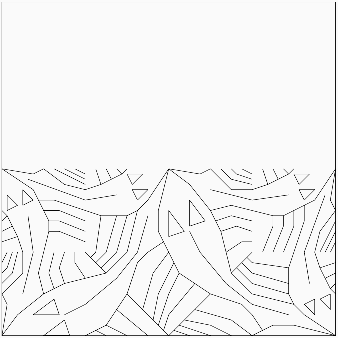

Figure 18: `side1 = quartet(blank, blank, rot(t), t)`

side2

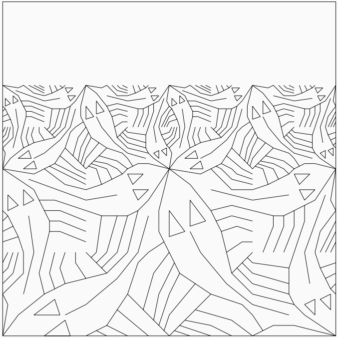

Figure 19: `side2 = quartet(side1, side1, rot(t), t)`

corner1

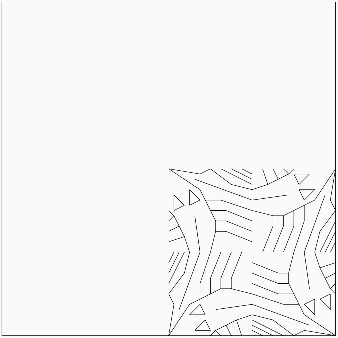

Figure 20: `corner1 = quartet(blank, blank, blank, u)`

corner2

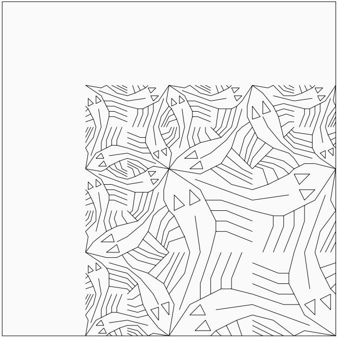

Figure 21: `corner2 = quartet(corner1, side1, rot(side1), u)`

corner

```
corner = nonet(corner2, side2, side2, rot(side2), u, rot(t),
rot(side2), rot(t), rot(q))
```

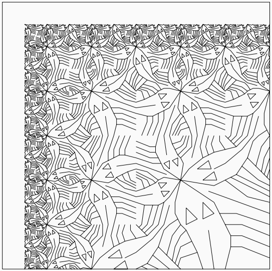

Figure 22: corner

squarelimit

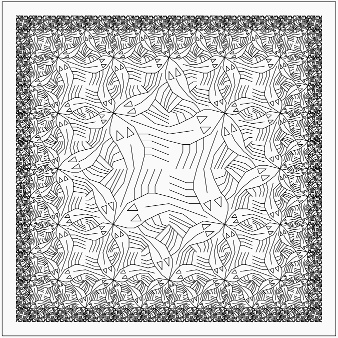

Figure 23: `squarelimit = cycle(corner)`

Implementation

Vectors

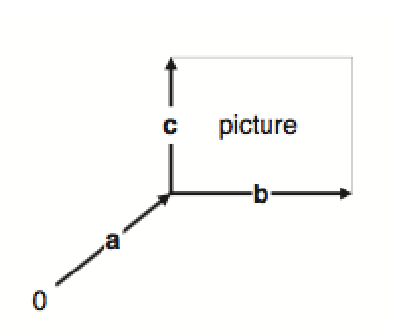

Figure 24: Basic vectors

Implementation

$$over(p, q)(a, b, c) = p(a, b, c) \cup q(a, b, c)$$

$$blank(a, b, c) = \{\}$$

$$beside(p, q)(a, b, c) = p(a, \frac{b}{2}, c) \cup q(a + \frac{b}{2}, \frac{b}{2}, c)$$

$$above(p, q)(a, b, c) = p(a, b, \frac{c}{2}) \cup q(a + \frac{c}{2}, b, \frac{c}{2})$$

Implementation

$$rot(p)(a, b, c) = p(a + b, c, -b)$$

$$flip(p)(a, b, c) = p(a + b, -b, c)$$

$$rot45(p)(a, b, c) = p(a + \frac{b + c}{2}, \frac{b + c}{2}, \frac{c - b}{2})$$

Demo

Future

- Add support for SVG Path.
- Escher's "Circuit Limit III" picture.
- Increase code coverage
- Include property-based tests (QuickCheck, Quixir)

Circuit Limit III

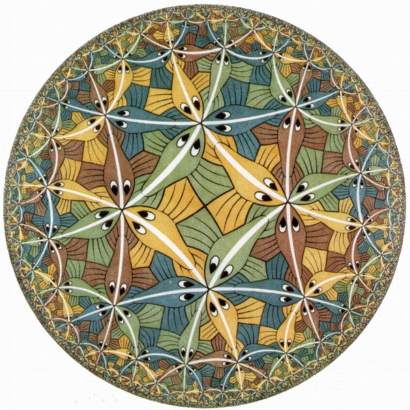

Figure 25: Circuit Limit III

Source: `http://mathstat.slu.edu/escher/upload/9/90/Circle-Limit-III.jpg`

Thanks!

https://github.com/milmazz/func_geo

https://github.com/milmazz/slides

References I

P. Henderson.
Functional geometry, 1982.

P. Henderson.
Functional geometry, 2002.