Bijlage 5
Tiling in PostScript and METAFONT – Escher's wink

Kees van der Laan

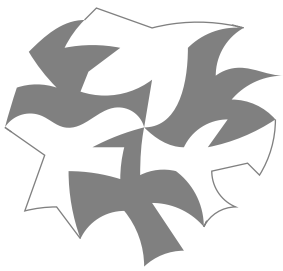
abstract
Drawing tilings by computer is discussed. Examples are borrowed from literature. New are their included METAFONT and PostScript programs, with sometimes a new variation of a picture.

keywords
Apollonius, Computer art, Douat, Dürer, education, EPS, Escher, graphics,METAFONT/Post, outline, Kepler, Mondrian, parquet, Penrose, PostScript, puzzles, Soddy, tiling, Truchet, turtle graphics, zero finding.

1 Introduction

Children like to play, like jigsaw puzzles. Tiling avant la lettre, where the form and color of the pieces matter.

Example *(Part of Escher's Sun and Moon)*



Dark birds in bright daylight or white birds in the dark?

What has this got to do with tiling?

Well ... it is a tiling of free forms. When considered as a non-tight tiling of either dark or white birds the others are spurious. Intriguing. Underneath is a grid of triangles, reshaped into birds.

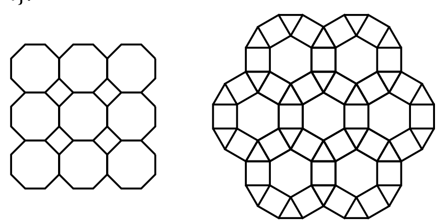
Escher anthropomorphised tiling by introducing creatures as decoration of the tiles as opposed to traditional tiling. Tiles used to take strict geometrical patterns especially the Islamic ones because of religion.

Escher considered lines as to be 2-sided, and exploited this in depth. Seen from one side the line is the boundary of a figure and looked upon from the other side gives meaning to another figure.

Tiling has all to do with filling up the plane. It is an old human activity of which beautiful results have been preserved, for example in the Alhambra.[1] In this note a few examples of tilings are shown next to their codes – METAFONT, or POSTSCRIPT – for drawing them by computer.

Intriguing and captivating are Escher's contributions. In some patterns he not only translates but also rotates and reflects. In others, the so-called limit cycles, he shrinks the size of the basic element gradually, suggestively infinitely. In his metamorphoses tiles change their form in crescendo.

Example *(Classification of homogeneous tiles)*
The following homogeneous, Archimedean tiles are classified in mathematical tiling theory by the polygons at each vertex. The left is denoted by $\{4, 8, 8\}$ – a square and 2 octagons join at each corner – and the right one by $\{3, 4, 6, 4\}$.[2]



However, no attempt will be made to treat mathematical tiling theory. It seems that a constructive computer-oriented approach and programming terminology are needed. I found it advantageous to concentrate on non-tight tilings with spurious elements. For example the right figure – extended infinitely – can be drawn by just squares appropriately positioned. The result will contain the spuri-

1. A thorough reference about tiling is: Tilings and patterns by Branko Grünbaum & G C Shephard. Freeman, New York. ISBN 0-7167-1193-1. Martin Gardner in his Scientific American notes has popularized various (mathematical) puzzles. Bouwkamp, a Dutch physicist, has worked on 3D tilings. MacGillavry, a Dutch crystallographer, has analyzed Escher's use of symmetries and correlated this to crystal symmetries. Coxeter, a Canadean mathematician, has worked with Escher and contributed to mathematical tiling theory, if not for the limit cycles.

2. Do you recognize a variant of the Pythagorean tree in the right one? Isn't it amazing? Pythagorean trees will be treated in the note on fractals.

ous triangles and hexagons.
To draw Pythagorean trees needed a lifetime in the 40-ies. I don't know how much time it took to draw tilings, but with the computer it is just a matter of programming, which is substantial less especially when we can build from templates.

1.1 Coding

The coding of tilings comes down to finding a basic element – a tile – and to make compositions of rotated and/or translated copies. Because these symmetry operations can be easily programmed for a computer, drawing and designing tilings can be seen as computer art nowadays.

In contrast with general POSTSCRIPT codes which are usually generated by programs just for use, my codes are concise, consistent,[3] educational, procedural, and next best to literate.

The POSTSCRIPT codes are ready for use. The METAFONT codes don't build a character, just the inherent aspects are shown, to illustrate differences with coding in POSTSCRIPT.[4] For use adaptations are needed reflecting your computing environment, be it to build a character from the code and export this etc., or to make a MetaPost code from it, and so on.

Not all codes have been included, especially the more elaborate ones, mostly of composite tilings, have been omitted. Nor are there METAFONT codes for all pictures. The reason is that I started with METAFONT but experienced later on POSTSCRIPT, and when the note grew only POSTSCRIPT codes were developed, which for these problems is sufficient and practical enough, given my situation. It is hoped that the included METAFONT codes serve their purpose.

Fractal tilings and space filling curves – well ... mathematical curves – will be treated in separate notes.

All the enclosed pictures have POSTSCRIPT codes; none has been scanned.

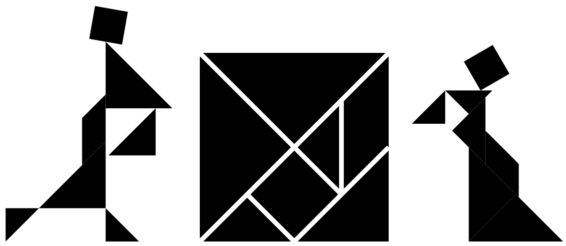
Conventions I've adopted the following conventions in coding POSTSCRIPT and hope by stating them it will ease the reading of the codes.

- Constants are at the beginning and have short names. The prefix m (minus) denotes the negative value, h denotes half the value. The postfix loc denotes a quantity to be used locally, especially in recursion. So $mhsloc$ denotes $-.5s$, to be used locally.[5]
- $/name$ exch def, lays hands on the value on the top of the stack. This is used to reuse an argument (by name), supplied on the stack (before the invoke of an operator).

Generally, intelligibility is served by appropriate spacing, and when spacing is absent either it is trivial or spacing is not relevant for understanding.
The following selection of structuring elements are reminders for those not familiar with POSTSCRIPT.

- definitions[6]
  $/name\dots$ def
  $/name\{\dots\}$ def
- structures
  gsave... grestore, the graphics state delimiters[7]
  $boolean$ $\{truepart\}$ if
  $boolean$ $\{truepart\}$ $\{falsepart\}$ ifelse
  $value$ $\{loopbody\}$ repeat
  $begin\ step\ end$ $\{loopbody\}$ for
  $array$ $\{loopbody\}$ forall[8]
  $\{loopfirstpart\ boolean$ {exit} if $loopsecondpart$ } loop

Example *(Tangram)*
Another way of looking might emerge from playing with Tangram-like puzzles, to combine the (holy) 7 pieces into free forms, stimulating phantasy.



In POSTSCRIPT the Tangram tablet can be drawn as follows.

```
%!PS-Adobe- Tangram, cgl Jan 97
%%BoundingBox: -50 -50 50 50
/s 50 def       /ms   s neg def
/hs s 2 div def /mhs hs neg def
ms ms moveto ms  s lineto
 s  s lineto  s ms lineto
closepath fill %square
s s moveto     ms ms lineto
```

3. I did not pluck from the net codes programmed by various people, with as a consequence that the programs enjoy a common approach and programming style facilitating intelligibility.
4. My METAFONT codes are essentially compatible with MetaPost codes, because I pay attention to use only the common features. For example, I don't rely on the fact that in METAFONT pixels can have more than two values.
5. METAFONT allows $-.5s$ as such, nice syntactic sugar.
6. The difference is that in the first only the result is associated and stored under the name, while the latter stores and associates all that is between the curly braces with the name.
7. There are no scope braces.
8. For all the values supplied in the array the loop is executed.

```
mhs mhs moveto  0 ms lineto s 0 lineto
1 setgray 3 setlinewidth stroke
ms s moveto hs mhs lineto hs hs lineto
2 setlinejoin stroke %showpage
```

The 2 `setlinejoin` prevents unwanted sharp corners. If you understand this code this note should be easy reading for you, and do contribute gems of your own.[9]

1.2 Audience

The aimed at audience consists of users of (La)TEX, METAFONT/Post, POSTSCRIPT, …who are familiar with programming and not afraid of coding in terms of graphical primitives. The benefit of straight POSTSCRIPT (hand)coding is conciseness, efficiency, portability and universality. The drawback is little assistance, scarce diagnostic reports, tedious proofing, and so on, while developing the codes. Maybe the included codes can function as templates.

Reading (and understanding) all is too much hoped for. Nowadays we are flooded by information, and we all seem to suffer from the desease of our times: lack of time. If only *one* example will appeal to you or spark your imagination, I'll be happy.

1.3 Why?

What has POSTSCRIPT got to do with TEX?

EP history has it that TEX and POSTSCRIPT form a real good team. POSTSCRIPT provides the graphical primitives which TEX is lacking.

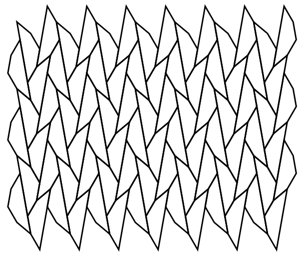
Some authors provide codes in BASIC, such as Lauwerier and Peitgen c.s. I completely agree that for those cases where images can be specified in terms of just a few graphical primitives it is wise to use omnipresent, cheap and trustworthy tools like BASIC (and Ghost/POSTSCRIPT), especially when time-invariance of the algorithms is at stake. However, with the ubiquitous TEX together with multiplatform drivers which so easily and seemlessly merge the POSTSCRIPT pictures into the dvi script, one can go a step further and use POSTSCRIPT instead of BASIC, the more so because I expect POSTSCRIPT to enjoy a significant lifetime.

The above does not hold for other proposed teams like TEX and SGML or PDF (yet) IMHO, with all respect.

1.4 Division of the plane

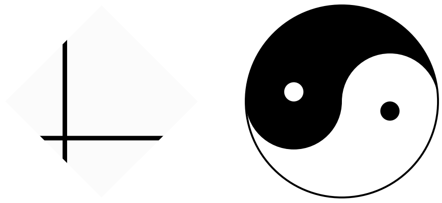
Related to tiling is the art of cutting a figure into pieces. And if we consider colors the stained-glass windows come to mind as beautiful examples.

Example *(Stained-glass (monohedral) pattern)*



In black-and-white the following are simple but famous examples.[10]

Example *(Lines and arcs)*



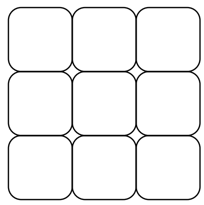
Puzzles bridge the gap between cutting a picture into pieces – the design – and tiling – the remake.

2 Squares

A square is a common tile. An ordinary pavement is built from squares. Bathroom walls are tiled with them. And so on. Empty squares with deformed sides are abundant.

It's amazing what can be achieved with just squares. Cut out the next one and tile, or better still switch on your computer and try some of your own.

Example *(Squares with rounded corners)*



The coding below illustrates how to tile in POSTSCRIPT: create a (path for a) tile, symmetrically around the origin,

9. In order to draw (free) forms it is better to name the pieces – triangle, square and diamond – and manipulate these. However, it is rather cumbersome to position them into patterns. For puzzling purposes it would be better if the X-works software provide the tangram pieces as part of their icons or allow a user to add icons. Considering the total process especially the reuse – in papers, as logos and so on – I'm happy with the investment of having coded the pieces in raw POSTSCRIPT.

10. For the codes see the Appendix.

and draw translated copies.

```
%!PS-Adobe- Tiling of rounded squares
%%BoundingBox: -75 -75 75 75
/r 10 def /a 50 def /ma a neg def
/ha a 2 div def /mha ha neg def
/tile{4{ha r sub mha moveto
        mha mha mha mha r add r arcto
        90 rotate
      }repeat
}def
ma a a{/i exch def
ma a a{/j exch def
 gsave i j translate tile stroke grestore
}for}for %showpage
```

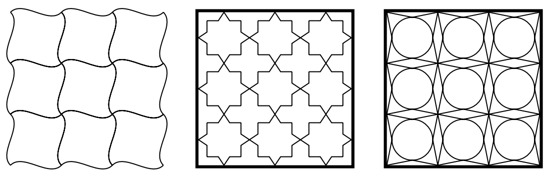
Remarks. Programming the rounded corners via an appropriate use of `arcto` is borrowed from Adobe's red book, the POSTSCRIPT reference manual. There are two parameters: the side of the square and the radius of the rounded corners. Note how the values of the hidden loop variable can be used.[11]

In METAFONT the coding of the tiling template can read as follows.

```
a=50;r=5;
path p,tile; pickup pencircle scaled 1;
p:=quartercircle scaled 2r
         shifted (.5a-r,.5a-r);
p:=p--(p rotated 90);
tile:=p--(p rotated 180)--cycle;
for i=-a step a until a:
for j=-a step a until a:
         draw tile shifted(i,j);
endfor; endfor; showit; end
```

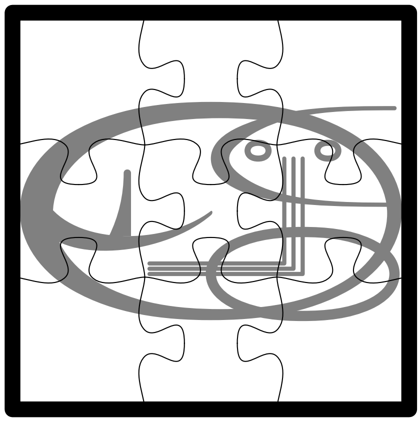
Remarks. Another possibility is to draw the tile and copy pictures. Note that only the corners are specified; the connecting straight lines are implicit.

Example *(Variant tiles)*

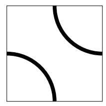


In art we have the tilings by members from 'De Style' like Mondrian and van Doesburg. But before we go over to Mondrian we will play a little longer with nearly empty tiles.

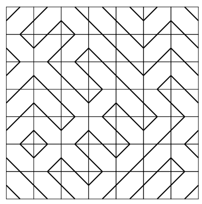
Example *(Puzzle)*
I leave it to your imagination how to draw the enclosed puzzle. It is fun to code the puzzle pieces. I did it with only two pieces, which of course are each others inverses in the sense that the sides fit. Nice exercise in using POSTSCRIPT's symmetry operators. For the background any picture would do of course.



Example *(Truchet tilings)*
Instead of deterministic tiling we can select tiles randomly from a set and tile with these. Truchet[12] considered the tile, well... together with its rotated variant, to yield a set of two.



Patterns composed of these resemble the dragon figures as mentioned in The TeXbook. In the following the tile is even further simplified by replacing the quarter circle by a straight line.



The above is obtained by the following code.

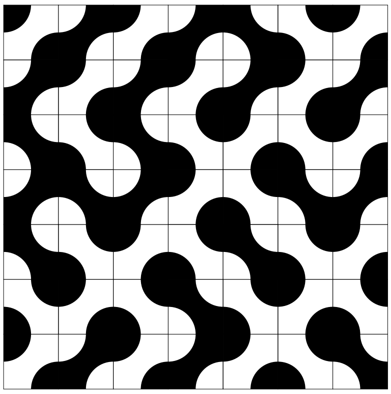
```
%!PS-Adobe- Truchet's tiling, cgl Mrt 96
%%BoundingBox: -87.5 -87.5 87.5 87.5
/a 25 def       /ma a neg def
/ha a 2 div def /mha ha neg def
/tile{rand dup 2 idiv 2 mul eq {90 rotate}if
```

11. PostScript's 'for' pops the value of the loop control variable on the stack among other things, such as excecuting the loop body. '/i exch def' uses the value on the top of the stack and stores the value-name pair in the current dictionary, ready for later use of the loop control variable value by the name i.

12. A French dominican priest of the early 18th century.

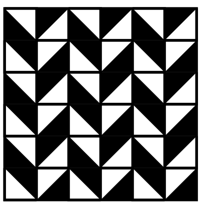
```
   mha mha moveto ha mha lineto
    ha  ha lineto mha ha lineto
   closepath .1 setlinewidth stroke
   contents
}def
/contents{0 ha moveto ha 0 lineto
          0 mha moveto mha 0 lineto
          1 setlinewidth stroke
}def
/dotiling{f ma mul a f a mul{/i exch def
          f ma mul a f a mul{/j exch def
             gsave i j translate
             tile stroke grestore
         }for}for
}def
%
/f 3 def 5 srand dotiling showpage
```

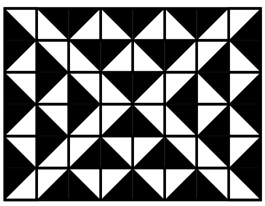
The GUTenberg (French) TUG had in 1995 a contest along with their PSTricks/POSTSCRIPT tutorial about tiling à la Truchet.[13] It turns out that for coloring 2 colors are sufficient. Have a try.[14]



A nice game. Of course the curves can be varied further and nice results will come your way.

Example *(Douat's parquets)*
A variant for parquets where a square is divided into 2 along the diagonal. By properly arranging the elements parquets are obtained. Douat[15] classified varies parquets.

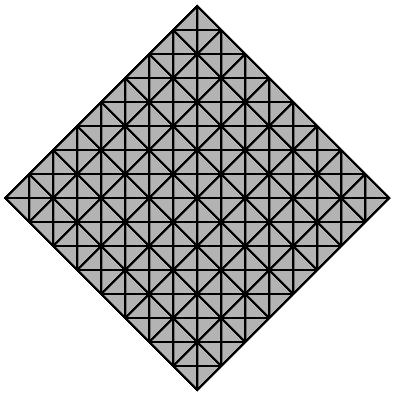




The left figure is obtained as follows.

```
%!PS-Adobe- Douat's parquets, cgl April 97
%%BoundingBox: -37.5 -37.5 37.5 37.5
/a 25 def        /ma a neg def
/ha a 2 div def  /mha ha neg def
/qa ha 2 div def /mqa qa neg def
/ll{gsave qa mqa moveto
    mqa mqa lineto
    mqa  qa lineto closepath fill
     qa mqa moveto
     qa  qa lineto
    mqa  qa lineto stroke
    grestore
}def
/lr{gsave  90 rotate ll grestore}def
/ur{gsave 180 rotate ll grestore}def
/ul{gsave -90 rotate ll grestore}def
/tile{gsave mqa  qa translate ll grestore
      gsave mqa mqa translate ur grestore
      gsave  qa mqa translate lr grestore
      gsave  qa  qa translate ul grestore
}def
ma a a{/i exch def
ma a a{/j exch def
   gsave i j translate tile grestore
}for}for %showpage
```

Example *(Lozenge by Mondrian)*



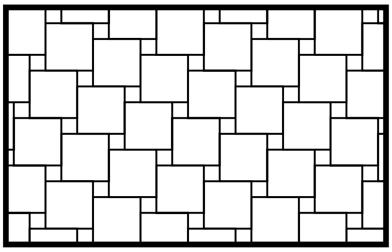
13. As communicated by Denis Roegel, see Cahiers GUTenberg 1995 for details. For solutions and the winner see the spring 1997 cahiers.
14. I solved the problem in POSTSCRIPT by looking at the colored pieces as pieces of a puzzle to be put together under the restriction of continuity of the colors along the sides. I grouped the 4 tiles into 2 sets. The tiling is obtained by taking a tile from each group on turns. Which tile to take from a group is free and therefore can be subject to throwing dice.
15. A French priest of the early 18th century. See H.A Lauwerier(1988): Symmetrie – Regelmatige structuren in de kunst. Aramith. ISBN 90 683 4032 8, NUGI 819. As usual it contains also programs in BASIC

Remark. The real thing is not so perfect, it has lines of varying width making it more interesting. For computer art it might be a challenge to imitate this. But maybe we should leave the imperfections to humans and concentrate on what the computer is good at.

The POSTSCRIPT program below illustrates another approach to coding when the pattern is so straight: (redundant and clipped) lines, no tiles. Note the use of the symmetry by rotating over 90 degrees.

```
%!PS-Adobe- Mondrian's Lozenge, cgl Jan 97
%%BoundingBox: -80 -80 80 80
/e 10 def /r e 8 mul def /mr r neg def
/frame{r 0 moveto 0  r lineto mr 0 lineto
       0 mr lineto closepath
}def
/lines{-7 1 7{e mul dup mr moveto
              r lineto
             }for stroke
}def
/diagonals{2 2 15{e mul dup r moveto
                  neg mr exch lineto
                 }for stroke
}def
gsave
frame gsave .95 setgray fill grestore clip
lines diagonals 90 rotate lines diagonals
grestore frame stroke %showpage
```

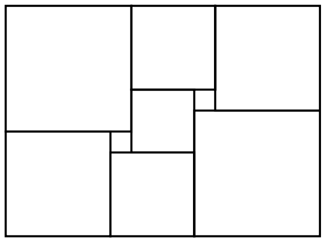
Example *(Van Doesburg inspiration)*



```
%!PS-Adobe- Van Doesburg squares, cgl Feb 97
%%BoundingBox: 25 -6.25 250 115
/s 25 def          /ms s neg def
/alfa s 3 div def  /malfa alfa neg def
/hs .5 s mul def   /mhs hs neg def
/square{hs mhs moveto hs hs lineto
    mhs hs lineto mhs mhs lineto
    closepath stroke
}def
/frame{s mhs moveto 9 s mul mhs lineto
  9 s mul 4.5 s mul lineto
  s       4.5 s mul lineto closepath
}def
gsave frame clip
11{gsave 11{square s malfa translate}repeat
   grestore alfa s translate
  }repeat grestore
frame 3 setlinewidth stroke %showpage
```

Explanation. It is all about one square with appropriately shifted copies, with the shift parameterized by α. Variant patterns can be easily obtained by changing the value of α. The small squares are spurious. In reality van Doesburg enriched patterns like these by coloring the squares.

Example *(Squaring the square)*
Is it possible to partition a square into unique squares? A simpler problem is to divide a rectangle into squares, and allow double occurrences.[16]

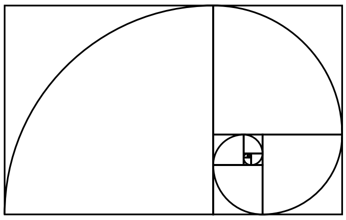


The following is a straightforward POSTSCRIPT code, with the use of (point) inversion. Interesting, and not trivial IMHO, is a to write a POSTSCRIPT operator to draw the picture automatically starting with Bouwkamp's notational array as argument. A nice application of nested `forall`-s. For the case at hand the notational array reads [[6 4 5] [3 1] [6] [5 1] [4]].

```
%!PS-Adobe- Squaring the square, cgl Mrt 97
%%BoundingBox: -75 -55 75 55
/square{%s(ide) on stack
  /s exch def
  /hs .5 s mul def /mhs hs neg def
  mhs mhs moveto  hs mhs lineto
  hs  hs  lineto  mhs hs lineto
  closepath stroke
}def
30 square
2{gsave -45 25 translate 60 square grestore
  gsave   5 35 translate 40 square grestore
  gsave  50 30 translate 50 square grestore
  -1 -1 scale
 }repeat %showpage
```

16. For an account of the squaring the square problem see for example Gardner's More Mathmatical puzzles booklet.

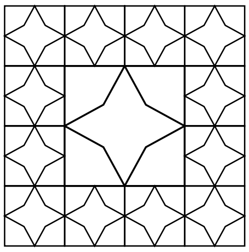
Example *(Nested rectangles and spiral of life)*



The nested rectangles have as ratio between width and height of the sides .618, that is an approximation of the golden ratio constant, commonly denoted by ϕ.[17] Nice is the (approximate) life spiral curve through the corners of the squares.

```
%!PS-Adobe- Shrinking squares, cgl Feb 97
%%BoundingBox: 0 0 200 125
/x 200 def /y .618 x mul def
/square{0 y lineto y y lineto y 0 lineto
  y 0 y 180 90 arcn y y translate
}def
12{0 0 moveto square -90 rotate
   /aux x def /x y def /y aux y sub def
  }repeat stroke %showpage
```

Example *(Icons)*
A central painting surrounded by smaller ones – in Russian: klema – illustrating episodes related to the central image.

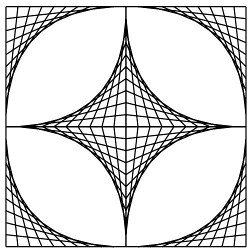


In the code below the peripheral squares are the same, but can be varied of course.

```
%!PS-Adobe Icon, cgl Feb 97
%%BoundingBox: -64 -64 64 64
/s 64 def /ms s neg def
/hs .5 s mul def /mhs hs neg def
/element{%hs on stack
    /hsloc exch def /mhsloc hsloc neg def
    frame contents
}def
/frame{
  hsloc mhsloc moveto hsloc hsloc lineto
  mhsloc hsloc lineto mhsloc mhsloc lineto
  closepath stroke
}def
/contents{hsloc 0 moveto
  4{45 rotate .5 hsloc mul 0 lineto
    45 rotate    hsloc     0 lineto}repeat
  closepath stroke
}def
/indices[-1.5 -.5 .5 1.5]def
%
hs element .75 setlinewidth
indices{/r exch def
indices{/c exch def
  c abs 1 gt r abs 1 gt or
     {gsave r hs mul c hs mul translate
            .5 hs mul element
      grestore}if
}forall}forall %showpage
```

Explanation. The tile – `element` – is parameterized over the size. The positioning is prescribed in the array `indices`. A double loop with an appropriate selection criterion whether a klema should appear. All that matters are the appropriate shifts and the coding of the element. The shifts are the values of the loop variables, apart from a scaling factor. The element is a framed star. The coding of the star has been treated in ‘Stars around I.’ It is rather straightforward, once you look upon it as a problem similar to the drawing of a polygon, namely with a broken line as side.

Example *(2D regular surface)*
This pattern is a remade of my youngest daughter’s handwork when at primary school. She did only one tile, that is a quarter of the pattern. This example illustrates the advantage of the computer. Once we have the basic element assemblages of translated or rotated copies can be done perfectly and easily.



POSTSCRIPT code by which the picture above is drawn.

```
%!PS-Adobe- Roos’ 2D Gabo, cgl Dec 96
%%BoundingBox: -100 -100 100 100
/r 100 def /n 10 def /h r n div def
4{0 1 n{/y exch h mul def
        r y moveto
```

17. As much as a constant like π or e, but lesser known. Maybe if Knuth would adopt this constant…

```
        r y sub r lineto
        y 0 moveto
        0 r y sub lineto
       }for
  90 rotate
}repeat stroke %showpage
```

Remark. Note how the spurious superellipse and cusp emerged as envelopes. A variant code more in the spirit of the earlier given template reads as follows.

```
%!PS-Adobe- Roos' 2D Gabo, cgl Dec 96
%%BoundingBox: -100 -100 100 100
/r 100 def /n 5 def /mn n neg def
/hr .5 r mul def /mhr hr neg def
/h hr n div def
/tile{2{mn 1 n{/y exch h mul def
               hr y moveto y neg hr lineto
              }for 180 rotate
       }repeat
}def
mhr r hr{/i exch def
mhr r hr{/j exch def
   gsave i j translate
      i j mul 0 lt{90 rotate}if
      tile stroke
   grestore
}for}for %showpage
```

METAFONT code. In this code use has been made of the operator point ⟨*expression*⟩ of ⟨*path*⟩.

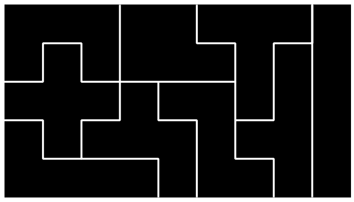
```
size=50; path p;
p=(origin..controls (0,.6) and (.4,1)..(1,1)
   ..controls (1,.4) and (.6,0)..cycle
   &unitsquare) scaled size;
draw p;
for t:=1 upto 9:
% draw point   .1t of p -- point 1+.1t of p;
  draw point 2+.1t of p -- point 3+.1t of p;
  draw point 4+.1t of p -- point 5+.1t of p;
endfor
addto currentpicture also
  currentpicture  transformed
 (identity rotated  90 shifted (2size, 0));
addto currentpicture also
  currentpicture transformed
  (identity rotated 180 shifted (2size, 0));
showit; end
```

Remark. The inner boundary is part of the path because in the real thing the inside 'eye' was also filled with lines. These splines could have been specified equally simple in POSTSCRIPT, but there is as yet not a POSTSCRIPT equivalent of METAFONT's operator point of.

2.1 Groups of squares

Combining pentominoes – 5 squares grouped together – is a teaser.[18]

Example *(Pentomino puzzle)*



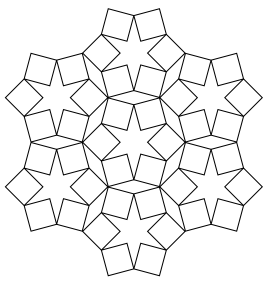
The coding is tedious but straightforward.[19]

```
%!PS-Adobe- Pentominoes, cgl Jan 97
%%BoundingBox: -180 -100 180 100
/u 20 def /mu u neg def
/t u 3 mul def /v u 5 mul def
/s u 7 mul def /n u 9 mul def
/mt u -3 mul def /mv u -5 mul def
/ms u -7 mul def /mn u -9 mul def
mn mv moveto n mv lineto n v lineto
         mn v lineto closepath fill
mn  u moveto
ms  u lineto  ms  t lineto
mv  t lineto  mv  u lineto
mt  u lineto  mt  v lineto
mt  u moveto
%similar for other boundaries
2 setlinejoin 1 setgray
2 setlinewidth stroke %showpage
```

2.2 Non-tight tilings

Non-tight tilings have been exercised by for example Kepler.

Example *(Non-tight tiling of squares)*



18. Similarly one can group hexagons and so on as pieces for a puzzle.
19. As with the tangram, when we want to draw more tessellations it is convenient to write definitions for each piece and so on.

Because of the inherent symmetry the coding does not take much lines. Just a (rotated) square as basic element with rotated and translated copies to form the pattern. The diamonds, dodecagons and stars are spurious.

```
%!PS-Adobe- Square madness, cgl Dec 96
%%BoundingBox: -150 -150 150 150
/r 50 def /s r 2.7320 div def
          /d r 2 mul s .707 mul sub def
/rotsq{gsave s 0 moveto
   4{0 s lineto 90 rotate}repeat
   closepath stroke grestore
}def
/dodeca{6{gsave r s sub 0 translate rotsq
          grestore 60 rotate}repeat
}def
gsave dodeca grestore
6{gsave 0 d translate dodeca
  grestore 60 rotate}repeat %showpage
```

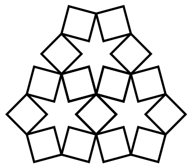
Explanation. The assembled squares yield spurious dodecagons. I chose as independent parameter for the drawing the radius r of the dodecagon. Its side equals $\frac{r}{2\sin 75}$. Note that the side of a square equals the side of the dodecagon. The radius s of the circumscribing circle of a square equals $\frac{r}{2\sqrt{2}\sin 75} \approx \frac{r}{2.7320}$. The shift d between dodecagons equals $2r - \frac{s}{\sqrt{2}}$, that is $2r$ minus half of the side, the overlap. Mandatory for the programming are: awareness of the CTM, the use of the graphics state scope delimiters gsave and grestore, and the difference between user and device space.

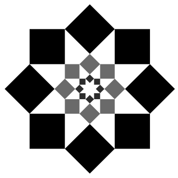
METAFONT code.

```
%Square madness, cgl Feb 97
r=50; s=r/2.7320; d=2r-.707s;
path square; picture p;
square=((s,0)--(0,s)--(-s,0)--(0,-s)--cycle)
       shifted (r-s,0);
for k=0 step 60 until 300:
    draw square rotated k; endfor
p:=currentpicture;
addto currentpicture also p shifted(0,d);
addto currentpicture also p shifted(0,-d);
addto currentpicture also
      p shifted( .866d, .5d);
addto currentpicture also
      p shifted( .866d,-.5d);
addto currentpicture also
      p shifted(-.866d,-.5d);
addto currentpicture also
      p shifted(-.866d, .5d);
showit; end
```

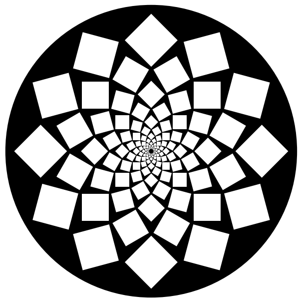
Remark. Note that the picture can be assembled by shifts only. METAFONT does not allow to rotate pictures over arbitrary angles.

The following is left as a finger exercise for the reader.[20]





Example *(Dwirling squares)*
Metamorphose or a circle limit? I associate it with the work of Schoonhoven. Well . . . what is in the name, who cares.

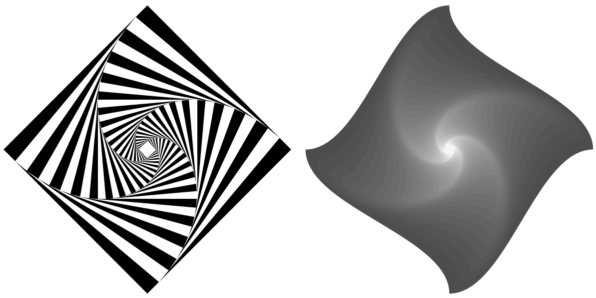


```
%!PS-Adobe- White squares, cgl Feb 97
%%BoundingBox: -65 -65 65 65
/r 50 def
/hs r 15 sin mul def /s 2 hs mul def
/rotsq{hs 1 sub 0 moveto
  4{90 rotate hs 1 sub 0 lineto}repeat
  closepath fill
}def
/ring{12{gsave r 0 translate rotsq
         grestore 30 rotate}repeat
}def
/rs r hs add 3 add def
/frame{rs 0 moveto 0 0 rs 0 360 arc
       closepath
}def
/f r hs sub r div 15 cos mul def
frame 0 setgray fill
1 setgray
12{ring f f scale 15 rotate}repeat
%showpage
```

20. By the way, it is interesting to see that the band of the right figure is enclosed from the outside by an 8-star and from the inside by a 6-star. However, the drawing can be done simpler.

Explanation. The square replaces a side of a dodecahedron, called `ring` here. Appropriately rotated and shrinked copies yields the result.

Example *(Shrinking and rotated squares)*
Lauwerier calls this a whirlpool.[21]



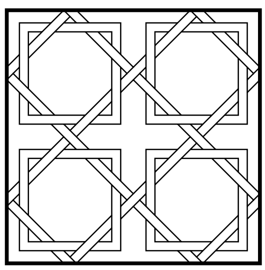
The left figure was obtained by the following code.

```
%!PS-Adobe- Shrinking squares, cgl Feb 97
%%BoundingBox: -100 -100 100 100
/r 100 def /alpha 5 def
/c 1 alpha cos alpha sin add div def
/square{r 0 moveto
  0 r lineto r neg 0 lineto 0 r neg lineto
  closepath sof}def
/sof{fill}def /flipflop true def
36{flipflop{0 setgray}{1 setgray}ifelse
   square
   /flipflop flipflop not def
   /r r c mul def
   alpha rotate}repeat %showpage
```

Explanantion. At the heart is a square parameterized over r. An appropriate use of `setgray` during rotation and shrinking yields the result.

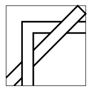
2.3 Classical tilings

Intriguing classical tilings are composed of simple geometrical patterns of which lines continue beyond the boundaries of the basic element, to form plaits.

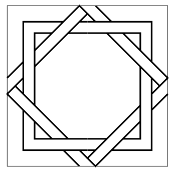


clipped pattern

How to draw these? What is to be considered as the basic element?



element



square/tile

As can be seen from the above, I choose a hook together with an overlaced strip as basic element. This element is not symmetric. A tile is composed of 4 of these elements rotated over $90k°$, for $k = 0, 1, 2, 3$, respectively. The resulting tile is (rotational, 4-valent) symmetric. Combining (translated) copies yields the pattern. The POSTSCRIPT code for the pattern above reads as follows, apart from the labeling.

```
%!PS-Adobe- Squares broidery, cgl Dec 96
%%BoundingBox: -100 -120 100 100
/s 50 def           /ms s neg def
/d 5 def /dst d 1.414 mul def
   /md d neg def
/a s d 2 mul sub def /ma a neg def
/aa a dst sub def    /maa aa neg def
/element{ms d moveto
 ma d 3 mul lineto
 maa d 3 mul dst add moveto
 md s lineto
 0 s d sub lineto
 maa d dst add lineto
 ma  d moveto
 ma d sub 0 lineto
%
 ma 0 moveto ma a lineto d -3 mul a lineto
 md a moveto 0 a lineto
 maa 0 moveto maa aa lineto
    d -3 mul dst sub aa lineto
 md dst sub aa moveto 0 aa lineto stroke
}def
%
/tile{4{element 90 rotate}repeat}def
%
gsave ms s  translate tile grestore
gsave  s s  translate tile grestore
gsave ms ms translate tile grestore
gsave  s ms translate tile grestore
```

21. He treats the general problem of rotating and shrinking of a polygon in general. H.A Lauwerier(1987): Meetkunde met de microcomputer. Epsilon 8. Jackowski – see EuroTeX 95 proceedings – used splines as sides and obtained beautiful results. The (right) figure is in his spirit. When I had nearly finished this paper another booklet by Lauwerier 'Symmetrie etc.' came my way. In that he explains Escher's approach similarly as I did. An eye-opener was his mentioning of the art of Hinterreiter, where metamorphoses are seen as a form of perspective. So just a transformation of the regular forms, that simple.

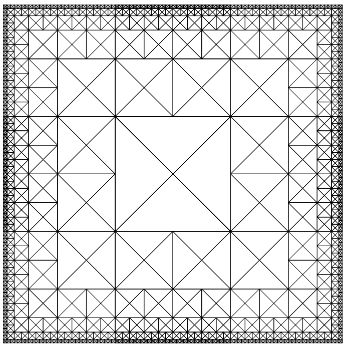
```
%frame
/ts s 2 mul def /fs ts ts add def
ts neg dup moveto fs 0 rlineto
0 fs rlineto fs neg 0 rlineto closepath
3 setlinewidth stroke %showpage
```

The coding in METAFONT goes similarly, because pictures can be rotated over $90k°$, for $k = 1, 2, \ldots$

2.4 Square limits

Michel Goossens in his 'LaTeX en PostScript – de complementariteit in de praktijk' has shown an example of this class of Eschers. Lengthy PostScript code borrowed from the net. The picture suffers from too much blackness near the boundary, IMHO, with all respect. This is an interesting aspect that we should draw thinner curves when they accumulate, because of the optical effect.

Example *(Square limits grid)*

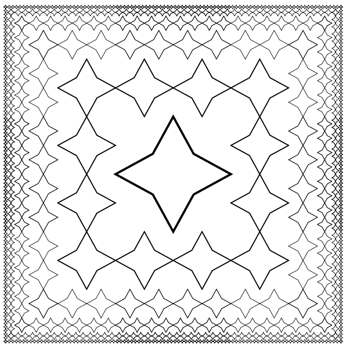


The code below is interesting because it demonstrates the use of recursion in POSTSCRIPT together with the use of the operand stack. Note that the line thickness also diminishes and that drawing is stopped overall at a certain depth, which I could not achieve via `scale`. Also interesting is the use of reflection about the line $y = x$. The code can be used as template for tiling – in the sense of Escher's limit process – with scaled (and rotated) copies of a master tile created with the origin as center. Here the (contents of the) tile – `element` – is simply a cross.[22]

```
%!PS-Adobe Square limits grid, cgl Feb 97
%%BoundingBox: -96 -96 96 96
/gs 64 def /mgs gs neg def
/square{%lw s x y   on stack
  /y exch def /lx exch def
  /hs exch 2 div def /mhs hs neg def
  /llw exch def
  gsave lx y translate llw setlinewidth
     element
  grestore
  .5 llw mul hs lx 1.5 hs mul add
                  y .5 hs mul add
  .5 llw mul hs lx 1.5 hs mul add
                  y .5 hs mul sub
  hs 1 gt {square square}
          {8{pop}repeat} ifelse
}def
/element{hs mhs moveto hs hs lineto
    mhs hs lineto mhs mhs lineto
    closepath stroke
    hs mhs moveto mhs hs lineto
    mhs mhs moveto hs hs lineto
    .25 llw mul setlinewidth stroke
}def
%
/pattern{4{/lw 1 def
  /s gs def /ms s neg def /x 0 def
  gsave
  6{gsave
    2{lw s x x square
      -1 1 scale 90 rotate
     }repeat
    grestore
    /s .5 s mul def /lw .5 lw mul def
    /x x 1.5 s mul add def
   }repeat
  grestore -90 rotate
 }repeat
}def
%
pattern %showpage
```

Escher enriched this grid with fishes for example. Maybe I'll redo that example in due time.[23]

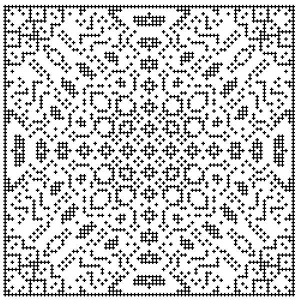
Example *(Variant with stars)*



22. The Sierpiński gasket – a fractal with limits allover – will be treated in the note on fractals.

23. On the net a larger code is available, of which the result is incorporated in Goossens' article.

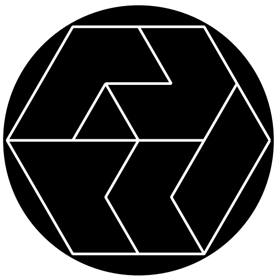
Example *(Broideries)*
Lauwerier introduced broideries. The tile is defined by a function with inherent symmetries. Lauwerier used among other things reflection symmetrie around the x-axis, y-axis, and the lines $y = \pm x$, for example as in the function $f(x, y) = (1 - x^2)(1 - y^2)$. The idea of broideries are obtained if at a point (x, y) a mark is drawn when the function value statisfies a certain criterion. In the broidery below a mark – small cross – is drawn at (x, y) if $\mathrm{int}(400\, f(x, y))$, equals a multiple of 3, for $x = -1, \ldots, -1/n, 0, 1/n, \ldots, 1$, $y = -1, \ldots, -1/n, 0, 1/n, \ldots, 1$. $n = 40$ was used.



3 Triangles

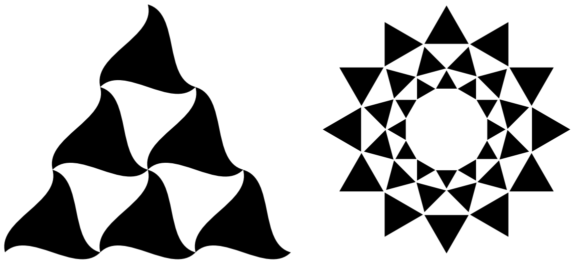
Triangles tile well, especially in space. For example they are used as elements in approximating surfaces among other things in analogy with line pieces for approximating curves in the plane. They form the sides of the Platonian solids tetraeder, octaeder, and icasohedron. I'll restrict myself to triangles in the (xy-)plane.

Example *(Puzzle of groups of triangles)*
As with tangrams we can write definitions for the pieces and so on.



Example *(Flexed triangles and coronas of triangles)*

We can modify the sides or arrange them in patterns.



The codes for these drawings read as follows.

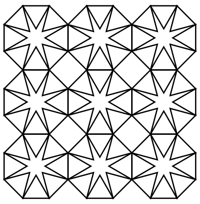
```
%!PS-Adobe- Flexed triangles, cgl Feb 97
%%BoundingBox: -15 -15 115 115
/s 50 def
/hs .5 s mul def  /mhs hs neg def
/hss 1.732 hs mul def
/qs .5 hs mul def /mqs qs neg def
/r 1.15 hs mul def
/hr .5 r mul def /mhr hr neg def
/tri{mhs mhr moveto
     3{mqs mhr qs add
       qs mhr mqs add hs mhr curveto
       120 rotate
      }repeat fill
}def
tri
gsave 2{s 0 translate tri}repeat grestore
gsave hs hss translate tri
       s   0 translate tri
grestore
s 2 hss mul translate tri %showpage
```

I did not rotate to allow more easily extension of the pattern.

```
%!PS-Adobe- Coronas of triangles, cgl Feb 97
%%BoundingBox: -72.5 -72.5 72.5 72.5
/r 50 def %BB: r(1+1.732 sin 15)
/hs r 15 sin mul def  /mhs hs neg def
/hss 1.732 hs mul def
/f r r hss add div def
/tri{hss 0 moveto
     0 hs lineto 0 mhs lineto
     closepath fill
}def
3{12{gsave r 0 translate tri grestore
     30 rotate
    }repeat f f scale 15 rotate
 }repeat %showpage
```

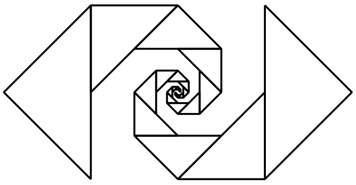
Note the use of `scale`.

Example *((Non-tight) triangles)*



For the coding the same template as for the square tiles was used. Note the spurious squares, stars and octagons.

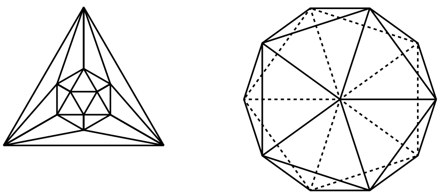
Example *(Triangular arms: east-west rencontre)*



With the following (tail) recursion code.

```
%!PS-Adobe- Triangle arms, cgl Feb 97
%%BoundingBox: 0 -50 185 50
/s 64 def /ts 2 s mul def
/tri{/locs exch def
     0 locs moveto locs 0 lineto 0 0 lineto
     closepath currentpoint stroke
     translate -45 rotate
     /locs .707 locs mul def
     locs 1 ge {locs tri} if
}def
2 setlinejoin -45 rotate
gsave s tri grestore
ts ts translate 180 rotate
s tri %showpage
```

Example *(Graph of icasohedron)*
The icasohedron has triangles as sides. The following projection shows them all. ‘Tiling’ to assist insight.

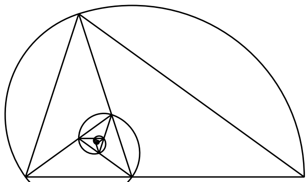


The left projection is obtained as follows.

```
%!PS-Adobe- Graph icasoheder, cgl Mrt 97
%%BoundingBox: -50 -30 50 60
/s 100 def /ss 1.732 s mul def
2 setlinejoin
/p1{.1667 s mul  -.0555 ss mul}def
/p2{.5    s mul  -.1667 ss mul}def
/p3{0            -.1111 ss mul}def
/p4{0            -.0555 ss mul}def
3{gsave
  2{p1 moveto p2 lineto p3 lineto
    p4 lineto p1 lineto p3 lineto
    -1 1 scale}repeat stroke
  grestore 120 rotate}repeat
p2 moveto
3{120 rotate p2 lineto}repeat stroke
p4 moveto
3{120 rotate p4 lineto}repeat stroke
%showpage
```

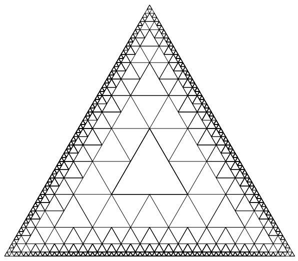
The right projection is composed of pentagons and a decahedron.

Example *(Nested triangles)*



This triangle comes back in pentagons: a side with two diagonals. The golden ratio is in there, and we can split of similar smaller triangles infinitely. The code is in the same spirit as that for rectangles.

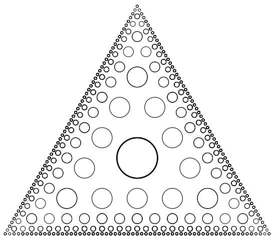
Example *(Triangular limits grid)*
In analogy with the square limits grid there is the triangular limits grid, which is related to the Sierpiǹski carpets, let us say it is an outbound variant. I did not find this one in Escher’s work. Maybe too much complexity in the corners?

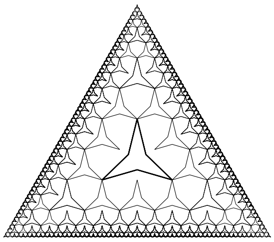


The code is in the same spirit as the code for the limit squares.[24]

24. The Sierpiński gasket – a fractal which is limiting allover – will be treated in the note on fractals.

Example *(Variants)*

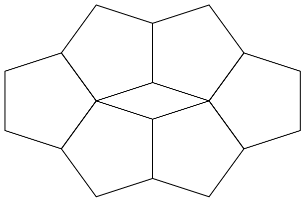




4 Pentagons

Pentagons don't tile tight. So what?

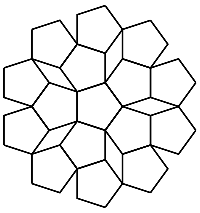
Example *((Non-tight) pentagons I; lips)*



In the coding the reflection via `scale` is interesting.

```
%!PS-Adobe- Simple Pentas, cgl Feb 97
%%BoundingBox: -175 -95 175 95
/r 50 def /mr r neg def
/rin r 36 cos mul def
/mhdx s 18 cos mul neg def
/pentagon{r 0 moveto
  5{72 rotate r 0 lineto}repeat
  stroke
}def
%
/figure{pentagon}def
2{gsave mhdx mr add 0 translate
  figure
  2{gsave 36 rotate 2 rin mul 0 translate
          figure
    grestore 1 -1 scale
   }repeat
  grestore -1 1 scale
 }repeat %showpage
```

Dürer tiled straightforwardly as follows. Only spurious diamonds show up.



Example *((Non-tight) pentagons II)*

Because of the inherent symmetry the coding does not take much lines. Just a pentagon as basic element with rotated and translated copies. The diamonds are spurious. Do you see how to code another ring, how to let the pattern grow?

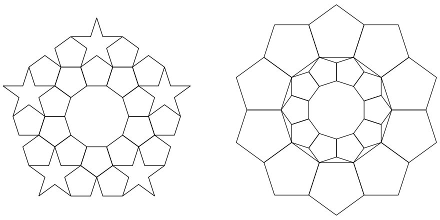
```
%!PS-Adobe- Duerer, cgl Jan 97
%%BoundingBox: -55 -65 55 65
/r 15 def /rin r 54 sin mul def
/p{gsave r 0 moveto
   5{72 rotate r 0 lineto}repeat
   stroke grestore
}def
/tr{2 rin mul 0 translate}def
p 36 rotate
5{gsave tr
  p gsave  36 rotate tr p grestore
    gsave -36 rotate tr p grestore
  grestore 72 rotate}repeat %showpage
```

METAFONT code. My earlier METAFONT code did *not* take into account the fact that the diamonds are spurious. Part of it is given below. Remarkable is that use is made of `reflectedabout` which is not needed in the simpler approach as worked out in the later POSTSCRIPT code.

```
r=15; n=5; s=2r*sind36; path p[];
diag=s*(1+2sind18);
height=s*(cosd54+cosd18);

p1=(r,0)--(r*cosd72, r*sind72);
p2=(r,0)--(r+s*cosd18,s*sind18)
    --(r+s*cosd18,s*(1+sind18))
    --(r,diag)--(r*cosd72, r*sind72);
p3=(r+s*cosd18,s*sind18)
    --(r+2s*cosd18,0)--
    (r+s*cosd18,-s*sind18);
p41=(r+2s*cosd18,0)
    --(r+2s*cosd18+s*cosd54,s*sind54)
    --(r+2s*cosd18,diag)
    --(r+s*cosd18,s*(1+sind18));
p42=p41 reflectedabout
    (origin,(r+s*cosd18,s*(1+sind18)));
for k=72step72until360:
    draw p1 rotated k;endfor
for k=0step72until360:
    draw p2 rotated k;endfor
for k=0step72until360:
    draw p3 rotated k;endfor
for k=0step72until360:
    draw p41 rotated k;draw p42 rotated k;
endfor
showit; end
```

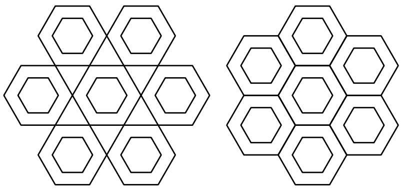
Example *(Decahedron at the heart)*



5 Hexagons

Hexagons tile tight and may yield intriguing patterns. Notice the difference in the optical effect.

Example *(Hexagon template tilings)*

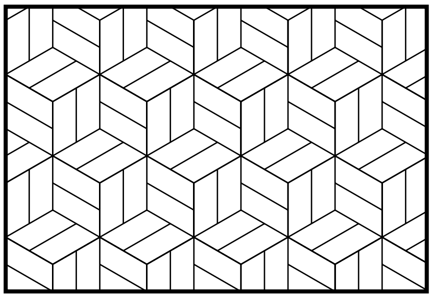


POSTSCRIPT code. Hexagon tiling templates.

```
%!PS-Adobe 6-stars, cgl Feb 97
%%BoundingBox: -75 -70 220 70
/r 25 def 2 setlinejoin
/sixtile{/rloc exch def %r on stack
  rloc 0 moveto
  6{60 rotate rloc 0 lineto}repeat
    closepath stroke
}def
%tile at corners
r sixtile .588 r mul sixtile
6{gsave 2 r mul 0 translate
  r sixtile  .588 r mul sixtile
  grestore
  60 rotate}repeat
%tile side by side
6 r mul 0 translate
r sixtile .588 r mul sixtile
30 rotate
6{gsave 1.732 r mul 0 translate 30 rotate
     r sixtile  .588 r mul sixtile
  grestore 60 rotate}repeat %showpage
```

Remark. This template will be reused when tiling hexagrams later on.

Example *(Horak's cubes)*
An optical effect is obtained, it looks like projected cubes. The minimal information – the parameter of the figure – is the radius r of the circumscribing circle of the hexagon.



POSTSCRIPT code. A hexagon is underneath, that is a cube in projection. Starting from this the code is straightforward.

```
%!PS-Adobe-Horak's tiles, cgl Jan 97
%%BoundingBox: 0 0 310 205
/r 40 def
/s r 2 div def /ms s neg def
/sst s 1.732 mul def
/ts s 2 mul def /mts ts neg def
/hs s 2 div def
/dia{0 0 moveto 0 ts lineto
   sst ms rlineto 0 mts rlineto
   0 s rmoveto 0 s lineto
}def
/tile{gsave 3{dia 120 rotate}repeat
      stroke grestore
}def
/pattern{gsave
2{gsave
  5{gsave tile sst s 3 mul translate
          tile
    grestore   sst 2 mul 0 translate
   }repeat
  grestore 0 s 6 mul translate
 }repeat grestore
}def
/a s 10.5 mul def /ma a neg def
/frame{0 0 moveto
  0 a rlineto s 15.5 mul 0 rlineto
  0 ma rlineto closepath
}def
gsave frame clip pattern grestore
frame 3 setlinewidth stroke %showpage
```

Explanation. The radius r of the hexagon is $2s$. An incomplete diamond – side of a cube – is drawn and rotated over 120° and 240° to form the basic hexagon tile, with the origin at its centre. The pattern emerges by copying hexagons translated over $k(s\sqrt{3}, 3s)$, and $k(2s\sqrt{3}, 0)$, for $k = 1, 2, \ldots$

It hints at how to draw 3D figures: draw the projection.[25]

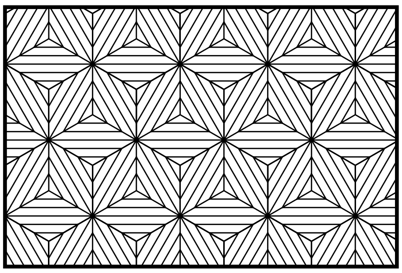
25. This does not mean that we should specify the projected figure in general. I prefer to specify the figure in 3D and use one of my

METAFONT code.

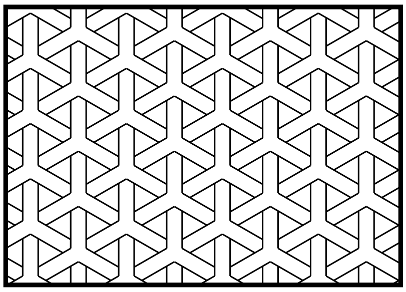
```
s=40;path p[]; %cgl Jan 97
p1=origin--(0,2s);
p2=p1 rotated-120 shifted(0,s);
p3=p1 & p2 shifted(0,s);
p4=p1 shifted(1.732s,-s);
for k=0 step-120 until-240: %draw tile
   draw p2 rotated k;
   draw p3 rotated k;
   draw p4 rotated k;endfor
for k=1upto2: %draw pattern
  addto currentpicture also currentpicture
       shifted (k*(1.732s,3s));
  addto currentpicture also currentpicture
       shifted (3.464s*k,0);
endfor
%Cut out a piece
cullit;
fill unitsquare xscaled 10size
     yscaled 7.5size shifted (2size,0);
cull currentpicture keeping (2,infinity);
%Frame the piece
pickup pencircle scaled 3;
draw unitsquare xscaled 10size
     yscaled 7.5size shifted (2size,0);
showit; end
```

Remarks. The clipping must be done differently in METAFONT, because the concept of a clipping boundary is not there. POSTSCRIPT allows any path. METAFONT requires as path a continuous line. As a consequence I have split up the basic element, although I could have traversed various parts of the element more than once.

Example *(Variant)*



Example *(Romanovsky's Chinese porcelain)*
A broidery of overlapping hexagons. The minimal information required to draw this pattern is the height s and the width w of the basic bar.



POSTSCRIPT code.

```
%!PS-Adobe-
% Romanovsky's porcelain, cgl Jan 97
%%BoundingBox: 0 0 255 180
/s 30 def /ms s neg def /ss s 1.732 mul def
          /hs s 2 div def /mhs hs neg def
          /hss hs 1.732 mul def
/w s 3 div def /wds w 1.732 div def
          /hw w 2 div def
          /hwds hw 1.732 div def
/line{hw s hwds sub moveto hw hwds lineto
}def
/tile{3{gsave line -1 1 scale line stroke
        grestore 120 rotate
       }repeat
}def
/pattern{gsave
6{gsave
  7{gsave tile
    hw hss add hs hwds add translate tile
    grestore w ss add 0 translate
   }repeat
  grestore 0 s wds add translate
 }repeat grestore
}def
/a s 6 mul def /ma a neg def
/frame{0 0 moveto     0 a  rlineto
  s 8.5 mul 0 rlineto 0 ma rlineto
  closepath
}def
gsave frame clip newpath
      mhs mhs translate pattern
grestore
frame 3 setlinewidth stroke %showpage
```

Explanation. As height I took the radius of the circumscribing hexagon. The tile is a hexagon, with 6 inner line pieces. The basic element is drawn by exploiting fully its symmetry. Once the shifts have been calculated the coding is straightforward. The pattern results after first adding a copy translated by $(s\sqrt{3}+w, s+w/\sqrt{3})/2$, and then tile

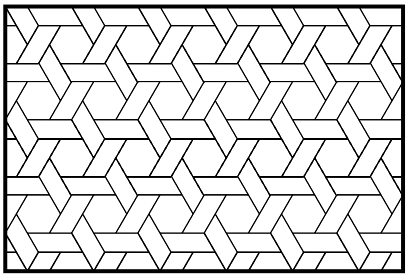
pointtopair projection operators.

this with horizontal and vertical translations $k(s\sqrt{3}+w, 0)$, and $k(0, s+w/\sqrt{3})$, for $k = 1, 2, \ldots$

METAFONT code. Note that in METAFONT we can't rotate pictures arbitrarily, but we can do with paths. The code below I wrote a year earlier than the POSTSCRIPT code, and differs a little, if not for the basic element.

```
s=30; w=.333s; path p[]; %cgl, Jan 97
p1=( .5w, .288w)--( .5w,s-.288w);
p2=(-.5w, .288w)--(-.5w,s-.288w);
for k= 0 step 120 until 240: %tile
   draw p1 rotated k; draw p2 rotated k;
endfor
addto currentpicture also currentpicture
          shifted(.866s+.5w,.5s+.2887w);
for k= 1 step 1 until 2:
addto currentpicture also currentpicture
      shifted(k*(0, s+.5774w));
addto currentpicture also currentpicture
      shifted(k*(1.732s+w,0));
endfor
cullit;%clipping boundary functionality
fill unitsquare xscaled 6.5s
       yscaled 4s shifted(.25s,0);
cull currentpicture keeping (2,infinity);
pickup pencircle scaled 3; %frame
draw unitsquare xscaled 6.5s
       yscaled 4s shifted(.25s,0);
showit; end
```

Example *(Variant hex pattern)*
And what about the following with only the side of the hexagon as parameter?



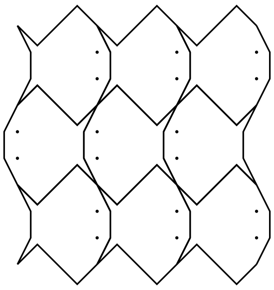
6 Escher

M C Escher, a Dutch (graphics) artist of the first half of the XX-th century, has among other things enriched classical tilings by loosening the strict geometrical tradition. He drew creatures like Buddhas, reptiles and so on, appealing to the masses, well … science biased.

Similar to the above classical plait his basic element is not symmetric.[26]

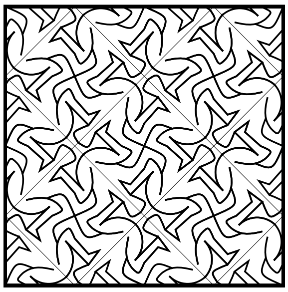
He realized that the picture on the tile is only restricted by the points where it cuts the boundary. Maybe he just thought of deforming the boundaries. Moreover, the boundaries are related by symmetry, induced by tiling.

Example *(Escher-like fishes)*
The following is borrowed from Lauwerier's earlier mentioned booklet 'Symmetrie etc.' The tiling is much in the spirit of our earlier template. The basic tile consists of 2 lines, one also translated and the other also reflected. The middle row consists of reflected tiles: fishes swimming in the other direction.



6.1 Escher's Squares

Example *(Escher's Buddhas)*



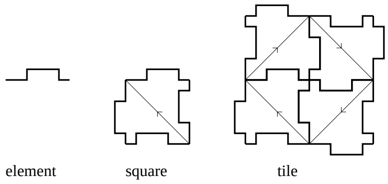
How to draw these? How to program?

Analyzing the picture reveals that the basic element is a modified side of the square. This side can be rotated and reflected to yield a square. Rotated copies of this square form a tile. Translated copies will yield the pattern.

Below the basic element has been simplified into a line with a notch, and the intermediate phases have been shown.

26. MacGillavry, a Dutch crystallographer, has studied Escher's symmetries and published on the issue.

Example *(Escher's mechanism)*



Programming the four phases

$$element \xrightarrow{R,M} square \xrightarrow{R} tile \xrightarrow{T} pattern$$

can be read from the following POSTSCRIPT code for Escher's Buddhas. (R, M, T denote $Rotation$, $Mirroring$ and $Translation$, respectively.)

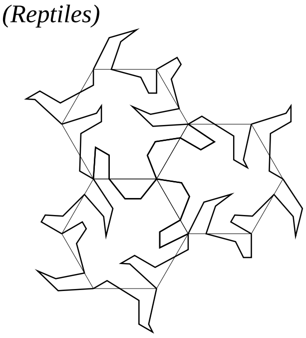
```
%!PS-Adobe- Escher Buddhas, cgl Dec 96
%%BoundingBox: -20 -45 320 140
/s 20 def /ms s neg def /ts s s add def
/pa{s s}def
/pb{.7 s mul 1.7 s mul}def
/finger{.2 s mul 1.5 s mul}def
/pc{-.4 s mul 1.45 s mul}def
/pd{.5 s mul 1.35 s mul}def
/knee{.7 s mul 1.15 s mul}def
/pf{.35 s mul s}def
/ankle{-.1 s mul .6 s mul}def
/toe{0 .3 s mul}def
/heel{-.4 s mul .6 s mul}def
/head{-.9 s mul .9 s mul}def
/center{-.25 s mul 1.8 s mul}def
/cpa{0 1.55 s mul}def
/cpb{.25 s mul 1.55 s mul}def
/cpc{-.6 s mul 1.1 s mul}def
%
/element{gsave
pa moveto pb pb finger curveto
pc moveto cpa cpb pd curveto
knee knee pf curveto %pf lineto
ankle lineto
toe lineto
heel lineto
center .75 s mul 315 225 arcn
heel moveto cpc cpc head curveto
stroke grestore}def
%
/tile{element
gsave 90 rotate 0 s -2 mul translate
  element
grestore gsave 1 -1 scale 90 rotate
  element
grestore gsave -1 1 scale
      0 s -2 mul translate
      element
grestore gsave ms s moveto s ms lineto
     .05 setlinewidth stroke grestore
}def
%
/pattern{gsave
   tile s 4 mul 0 translate
   tile 0 s 4 mul translate
   tile s -4 mul 0 translate
   tile grestore
}def
%
/as 8 s mul def /mas as neg def
/contour{ms ms moveto
   as 0 rlineto 0 as rlineto
  mas 0 rlineto closepath
}def
%
/Times-Roman findfont
             10 scalefont setfont
%Draw
element
ms ms 25 sub moveto (element)show
65 0 translate square
ms ms 20 sub moveto (square)show
75 0 translate tile
0 ms 20 sub moveto (tile)show
110 0 translate
0 ms 20 sub moveto (clipped pattern)show
contour clip pattern
contour 2 setlinewidth stroke %showpage
```

My METAFONT code is similar to the code for the reptiles given below.

6.2 Escher's hexagons

The same mechanism as demonstrated with squares was applied by Escher to a hexagonal grid.

Example *(Reptiles)*



METAFONT code.[27]

```
%Escher's Reptiles, cgl May 96
pickup pencircle scaled 1;
pair p[]; path ctoad;
s:=50;
%hexagon p1--p3--p6--p8--p10--p12--cycle
%as grid/canvas
p3=s*up;
p6=p3 rotated60;
p8=p6 rotated60;
p10=p8 rotated60;
p12=p10 rotated60;
p1:=p12 rotated60;
%pickup pencircle scaled .05;
%draw p1--p3--p6--p8--p10--p12--cycle;
p2=.3333[p3,p1];p13=.3333[p12,p1];
p4=.3333[p3,p6];p7=.3333[p8,p6];
p5=.3333[p6,p3];
p9=.3333[p10,p8];
p11=.3333[p10,p12];
%pickup pencircle scaled 5;
%for k=1 upto 13: drawdot p[k];endfor
p14=.16667[p1,p6] + .1s*down;
p141=p14 +1.1s*(-.17,.1);
p15=.25[p3,p10];
p16=.16667s*(-1,1);
p17=p16 + .3333s*up;
p18=p5 + .16673s*(.2,-1);
p19=p18 - s*(.1,.1);
p20=p19 +.3333s*down;
p21=p8 + s*(.28,0);
p22=.22[p9,p2];
p23=.3[p11,p4];
p24=p12 + 1.5s*(-.17,.1);
p25=p24-.1s*(1,.5);
p26=p25+.1s*left;
p27=p26+.3333s*up;
p28=.2[p13,p6];
pickup pencircle scaled 2;
%a complete toad, well reptile
ctoad=p1--(p1--p14--p141--p2)
        rotatedaround(p1,120)--
      p13--p28--p27--p26--p25--p24--p12--
      (p8--p21--p9)rotatedaround(p1,-120)
        shifted (0,-3s)--
      p11--p23--p22--p10--
      (p10--p22--p23--p11)
        rotatedaround(p10,120)--
      p9--p21--p8--
     (p3--p15--p16--p17--p4)
        rotatedaround(p1,-120)
        shifted (-1.5s*sqrt3,-1.5s)--
      p7--p20--p19--p18--p5--p6--
     (p6--p5--p18--p19--p20--p7)
        rotatedaround(p6,120)--
     p4--p17--p16--p15--p3--
     (p12--p24--p25--p26--p27--p28--p13)
        rotatedaround(p1,-120)--
        p2--p141--p14--p1;
draw ctoad;
draw ctoad rotatedabout (p1,120);
draw ctoad rotatedabout (p1,-120);
showit; end
```

Explanation. This code is based on points on the circumference of the hexagon. These are related by symmetry induced by the tiling as can be seen from the figure. A more extensive pattern can be obtained by adding shifts to the current picture.

POSTSCRIPT code by which the included figures are drawn.

```
%!PS-Adobe- Escher's reptiles, cgl Jan 97
%%BoundingBox: -50 -70 130 150
/r  50 def            /mr r neg def
/hr r 2 div def       /mhr hr neg def
/qr r 4 div def       /mqr qr neg def
/er r 8 div def       /mer er neg def
/hrx hr 1.732 mul def /mhrx hrx neg def
/qrx hrx 2 div def    /mqrx qrx neg def
/erx qrx 2 div def    /merx erx neg def
/delta er 2 div def
/head{hr 0 moveto
      qr mqr 1.25 mul lineto
      0  mqr 1.25 mul lineto
      mqr 0 lineto
      mqr qr 1.5 mul lineto
      merx er rlineto
      mhr 0 lineto
      stroke
}def
/side{hr 0 moveto
   mer 2 div delta sub
   mhr delta sub  rlineto
   delta mer rlineto
   merx er  rlineto
   0 hr er sub rlineto
   mhr erx add er lineto
   mhr 0 lineto stroke
}def
/tail{hr 0 moveto
   qr mhr  lineto
   mqrx mer  rlineto %end
```

27. A tedious code. An exercise in MF programming. My late POSTSCRIPT code is simpler.

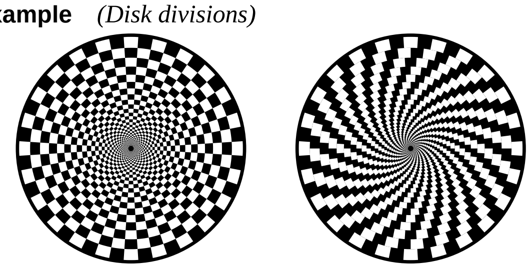
```
   qr er 1.5 mul  rlineto
   erx 0 lineto
   mqr er lineto
   mer qr rlineto
   mer 0 rlineto
   mhr 0 lineto stroke
}def
/reptile{
gsave 0 hrx translate head grestore
gsave hr qr add qrx translate
      120 rotate head grestore
gsave 0 mhrx translate
      side grestore
gsave hr qr add mqrx translate
      -120 rotate side grestore
gsave mhr mqr add qrx translate
      60 rotate tail grestore
gsave mhr mqr add mqrx translate
      -60 rotate tail grestore
}def
reptile
/grid{gsave r 0 moveto
   6{60 rotate r 0 lineto}repeat
   .05 setlinewidth stroke grestore
}def
grid
gsave r hr add hrx translate
   120 rotate grid reptile grestore
gsave 0 hrx 2 mul translate
  -120 rotate grid reptile grestore
%showpage
```

Explanation. This code is based on the symmetry of pairs of sides of the hexagon: head, side and tail. Open as yet is how to compose contours from these for coloring purposes.

6.3 Circles

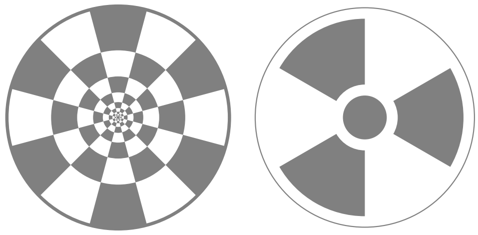
With circles we have two special locations: the centre and the boundary. I associate this with implosion and explosion. When convergence is towards the centre it is about a centre of multiplication. When convergence is outbound I'll talk about a circle limit, in pursuit of Escher.

Example *(Disk divisions)*



```
%!PS-Adobe- Circular checker board, cgl Feb 97
%%BoundingBox: -100 -103 350 103
/r 100 def
/sq{rin 3.75 cos mul rin -3.75 sin mul moveto
   0 0 rout -3.75 3.75 arc
   0 0 rin 3.75 -3.75 arcn fill
}def
/ring{/rin rout 1 1.5 3.75 sin mul sub mul def
      24{gsave sq grestore 15 rotate
        }repeat /rout rin def
}def
/frame{rf 0 moveto 0 0 rf 0 360 arc}def
%
/rout r def /rf rout 3 add def
gsave frame 0 setgray fill 1 setgray
      36{ring 7.5 rotate}repeat
grestore
250 0 translate %spirals
/rout r def /rf rout 3 add def
frame 0 setgray fill
gsave 1 setgray 36{ring 3.75 rotate}repeat
grestore %showpage
```

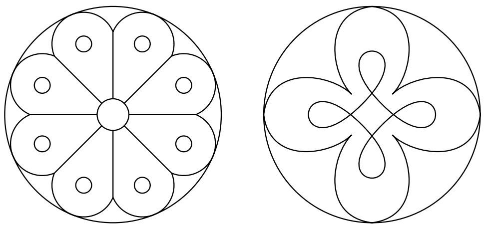
Example *(Darts board and radioactive radiation logo)*



```
%!PS-Adobe- Circular checkerboard, cgl Feb 97
%%BoundingBox: -105 -105 330 105
/r 100 def
/sector{rin 15 cos mul rin -15 sin mul moveto
   0 0 rout -15 15 arc
   0 0 rin 15 -15 arcn
   closepath fill
}def
/ring{/rin rout 1 1.5 15 sin mul sub mul def
      6{gsave sector grestore 60 rotate
       }repeat /rout rin def
}def
/frame{rf 0 moveto
   0 0 rf 0 360 arc
   closepath
}def
%
gsave
/rout r def
/rf rout 3 add def
```

```
frame 0.5 setgray fill
1 setgray
9{ring 30 rotate}repeat
grestore
%Radioactive radiation logo
225 0 translate
/rout r 10 sub def /rin rout 3 div def
/rf r def
frame .5 setgray stroke
/sector{rin 30 cos mul
        rin -30 sin mul moveto
   0 0 rout -30 30 arc
   0 0 rin 30 -30 arcn
   closepath .5 setgray fill
}def
3{gsave sector grestore 120 rotate}repeat
rin 10 sub 0 moveto
0 0 rin 10 sub 0 360 arc .5 setgray fill
%showpage
```

Example *(Windows: From a barn and from Malbork)*

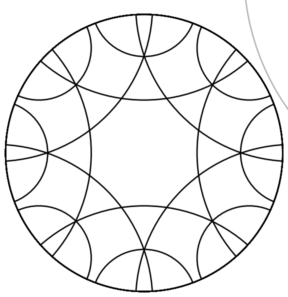


6.4 Circle limits

Escher also used hyperbolic grids: circular arcs within a circle, where the centers of the circles converge towards the boundary, and the arcs cut the boundary perpendicularly.

It did take me some time to realize what is really meant by circle limits. The basics, unblurred by Escher's approach, is really simple, and in analogy with triangular and square limits, and ... as it turned out easier to code as well. Let us first do it à la Escher followed by a straight one.

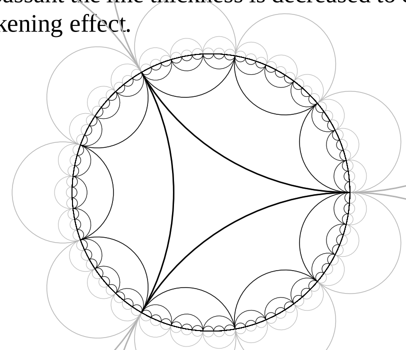
Example *(Circle limits grid à la Escher)*



```
%!PS-Adobe- Circle limits, cgl Jan 97
%%BoundingBox: -100 -100 100 100
/R 100 def /tR 2 R mul def
/a .38 R mul def
/m .5 R R a div mul a add mul def
/r m a sub def /tr 2 r mul def
/circle{%x y r on stack
   /rad exch def
   /y exch def /x exch def
   x y translate
   rad 0 moveto
   0 0 rad 0 360 arc
}def
0 0 R circle clip
%
2{8{gsave m 0 r circle stroke grestore
    45 rotate}repeat
   /r r 3 div def
   /m R R mul r r mul add sqrt def
 }repeat
0 0 R circle 2 setlinewidth stroke %showpage
```

Escher extended this also to the surface of a sphere, for example in his 'Angels and Devils.'

Example *(The straight circle limits grid)*
In analogy with squares and triangles the straight circle limits are even easier to code. The starting point is the starting number of arcs `n` as parameter. 360 divided by `n` yields the arc of the main circle cutout by the biggest circle of the limit set of circles. The centre and radius of the circle follow easily from the data of the main circle, the cutout arc, and that the circles intersect orthogonally. Rotate the template – `ring` – and loop this `levels` times. Note that en-passant the line thickness is decreased to counteract the blackening effect.



```
%!PS-Adobe- Straight circle limits, cgl Feb 97
%%BoundingBox: -100 -100 100 100
/R 100 def /levels 4 def
/circle{%x y r on stack
   /radius exch def /y exch def /x exch def
   x y translate
```

```
   radius 0 moveto 0 0 radius 0 360 arc
}def
/ring{/n exch def %n on stack
 /ang 360 n div def
 /r R .5 ang mul sin
      .5 ang mul cos div mul def
 /m R R mul r r mul add sqrt def
 gsave -.5 ang mul rotate
 n{gsave m 0 r circle stroke grestore
   ang rotate}repeat
 grestore
}def
/picture{/n 3 def /f n def /wl 1 def
levels{wl setlinewidth
       n ring /n f n mul def
              /wl .5 wl mul def
      }repeat
}def
gsave .925 setgray picture grestore
0 0 R circle clip picture 0 0 R circle
1 setlinewidth stroke %showpage
```

Circles within a circle Apollonius already considered circles which touch on 3 circles.

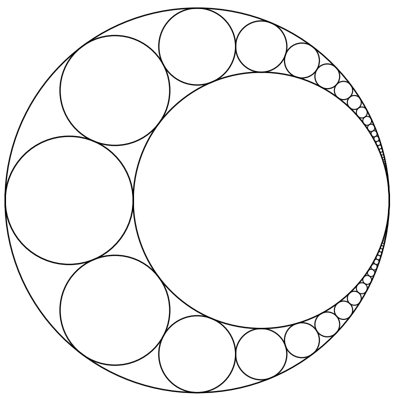
The equations for a circle, in terms of its centre and radius $\{x, y, \delta\}$, which touches on 3 other circles are

$$(x - x_i)^2 + (y - y_i)^2 = (r_i \pm \delta)^2, i = 1, 2, 3$$

with $\{x_i, y_i, r_i\}_{i=1}^3$ the data of the given circles. $\pm$ accounts for touching on the outside, respectively inside.

For the covering below the 3 quadratic equations can be simplified into 2 linear equations and a quadratic equation. These linear equations have been made explicit as function of δ for the centres of circles which touch the main circle and the circumscribing circle.

Example *(Circle covered by circles)*



Explanation. I chose as origin the midpoint of the circumscribing circle. The main circle has radius R and the circumscribing circle has radius $R + r$.

Because of this special configuration, next to my choice of origin and x-axis, the equations for the centre of the circle which touches the outer circle and the main circle, parameterized over its radius δ, simplify into

$$x = R + r - \delta \frac{2R + r}{r}$$
$$y^2 = (R + r - \delta)^2 - x^2.$$

Limit situations are $(x, y) = (R + r, 0), (-R, 0)$, for $\delta = 0, r$.

The requirement for touching the 3^{rd} circle, $\{x_i, y_i, r_i\}$, yields the equation in δ, with (x, y) given by the above formulas[28]

$$(x - x_i)^2 + (y - y_i)^2 = (r_i + \delta)^2.$$

By changing the meaning of the 3^{rd} circle repeatedly – the just determined circle becomes the next – the set of shrinking touching circles, suggestively infinitely, has been obtained.

The above considerations and ideas have been coded in POSTSCRIPT as follows.

```
%!PS-Adobe- Circle with circles, cgl Mrt 97
%%BoundingBox: -150 -150 150 150
/R 100 def /r 50 def /Rr R r add def
/xi R neg def /yi 0 def /ri r def
 Rr 0 moveto          r 0  R 0 360 arc
 Rr 0 moveto          0 0 Rr 0 360 arc
  r neg 0 moveto    xi yi  r 0 360 arc
 stroke
%tangent circle: x, y, d(elta)
/x{Rr d 2 R mul r div 1 add mul sub}def
/y{Rr d sub dup mul x dup mul sub sqrt}def
/fd{ri d add
    x xi sub dup mul
    y yi sub dup mul add sqrt sub}def
/solveit{%invariant: l fd > 0 and u fd <= 0
  /d .5 l u add mul def
  fd 0 lt{/l d def}
         {/u d def}ifelse
  u l sub eps gt{solveit}
         {/d .5 l u add mul def}ifelse
}def
%
```

28. The concise and implicit equation for δ is suited for solving by computer. The explicit solution was communicated by H.J. van de Stadt and is due to Soddy. Another approach for obtaining an explicit formula for δ in terms of a square root and the arithmetic operations is by formula manipulation programs. I'm not sure whether these programs would yield Soddy's elegant representation.

```
/eps .01 def /lw 1 def %decrease linewidth
25{/l .25 ri mul def /u ri def solveit
   x d add y     moveto x y     d 0 360 arc
   x d add y neg moveto x y neg d 0 360 arc
   /lw lw .035 sub def lw setlinewidth stroke
   /ri d def /xi x def /yi y def
}repeat %showpage
```

Remarks.

I coded a – primitive, quick and dirty – zerofinding (bisection) operator in POSTSCRIPT.[29]

Interesting is to extend tiling to the hyperbolic plane à la Coxeter, or to variate the circular arc by a pleasing line, for example anthropomorphically à la Escher.

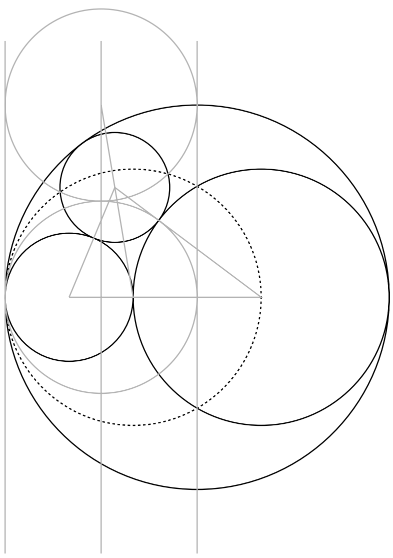
To write a program to fill up all (3-sided) cusps by circles, or all the new circles, infinitely, is interesting, and a nice, but not trivial, exercise in recursive programming.[30]

An explicit solution for the radius of the 4^{th} inscribed circle due to Soddy was communicated when the paper was in proof by H.J. van de Stadt and reads as follows.

$$\frac{1}{\delta} = \frac{1}{r_1} + \frac{1}{r_2} + \frac{1}{r_3} + 2\sqrt{\frac{1}{r_1 r_2} + \frac{1}{r_2 r_3} + \frac{1}{r_3 r_1}}$$

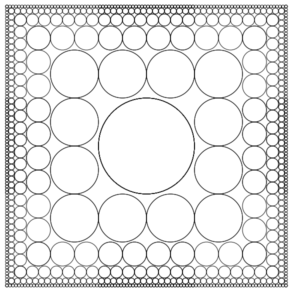
I decided to let the POSTSCRIPT code for this drawing unaltered. The use of the zero finding in POSTSCRIPT for solving an equation might be of use in other contexts.

Pondering about this problem made me realize that times have changed. We don't need anymore for these kinds of problems solutions by compass and ruler, or solutions by transformation techniques such as the inversion in a circle.[31] With our fancy computers approaches akin to this tool are emerging, and IMHO, with all respect, the above is such an approach. Apollonius revisited, aha.



Example *(The old approach)*
For the interested reader a picture which exhibits the inversion method is included on the left. The bold circles are what we are talking about. The thin circles and lines are the inverted ones, with the dashed circle the circle of inversion.

Example *(Squares limits variant with circles)*

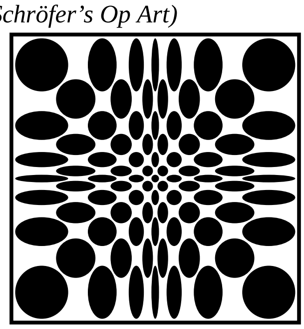


6.5 Metamorphoses

Escher played with changing the patterns while tiling. Schröfer did this with changing a circle and his result is captivating, with a nice Op Art effect. Vasarely[32] has exploited this in depth, and used colors with breathtaking results.

Hofstadter[33] calls this parquet deformations.

Example *(Schröfer's Op Art)*



Note how the circle in the center grows when moving outbound, and in general becomes an ellips. The vertical and

29. Maybe, I should recast in POSTSCRIPT in due time the famous zeroin algorithm developed at the CWI, where a combination of strategies – bisection, interpolation and extrapolation – have been implemented.

30. But, ... maybe there is another inroad when we look upon it as a fractal. H.J. van de Stadt mentions that this fractal is known as Pharaoh's Breastplate, and can be found in the book on fractals by Mandelbrot. In the note on fractals I'll come back on Soddy's formula and the fractal.

31. The approaches from the old days are discussed in for example Courant and Robbins: What is mathematics? OUP, New York, ISBN 0-19-502517-2.

32. A Hungarian computer artist.

33. Hofstadter, D R ( ): Metamagical themas – questioning for the essence of mind and pattern.

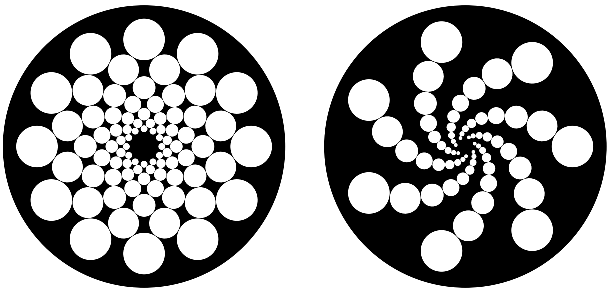
horizontal axes of the ellipses are proportional to the abscis and ordinate of its position.

```
%!PS-Adobe Schroefer's Op Art, cgl Nov 96
%%BoundingBox: -190 -190 190 190
/s 5 def
/drawgc{gsave r c translate
   r abs 5 div s add
   c abs 5 div s add scale
   0 1 moveto 0 0 1 0 360 arc fill
   grestore
}def
/schrofer{/flipflop true def
/indices[30 21 14 9 5 2 0
         -2 -5 -9 -14 -21 -30]def
indices{/r exch s mul def gsave
indices{/c exch s mul def
        flipflop{drawgc}if
        /flipflop flipflop not def
}forall grestore}forall
-38 s mul dup moveto 0 76 s mul rlineto
76 s mul 0 rlineto 0 -76 s mul rlineto
closepath 5 setlinewidth stroke
}def
schrofer %showpage
```

Remark. In the coding the use of an array for the indices, next to `forall`, and the flipflop mechanism are interesting.

My METAFONT code is similar, modulo some syntactic sugar.

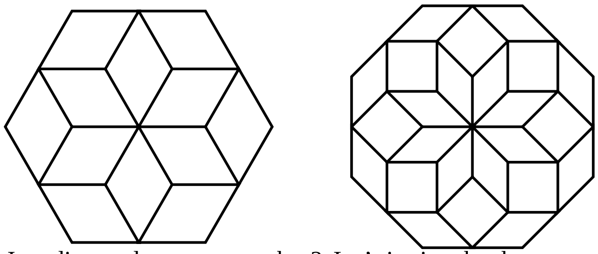
Example *(Variants)*



7 Tiling by diamonds

The following example allows a nice demonstration of how to use the ‘Turtle graphics’ idea for coding in POSTSCRIPT.

Example *(Division of hexagon and octogon)*



Just diamonds, a star or cubes? Isn't it nice that hexagons can be divided so beautifully into 12 diamonds?

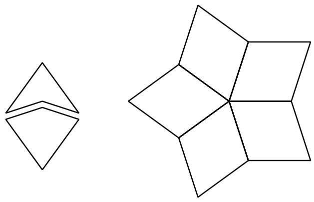
```
%!PS-Adobe- Diamonds, cgl Feb 97
%%BoundingBox: -50 -42 175 42
/turtle{%direction and stepsize on stack
  /step exch def /dir exch def
  currentpoint translate
  dir rotate step 0 lineto
}def
/r 25 def 2 setlinejoin
6{r 0 moveto -60 r turtle
             120 r turtle
              60 r turtle
             120 r turtle
             -60 r turtle
            -120 0 turtle
 }repeat stroke
%
125 0 translate /r .75 r mul def
8{r 0 moveto 45 r turtle
            -90 r turtle
            135 r turtle
             45 r turtle
            135 r turtle
            -90 r turtle
             45 r turtle
           -180 0 turtle
 }repeat stroke %showpage
```

Explanation. `turtle` draws a line, parameterized over size and direction.

Note the use of a zero step at the end, next to the direction.[34]

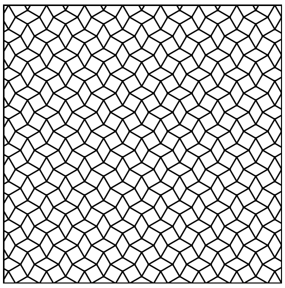
Example *(Penrose's kites and darts)*
Related to dividing hexagons and octogons into diamonds is Penrose's diamond, which he divided into 2 to cover the plain by its parts in a nonperiodic way.

34. Maybe I should have done this a little differently.



Remark. It's interesting to write a program which generates automatically tilings by these kites and darts. That is to teach a computer how to puzzle.

Example *(Rhombus as basic element)*
I chose a diamond as basic element. Of course one can also code the wiggy lines.



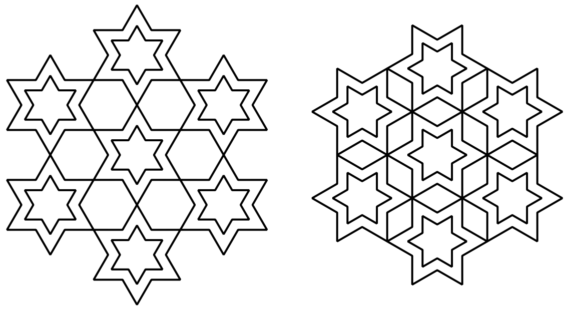
The above is obtained as follows

```
%!PS Tiling fourthree, cgl 96
%%BoundingBox: -100 -100 100 100
/a 10 def /ha a .5 mul def
/tile {% rhombus + 90 rotated rhombus
 ha 3 sqrt mul 0 moveto
 0 ha lineto  ha 3 sqrt mul neg 0 lineto
 0 ha neg lineto closepath
 ha 1 3 sqrt add mul ha 3 sqrt mul lineto
 ha 2 3 sqrt add mul 0 lineto
 ha 1 3 sqrt add mul ha 3 sqrt mul neg lineto
 closepath
}def
/tena a 10 mul def
/frame {tena neg tena moveto
 tena 2 mul 0 rlineto 0 tena -2 mul rlineto
 tena -2 mul 0 rlineto closepath}def
/dotiling{a -11 mul tena neg translate
 9{gsave
  11{tile a 1 3 sqrt add mul 0 translate
    }repeat stroke
  grestore
  gsave ha 1 3 sqrt add mul dup translate
  11{tile a 1 3 sqrt add mul 0 translate
     }repeat stroke grestore
  0 a 1 3 sqrt add mul translate
 }repeat
} def
frame clip dotiling %showpage
```

My METAFONT code reads similar.

8 Tiling by stars

The following – non-tight 'hexagon' tilings – reminds me of tilings by the Moors.

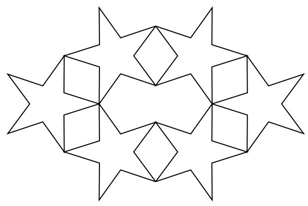


```
%!PS-Adobe 6-stars, cgl Feb 97
%%BoundingBox: -70 -70 70 70
/r 25 def 2 setlinejoin
/six{%r on stack
  /rloc exch def
  rloc 0 moveto
  6{30 rotate .577 rloc mul 0 lineto
    30 rotate  rloc     0 lineto}repeat
    closepath stroke
}def
30 rotate
r six .588 r mul six
6{gsave 2 r mul 0 translate
  r six .5 setlinewidth .588 r mul six
  grestore
  60 rotate}repeat %showpage
```

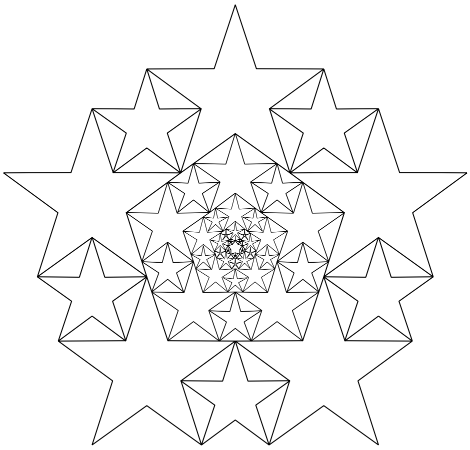
Looking at the left picture I found it hard to recognize that it was just a tiling of 6-stars. In reality the tessallations also exhibit 'hidden' lines.

Example *(Dual of composition of pentagons I)*
Stars and the enveloping polygons are related.

As exercise for the reader the following dual of the composition of pentagons I.



Example *(Composite of pentagrams)*



This code arose after Jackowski's visit to NTG's fall 1996 meeting, where he lectured and dealt with stars among other things.

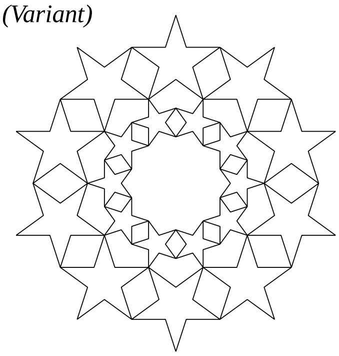
```
%!PS-Adobe- Tiling pentagrams, cgl Nov 96
%%BoundingBox: -100 -100 100 100
/star{%n r on stack, n>=5
 /rstar exch def /nstar exch def
 /alfahstar 180 nstar div def
 /rinstar rstar 2 div alfahstar cos
    mul 1.5 sub def
 0 rstar moveto
 nstar{alfahstar rotate 0 rinstar lineto
       alfahstar rotate 0 rstar lineto
      }repeat closepath stroke
}def
%
/n 5 def /r 5 def        %(r,n)-gon
/alfa 360 n div def
/lw .1 def 2 setlinejoin
%loop over the 'rings'
4{lw setlinewidth
  /rh r 2 div       def %r half
  /rin r 36 cos mul def %r inside
  %big stars
  /c rin 36 cos mul 2 mul def
  n{gsave 0 c translate n rin  star
    grestore alfa rotate
    }repeat
  %'half-sized' stars
  gsave 36 rotate
  /c rin rh add def
  n{gsave 0 c translate 36 rotate
    n rh star
    grestore alfa rotate
   }repeat
  grestore
  /r 36 cos dup 2 mul 1 add mul r mul def
  /lw lw .2 add def
}repeat %showpage
```

Explanation. The figure is composed of rings of stars, with in each ring stars of 2 sizes. The bigger stars circumscribe a spurious pentagon. The radius r of (the circumscribing circle of) this pentagon is taken as the independent parameter of the drawing. Dependent quantities are the following.

Radius inner circle: $r_{inside} = r\cos 36$
Side pentagon: $2r\sin 36$
Bigger star
radius: $r\cos 36$
distance from center: $2r\cos^2 36$
Smaller star
radius: $.5r$
distance from center: $r_{inside} + .5r$
The radius of the next ring: $r\cos 36\,(2\cos 36 + 1)$
BoundingBox:[35] $r\left(\cos 36\,(2\cos 36 + 1)\right)^n \approx 2.12^n r.$

Drawing a star has been explained in 'Stars around I.'[36] Note the use of 2 `setlinejoin` to circumvent spurious sharp corners.[37]

Example *(Variant)*



9 Acknowledgements

Thank you Bogusław Jackowski for your suggestions, inspiring examples, and help. Thank you Sasha Berdnikov for providing me with examples of tilings. Thank you Erik Frambach for comments and discussions. Peter van Summeren pointed me to some literature items. Denis Roegel informed me about Truchet's tiling and the GUTenberg

35. Radius of circumscribing pentagon, with n the number of rings.

36. See MAPS 97.1. The idea is that the star can be seen as a pentagon with a broken line as side. The coordinates of the junction of the 2 line elements of the broken line follow from a backside of the envelope calculation.

37. Courtesy Marcel Tünnissen, who also played with the figure and applied technology from 3D – extending the sides of polyhedra to yield starred figures – to 2D.

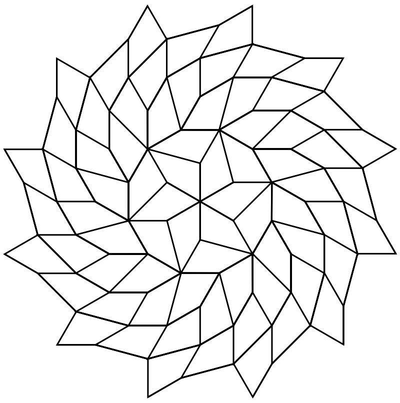
contest on the issue, next to providing me with a copy of a GUTenberg article about the winnars of the contest. H.J. van der Stadt communicated Soddy's formula.

Many examples have been borrowed from the earlier mentioned book by Grünbaum and Shephard.

As usual Jos Winnink proofed the paper and lent a helping hand in procrusting the article into MAPS outfit. Thank you all.

10 What more?

The tilings, as overwhelming they might seem, are just the top of the iceberg. What about tiling on curved surfaces, for example on a sphere, as well as general tiling in 3D? Maybe, I will come back on the issue when computers have real holographic 3D viewing possibilities.



11 Conclusions

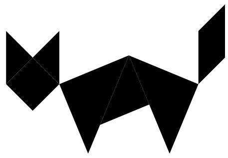
Just the use of a few graphical primitives in POSTSCRIPT and awareness of the user versus device space can yield interesting pictures via little POSTSCRIPT code. Sometimes it is handier and more elegant to use METAFONT as a declarative language. On the other hand METAFONT's pictures can't be rotated over arbitrary angles.

Differences in the concept of the path data structure in POSTSCRIPT and METAFONT entails different programs. The most apparent differences are in tiling and clipping. I would welcome in POSTSCRIPT an operator similar to METAFONT's `point` ⟨*expression*⟩ `of` ⟨*path*⟩.

Happily the same splines are used underneath facilitating METAFONT to calculate the splines from a declarative specification ready for use in POSTSCRIPT.

I never realized that tiling, especially when done by computer, can be so amuzing. So far I've no access to color printers, alas. Undoubtedly a mer a boire awaits me there.

I hope that the given codes will contribute to the literature of METAFONT and POSTSCRIPT codes. I welcome comments.



12 Appendix: Postponed codes

12.1 Mondrian's Two lines

In the original Mondrian the two lines did not divide the sides by the golden ratio. An oversight by Mondrian?

In principle a very simple code, but becomes interesting with proper coloring of the background and with lines of non-neglible thickness.

```
%!PS-Adobe- Mondrian's Two lines, cgl 96
%%BoundingBox: -50 0 50 100
/D 50 def
/R D 1.4142 mul def %side
/gr .618 R mul def
/delta 10 def /mdelta delta neg def
45 rotate
0 0 moveto R 0 lineto
R R lineto 0 R lineto
closepath
gsave .985 setgray fill grestore
clip newpath
%gr 0 moveto 0 gr lineto
 gr delta add mdelta  moveto
 mdelta  gr delta add lineto
%gr R moveto 0 R gr sub lineto
 gr delta add R delta add moveto
 mdelta  R gr sub delta sub lineto
R 30 div setlinewidth stroke %showpage
```

Explanation. I chose for a clipping boundary and to let the thick lines extend to be cut appropriately.

12.2 Yin-Yang

An exercise in using (circular) arcs.

METAFONT code.

```
%Yin-Yang, cgl Jan 97
size=100;
fill ((halfcircle&
     halfcircle rotated 180 scaled.5
                      shifted(-.25,0)&
     reverse (halfcircle scaled.5
                      shifted(.25,0))&
     cycle)scaled size);
draw fullcircle scaled size;
fill fullcircle scaled .125size
        shifted ( .25size,-.0625size);
```

```
unfill fullcircle scaled .125size
        shifted (-.25size, .0625size);
showit; end
```

Explanation. It is all about drawing, casu quo (un)filling, of (arcs of) circles, and to scale and reposition these appropriately. In plain METAFONT we have half/fullcircle which have to be scaled, rotated, and positioned. In POSTSCRIPT a circle macro is easily written and used similarly. Interesting is the use of reverse in METAFONT in order to create closed contours, which in POSTSCRIPT is done implicitely via `arcn`, that is the arc in negative direction.

A similar POSTSCRIPT code reads as follows.

```
%!PS-Adobe- Yin-Yang, cgl 96
%%BoundingBox: -50 -50 50 50
/R 50 def       /mR R neg def
/r R 8 div def  /mr r neg def
/hR R 2 div def /mhR hR neg def
/circle{%center on stack
 translate
 r 0 moveto 0 0 r 0 360 arc
}def
R 0 moveto   0 0  R   0 180 arc
           mhR 0 hR 180 360 arc
            hR 0 hR 180   0 arcn
fill
gsave mhR  r circle
      1 setgray fill grestore
gsave  hR mr circle
      0 setgray fill grestore
R 0 moveto 0 0 R 0 360 arc stroke %showpage
```

The following POSTSCRIPT code was output by MetaPost.[38] Some $\approx$ 50 lines of `curveto`, `fill` and `stroke`. Much less readable than my straight (handcoded) PostScript, moreover, ‘structure’ has disappeared.

```
%!PS-Adobe- Yin-Yang, MetaPost (JJW)
%%BoundingBox: -51 -51 51 51
newpath 50 0 moveto
50 13.26 44.73 25.98 35.35 35.35 curveto
25.98 44.73 13.26 50 0 50 curveto
-13.26 50 -25.98 44.73 -35.35 35.35 curveto
-44.73 25.98 -50 13.26 -50 0 curveto
-50 -6.63 -47.37 -12.99 -42.68 -17.68 curveto
-37.99 -22.37 -31.63 -25 -25 -25 curveto
-18.37 -25 -12.01 -22.37 -7.32 -17.68 curveto
-2.63 -12.99 0 -6.63 0 0 curveto
0 6.63 2.63 12.99 7.32 17.68 curveto
12.01 22.37 18.37 25 25 25 curveto
31.63 25 37.99 22.37 42.68 17.68 curveto
47.37 12.99 50 6.6350 0 curveto
closepath fill
 0 2 dtransform truncate idtransform
      setlinewidth pop [] 0 setdash
 1 setlinejoin 10 setmiterlimit
newpath 50 0 moveto
50 13.26 44.73 25.98 35.35 35.35 curveto
25.98 44.73 13.26 50 0 50 curveto
-13.26 50 -25.98 44.73 -35.35 35.35 curveto
-44.73 25.98 -50 13.26 -50 0 curveto
-50 -13.26 -44.73 -25.98 -35.35 -35.35 curveto
-25.98 -44.73 -13.26 -50 0 -50 curveto
13.26 -50 25.98 -44.73 35.35 -35.35 curveto
44.73 -25.98 50 -13.26 50 0 curveto
closepath stroke
newpath 31.25 -6.25 moveto
31.25 -4.59 30.59 -3.00 29.42 -1.83 curveto
28.25 -0.66 26.66 0 25 0 curveto
23.34 0 21.75 -0.66 20.58 -1.83 curveto
19.41 -3.00 18.75 -4.59 18.75 -6.25 curveto
18.75 -7.91 19.41 -9.50 20.58 -10.67 curveto
21.75 -11.84 23.34 -12.5 25 -12.5 curveto
26.66 -12.5 28.25 -11.84 29.42 -10.67 curveto
30.59 -9.50 31.25 -7.91 31.25 -6.25 curveto
closepath fill 1 setgray
newpath -18.75 6.25 moveto
-18.75 7.91 -19.41 9.50 -20.58 10.67 curveto
-21.75 11.84 -23.34 12.5 -25 12.5 curveto
-26.66 12.5 -28.25 11.84 -29.42 10.67 curveto
-30.59 9.50 -31.25 7.91 -31.25 6.25 curveto
-31.25 4.59 -30.59 3.00 -29.42 1.83 curveto
-28.25 0.66 -26.66 0 -25 0 curveto
-23.34 0 -21.75 0.66 -20.58 1.83 curveto
-19.41 3.00 -18.75 4.59 -18.75 6.25 curveto
closepath fill showpage
```

Remark. Note the redundancies, for example `newpath`.

12.3 Escher’s sun and moon

I ‘scanned’ the picture and created appropriate paths in METAFONT by incorporating the points, specifying directions and using ordinary and splice joins. When finished METAFONT was ordered to write the curves – data for the spline pieces suitable for POSTSCRIPT – to the log file. This file was edited into a straight PostScript program. My poor man’s MFTOEPS, which is conceptually simple and does not require knowledge of clever tools.

The essential issues of the program are listed below.

```
path p[]; s=30;
p11=origin..(-.7s,.5s)..(-2s,0)..
    (-2.6s,.3s)..(-3.5s,0);
p12=point 4 of p11..
```

38. Courtesy Jos Winnink, who adapted my METAFONT program for the purpose.

```
    {(1,3)}(-3.2s,s){(3,1)}..(-2.2s,s)&
    (-2.2s,1s){(-1,1)}..(-2.9s,1.9s)&
    (-2.9s,1.9s){(1,1)}..{right}(-1.5s,2.7s);
p13=point 6 of p12{(-1,-1)}..(-2s,2s)&
    (-2s,2s){(5,1)}..(-.6s,2s)&
    (-.6s,2s)..(-1.4s,1.4s)---
    (-1.6s,.9s)..(-s,.8s)..(.2s,1.2s)&
    (.2s,1.2s)..point 0 of p11;
p1= reverse(p11..p12..p13..cycle);
fill p1;
%write to log file suitable for PS
"Bird1";
for k=0 upto 24:
  show postcontrol k of p1;
  show precontrol k+1 of p1;
  show point k+1 of p1; "curveto";
endfor
%similar for other two (dark) birds
"spurious bird";
p4=point 0 of p33..(-2.2s,-2s)&
    (-2.2s,-2s){left}..(-3s,-2.1s)&
    (-3s,-2.1s)..(-2.5s,-.7s)&
    (-2.5s,-.7s)..point 4 of p11;
draw p4;
%write to log file suitable for PS
%similar for Bird 5 and 6
showit; end
```

Note. "⟨*string*⟩" writes the ⟨*string*⟩ to the log file.
The PostScript program is a concatenation of curveto-s.

```
%!PS-Adobe Escher's zon en maan; cgl  Jan 97
%%BoundingBox: -100 -100 100 100
%Bird1
 0 0 moveto
 2 12 4 24 6 36 curveto
 -4.34 28.13 -17.00 23.91 -30 24 curveto
%etc.
```

I guess the Sun and Moon picture could have been drawn easily by WYSIWYG graphics X-works software with a request for POSTSCRIPT output?

12.4 The squares patterns

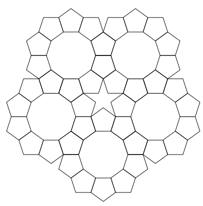
Without explanation the codes for the simple squares patterns exercises are included. I hope it is clear enough. For the accurate calculation of the BB a little knowledge of goniometry is needed.

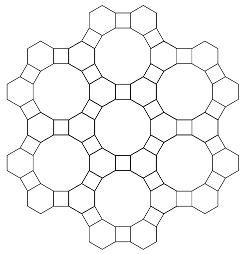
```
%!PS-Adobe- Kepler's squares, cgl Dec 96
%%BoundingBox: -50 -37.5 50 50
/r 25 def %r dodecahedron
/s r 2.8284 75 sin mul div def
/d r 2 mul s .707 mul sub def
/rotsq{gsave s 0 moveto
          4{0 s lineto 90 rotate}repeat
          closepath stroke
       grestore
}def
/dodeca{6{gsave r s sub 0 translate
             rotsq
          grestore 60 rotate
         }repeat
}def
/rstar s s mul r s sub dup mul add sqrt def
3{gsave 0 rstar translate dodeca grestore
  120 rotate
 }repeat %showpage
```

Interesting is the use of scaling in the code below.

```
%!PS-Adobe- More Square madness, cgl Feb 97
%%BoundingBox: -80 -80 80 80
/r 50 def
/s   .5858 r mul def  /ms     s neg def
/sds .707  s mul def  /msds sds neg def
/element{gsave r 0 translate
            sds 0 moveto 0 sds lineto
           msds 0 lineto 0 msds lineto
           closepath fill
        grestore
}def
/f r sds sub r sds add div def
/gl 1 def
%
3{8{element 45 rotate}repeat
  f f scale /gl f gl mul def
  gl setgray
 }repeat %showpage
```

My case rest.





By the way, the right figure is classified by {4, 6, 12}, a nice layout for a herb garden. Have fun, and all the best.