

FUNCTIONAL GEOMETRY

Peter Henderson
Oxford University Computing Laboratory
Programming Research Group
45 Banbury Road
Oxford OX2 6PE, U.K.

Abstract

A method of describing pictures is introduced. The equations, which describe the appearance of a picture, also form a purely functional program which can be used to compute the set of lines necessary to plot the picture on a graphical device. The method is illustrated by using it to describe the structure of one of the woodcuts of Maurits Escher.

Introduction

We shall define a data type *picture* and some operations upon it. In particular we shall show how *pictures* can be combined to form *pictures* by such operations as juxtaposition, rotation and superposition. The particular operations we have chosen are rather arbitrary but have been arrived at as the result of considerable experimentation. Hopefully the main example of this paper will convince the reader that they are adequate to a large class of illustrations.

A *picture* is a set of line segments. We can only illustrate a *picture* relative to a bounding box. Figure 1 shows a picture consisting of 25 line segments. It purports to depict a man. In all our illustrations the bounding box will be shown.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

It is not part of the picture.

If we choose different shaped bounding boxes, the same set of line segments lead to different illustrations as figures 2,3 and 4 show.

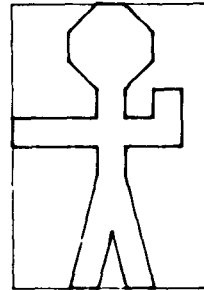


figure 1
man in box 14 by 20

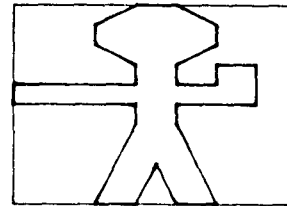


figure 2
man in box 20 by 14

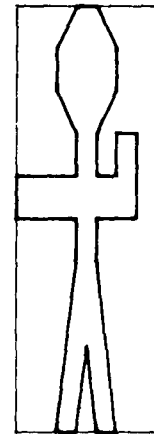


figure 3
man in box 10 by 30

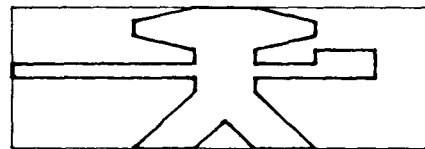


figure 4
man in box 30 by 10

The first operation which we shall need is one which allows us to build basic pictures from line segments. We define the function $grid : integer \times integer \times List(line\ segment) \rightarrow picture$ as follows. If m and n are two non-negative integer values then we construct a grid with x -coordinates ranging between 0 and m and with y -coordinates ranging between 0 and n . Each line segment then joins two points within the grid and thus can be represented by four integers x_0, y_0, x_1 and y_1 corresponding to its end points.

Figure 5 shows the 14 by 20 grid which was used to draft the picture which we have called *man*.

$man = grid(14, 20, ((6, 10, 0, 10), (0, 10, 0, 12), (0, 12, 6, 12), \dots, (4, 0, 6, 8), (6, 8, 6, 10)))$

Clearly *grid* can be used to draft any picture. We use it only to draft pictures which have little or no regularity.

The remaining operations all produce *pictures* from *pictures*. The operation

$flip : picture \rightarrow picture$

is such that $flip(p)$ is the reflection of p in a vertical axis exactly bisecting the picture. Figure 6 illustrates $flip(man)$.

The operation

$beside : integer \times integer \times picture \times picture \rightarrow picture$

is such that $beside(m, n, p, q)$ is the *picture* obtained by juxtaposing p to the left of q so that the ratio of their widths is m to n . Figures 7, 8 and 9 illustrate this operation.

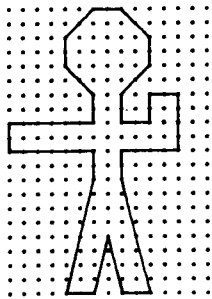


figure 5

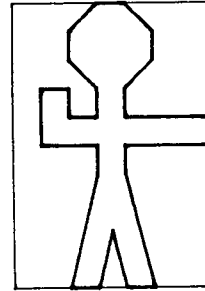


figure 6

$flip(man)$ in box 14 by 20

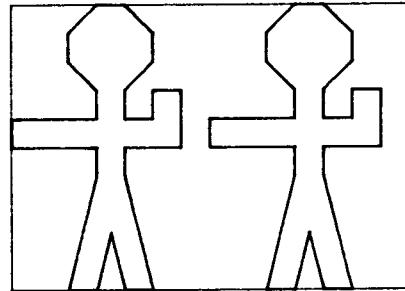


figure 7: $beside(1, 1, man, man)$
in box 28 by 20

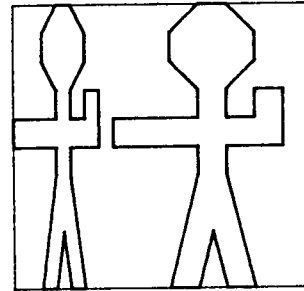


figure 8: $beside(1, 2, man, man)$
in box 21 by 20

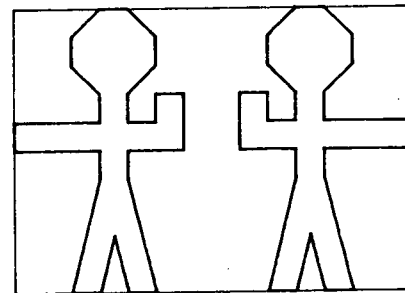


figure 9: $beside(1, 1, man, flip(man))$

Similarly $above(m,n,p,q)$ is the *picture* obtained by juxtaposing p above q so that the ratio of their heights is m to n . If we denote the *picture* with no line segments in it by nil then we can define

$fatboy = above(1,1,nil,man)$

$boy = beside(1,1,fatboy,nil)$

See figures 10,11,12,13 and 14. Notice in particular the difference between figures 13 and 14.

The operation

$rot : picture \rightarrow picture$

performs a 90° , anticlockwise rotation of the picture. The bounding box does not rotate. It is not part of the picture. Thus figure 15.

The operation

$overlay : picture \times picture \rightarrow picture$

is such that $overlay(p,q)$ is the *picture* which contains all the line segments of p and all the line segments of q . Thus figure 16.

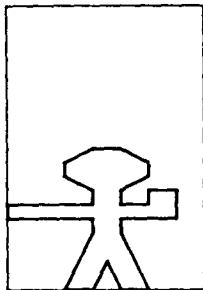


figure 10: fatboy

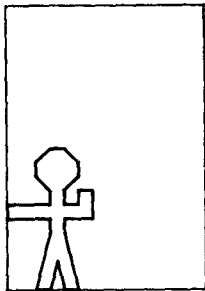


figure 11: boy

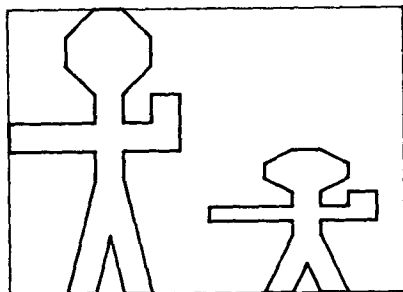


figure 12: beside(1,1,man,fatboy)

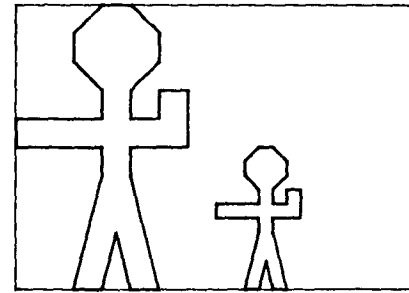


figure 13: beside(1,1,man,boy)

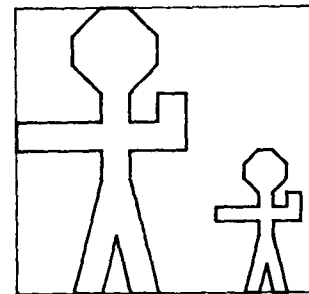


figure 14: beside(2,1,man,fatboy)

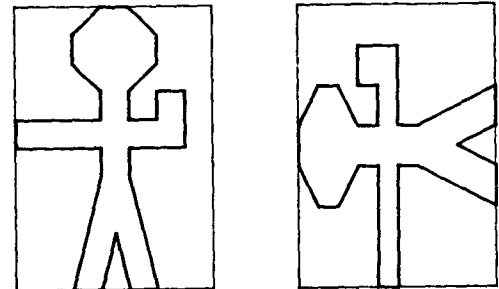


figure 15: man and rot(man)

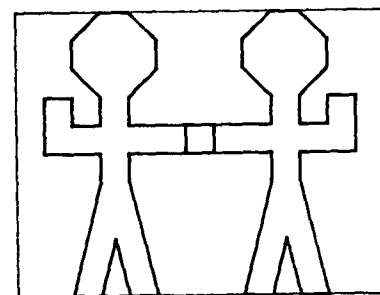


figure 16: overlay(p,flip(p))
where $p = beside(6,7,nil,man)$

Geometric interpretation

A bounding box has position and orientation. This can be fixed, relative to an origin, by three vectors \underline{a} , \underline{b} and \underline{c} , which respectively describe the position of the lower left corner of the box and the length and orientation of its sides. See figure 17. A bounding box can be a parallelogram.

Let us denote by $plot(p, \underline{a}, \underline{b}, \underline{c})$ the set of lines which will constitute an illustration of the picture p with respect to the bounding box $\underline{a}, \underline{b}, \underline{c}$.

If we have $p = grid(m, n, s)$ then for each $(x0, y0, x1, y1)$ in s the line $(\underline{a} + \underline{b} \times x0/m + \underline{c} \times y0/n, \underline{a} + \underline{b} \times x1/m + \underline{c} \times y1/n)$ is included in $plot(p, \underline{a}, \underline{b}, \underline{c})$.

The remaining operations are trivial.

$$\begin{aligned}
 plot(nil, \underline{a}, \underline{b}, \underline{c}) &= \emptyset \\
 plot(flip(p), \underline{a}, \underline{b}, \underline{c}) &= plot(p, \underline{a} + \underline{b}, -\underline{b}, \underline{c}) \\
 plot(rot(p), \underline{a}, \underline{b}, \underline{c}) &= plot(p, \underline{a} + \underline{b}, \underline{c}, -\underline{b}) \\
 plot(overlay(p, q), \underline{a}, \underline{b}, \underline{c}) &= \\
 &plot(p, \underline{a}, \underline{b}, \underline{c}) \cup plot(q, \underline{a}, \underline{b}, \underline{c}) \\
 plot(beside(m, n, p, q), \underline{a}, \underline{b}, \underline{c}) &= \\
 &plot(p, \underline{a}, \underline{b} \times m / (m+n), \underline{c}) \cup \\
 &plot(q, \underline{a} + \underline{b} \times m / (m+n), \underline{b} \times n / (m+n), \underline{c}) \\
 plot(above(m, n, p, q), \underline{a}, \underline{b}, \underline{c}) &= \\
 &plot(p, \underline{a} + \underline{c} \times n / (m+n), \underline{b}, \underline{c} \times m / (m+n)) \cup \\
 &plot(q, \underline{a}, \underline{b}, \underline{c} \times n / (m+n))
 \end{aligned}$$

The set of lines $plot(p, \underline{a}, \underline{b}, \underline{c})$ can be used to drive a plotter. It is sensible to be lazy and avoid building the entire set before beginning to consume it.

An Escher woodcut

M.C. Escher produced many fine woodcuts and lithographs. Among them is a variety where carefully designed animal shapes intertwine with each other to entirely cover the area of the illustration. Some are reproduced in "The World of M.C. Escher" [1]. One in particular, "Square Limit" which intertwines carefully constructed fish (plates 261 to 263), lends itself to description using the method of functional geometry. This illustration was chosen because, along with a reproduction of

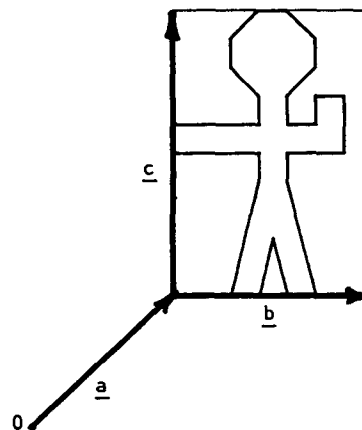


figure 17: a bounding box

the final woodcut, two sketches on graph paper are also published. These sketches provide a key to the way in which the picture was designed.

The decomposition given here is only one way of analysing the structure of "Square Limit". It almost certainly does not reflect the way that Escher himself saw the structure of his picture.

We shall begin by defining four basic *pictures*. These give, in a garish parody of Escher's original drawings, the general shape of each of three fishes from which the final picture will be constructed. Our objective is simply to reproduce a schematic version of "Square Limit" in which each of the fish is in its correct position and orientation.

All the *pictures* which we shall construct will be square. Frequently we shall require to take four of them and combine them in the obvious way to form a new square. We shall make use of the following operation for this purpose.

$$\begin{aligned}
 quartet(p1, p2, p3, p4) &= \\
 &above(1, 1, beside(1, 1, p1, p2), \\
 &\quad beside(1, 1, p3, p4))
 \end{aligned}$$

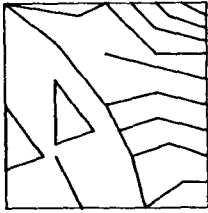


figure 18: p

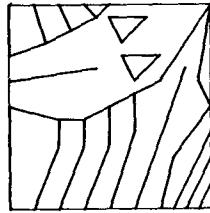


figure 19: q

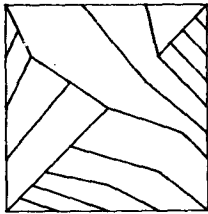


figure 20: r

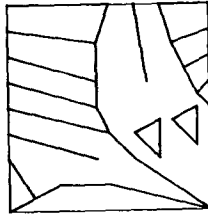


figure 21: s

Figures 18,19,20 and 21 show the four basic pictures which we shall call p, q, r and s . We shall see that they fit together in the most pleasing ways. Figure 22 shows the arrangement

$$t = \text{quartet}(p, q, r, s)$$

which is a new picture with some remarkable properties. Clearly p, q, r and s were designed so as to fit together in just this way.

Another picture which we shall need is constructed from four copies of the basic picture q , as shown in figure 23. This arrangement of four copies of a picture each rotated 90° relative to the next, also has general utility. Therefore let us define

$$\text{cycle}(p1) = \text{quartet}(p1, \text{rot}(\text{rot}(\text{rot}(p1))), \text{rot}(p1), \text{rot}(\text{rot}(p1)))$$

Using this definition we can describe the arrangement shown in figure 23 as

$$u = \text{cycle}(\text{rot}(q))$$

which we shall make use of later.

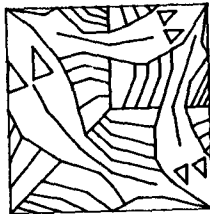


figure 22: $t = \text{quartet}(p, q, r, s)$

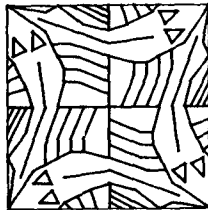


figure 23: $u = \text{cycle}(\text{rot}(q))$

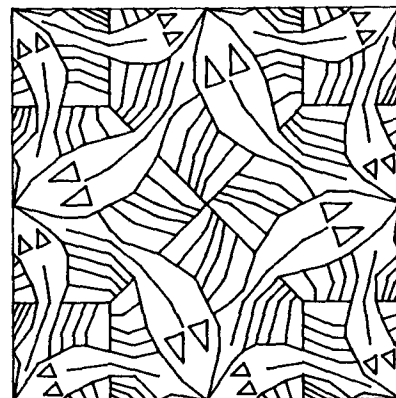


figure 24: $\text{cycle}(\text{rot}(t))$

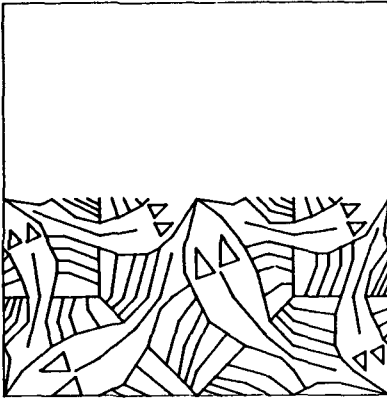


figure 25: side1



figure 26: side2

Consider now the arrangement of figure 25. This defines

$$side1 = quartet(nil, nil, rot(t), t)$$

which is unremarkable in itself until we define

$$side2 = quartet(side1, side1, rot(t), t)$$

which is shown in figure 26. Here we see the remarkable property of t , that when reduced to half its size, it sits happily on top of itself. Moreover, it does so in two quite distinct ways.

We could of course go on and define $side3, side4, \dots$ or even $side(n)$, but $side2$ will be sufficient to complete our task and tax our plotter.

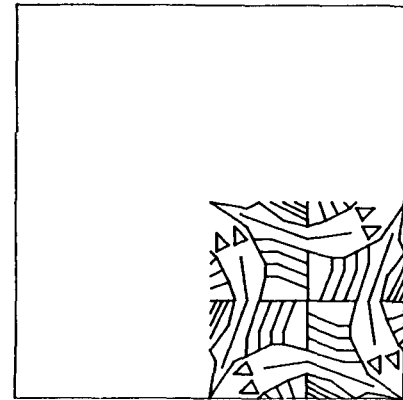


figure 27: corner1

Let us construct a corner in a similar manner. First define the almost empty

$$corner1 = quartet(nil, nil, nil, u)$$

shown in figure 27 and then

$$corner2 = quartet(corner1, side1, rot(side1), u)$$

which is shown in figure 28. Again we could continue in this vein but again it would be in vain for interesting though $corner3$ might be, we only need go as far as $corner2$.

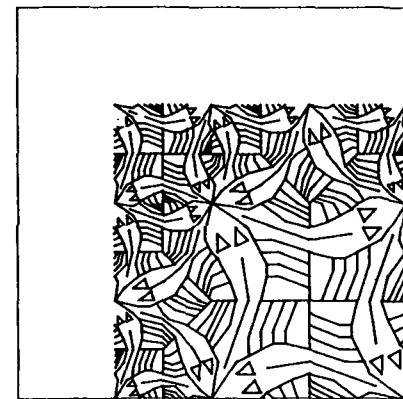


figure 28: corner2

The next and final step to "Square Limit" is less trivial. Before we take it we shall make a little detour.

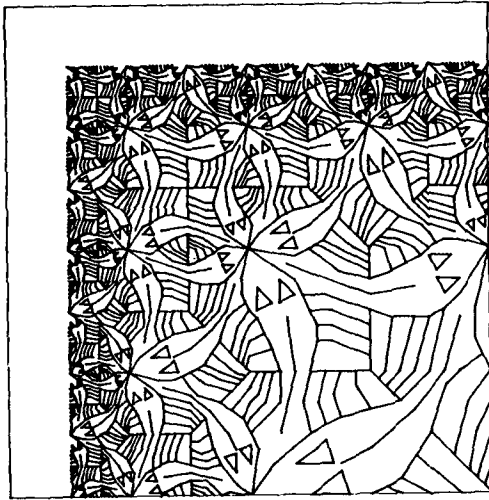


figure 29: pseudocorner

Clearly we could *cycle* any of the corners we have produced or postulated and generate a symmetric result. These are too regular to be very interesting. However a minor alteration to what could have been *corner3* leads to

```
pseudocorner = quartet(corner2,side2,
                       rot(side2),rot(t))
```

shown in figure 29. Note that we have replaced a slightly boring *u* by an interestingly irregular *rot(t)*. Now we can produce

```
pseudolimit = cycle(pseudocorner)
```

shown in figure 30. This is not unlike "Square Limit". Except near the edge, the fish are not yet in their correct orientation.

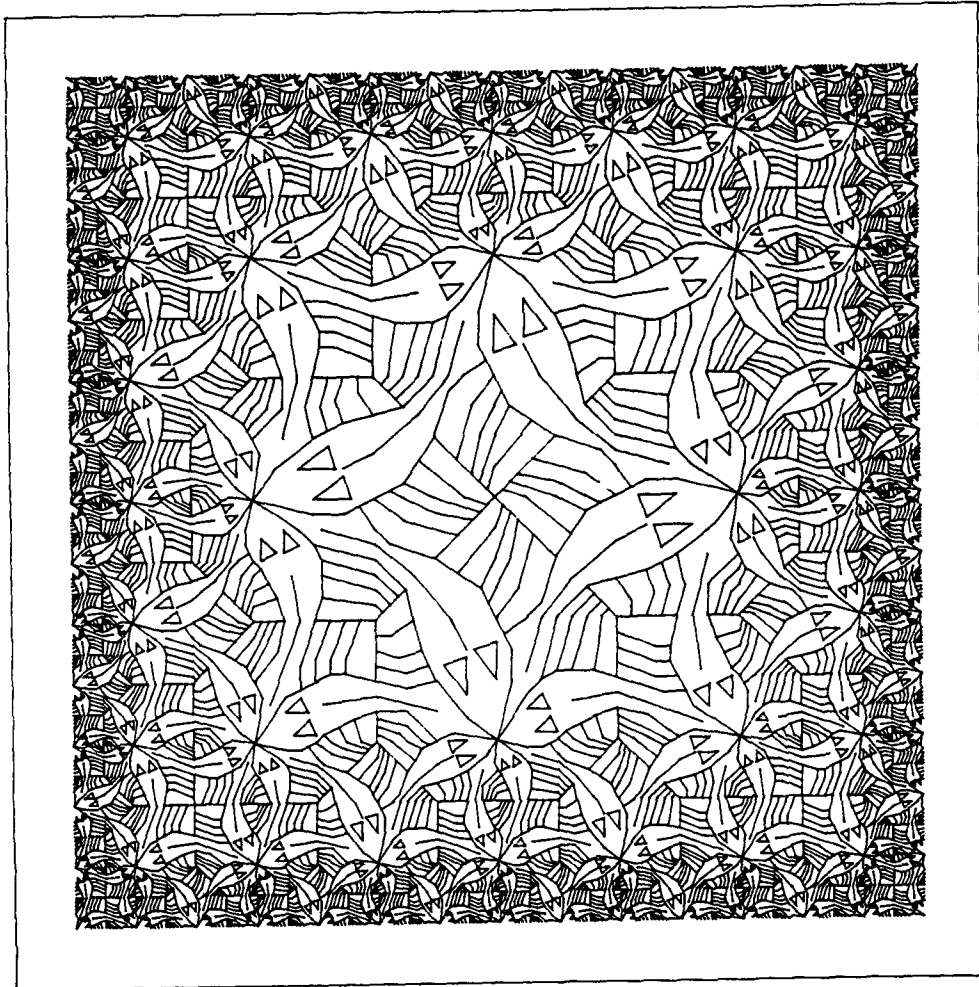


figure 30: pseudolimit

To resolve this final problem we need to define a function which will produce from nine pictures a regular three by three arrangement of them. We use the fact that the arguments of *above* and *beside* can be laid out in the text in the same positions as the corresponding pictures will occupy in the composition, to ensure that our definition is correct. At the same time we elevate this nice property to the new function.

```

nonet(p1,p2,p3,
      p4,p5,p6,
      p7,p8,p9) =
above(1,2,beside(1,2,p1,beside(1,1,p2,p3)),
      above(1,1,beside(1,2,p4,beside(1,1,p5,p6)),
          beside(1,2,p7,beside(1,1,p8,p9))))

```

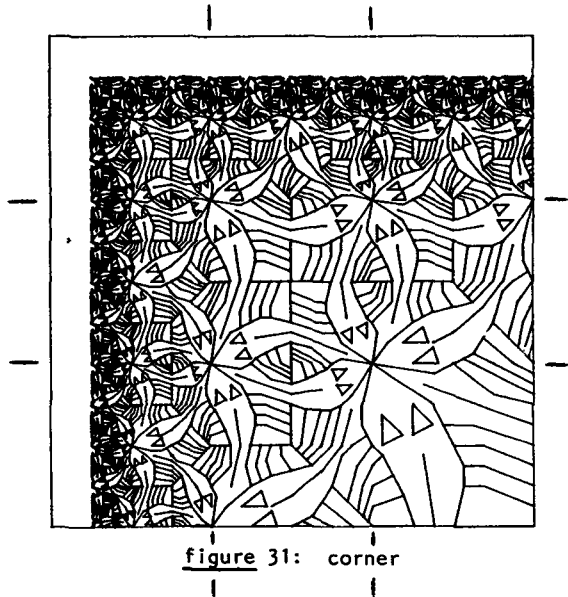
Now, finally, we can define what will be one corner of "Square Limit". It is a non-trivial arrangement of nine of the composite pictures which we have defined.

```

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

```

This *corner* is illustrated in figure 31. Observe that it is not symmetric about its main diagonal, as one might have expected. The definition of



corner shows this most clearly, with the two asymmetric occurrences of *rot(t)*.

So we reach the end of the story. Satisfyingly, for the author at least, figure 32 shows

```
squarelimit = cycle(corner)
```

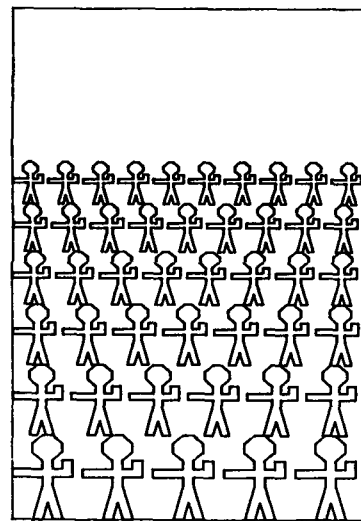
and all the fish are where they should be, all 412 of them.

[1] Locher J.L. (ed) "The world of M.C.Escher", Harry N. Abrams, Incorporated, New York (1971) ISBN 8109-0107-2

Acknowledgements

Many people have helped me to investigate the ideas presented here. Martin McLauchlan and Mary Sheeran each made implementations of these and many other basic operations and applied them to many problems, in particular that of integrated circuit layout. I have benefitted greatly from the comments of colleagues at the Universities of Edinburgh, Newcastle upon Tyne and Oxford. Les Belady gave me the book on Escher, six years ago, and generated all this fun and frustration. The illustrations were done using a functional geometry package implemented in UCSD Pascal by Mary Sheeran.

Post script As an exercise the interested reader might like to describe the following picture of a crowd of functional programmers.



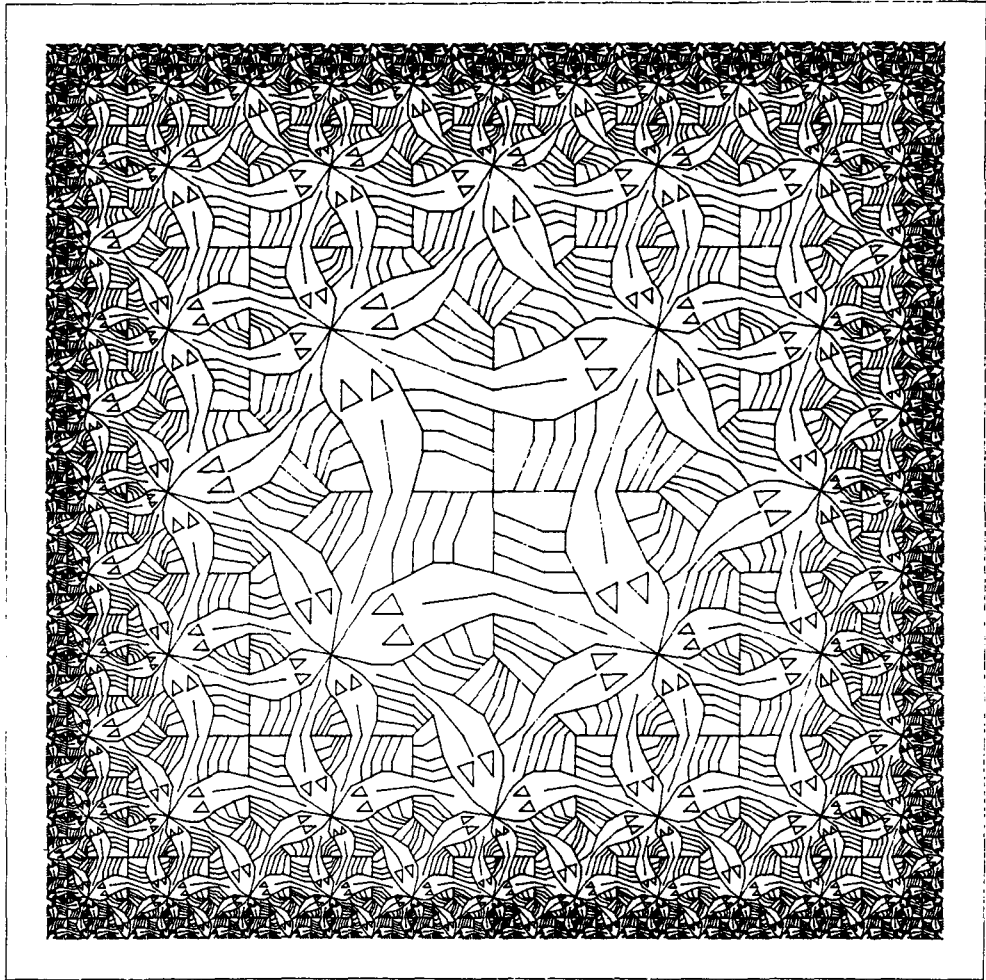


figure 32: square limit