MetaPost NEWS

Hans Hagen

Mikael Sundqvist

BachoTEX 2024

METAPOST NEWS

Playing with MetaPost/MetaFun is ... fun.

We will here present some new additions (and some ongoing work).

You can read more in the articles or in the documentation

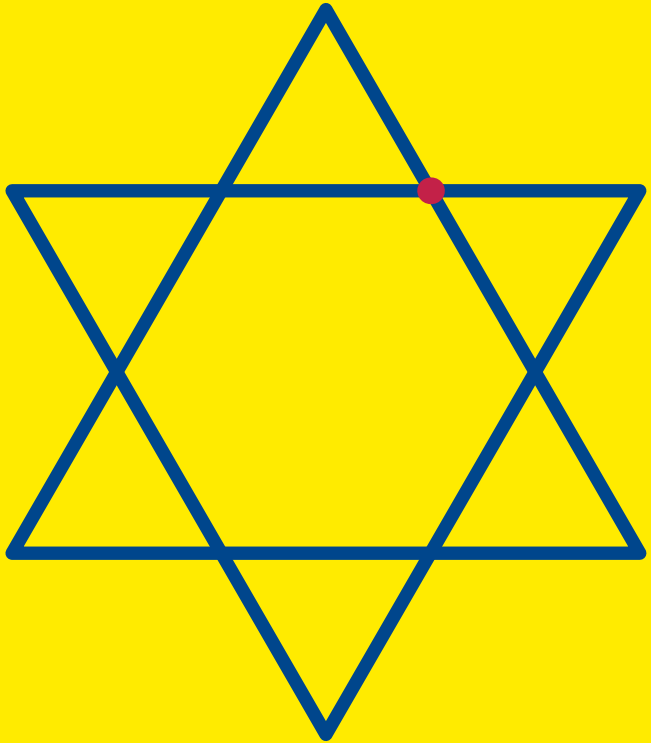
```
./tex/texmf-context/doc/context/documents/
 /general/manuals/luametafun.pdf
```

that is updated as work goes on.

FIND ONE INTERSECTIONPOINTS

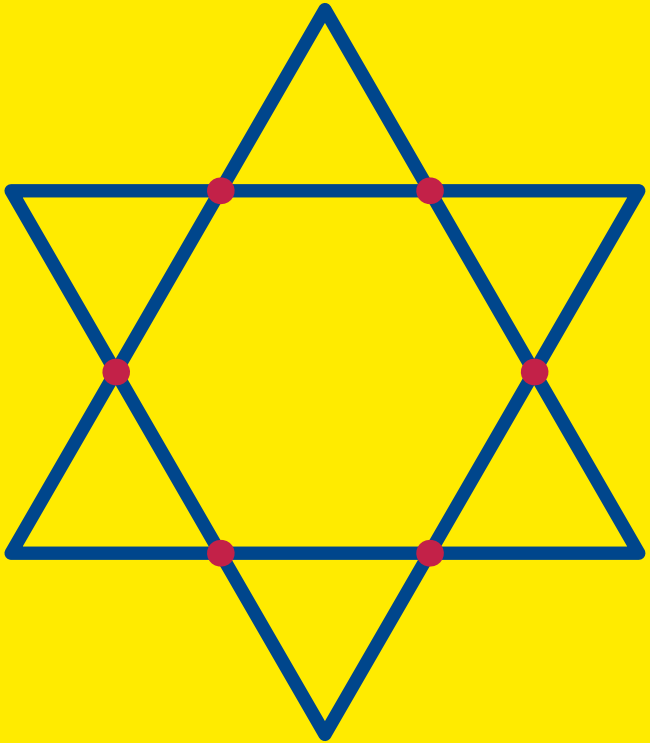
```
path p[] ;
p[1] := fulltriangle
        rotated 30
        xsized RightMarginWidth ;
p[2] := p[1] rotated 180 ;

pickup pencircle scaled 5 ;
draw p[1] ;
draw p[2] ;

pair ip ;
ip := p[1] intersectionpoint p[2] ;
drawdot ip
        withpen pencircle scaled 10
        withcolor "MyColors:3" ;
```

The `intersectionpoint` finds *one* intersectionpoint. Which one?

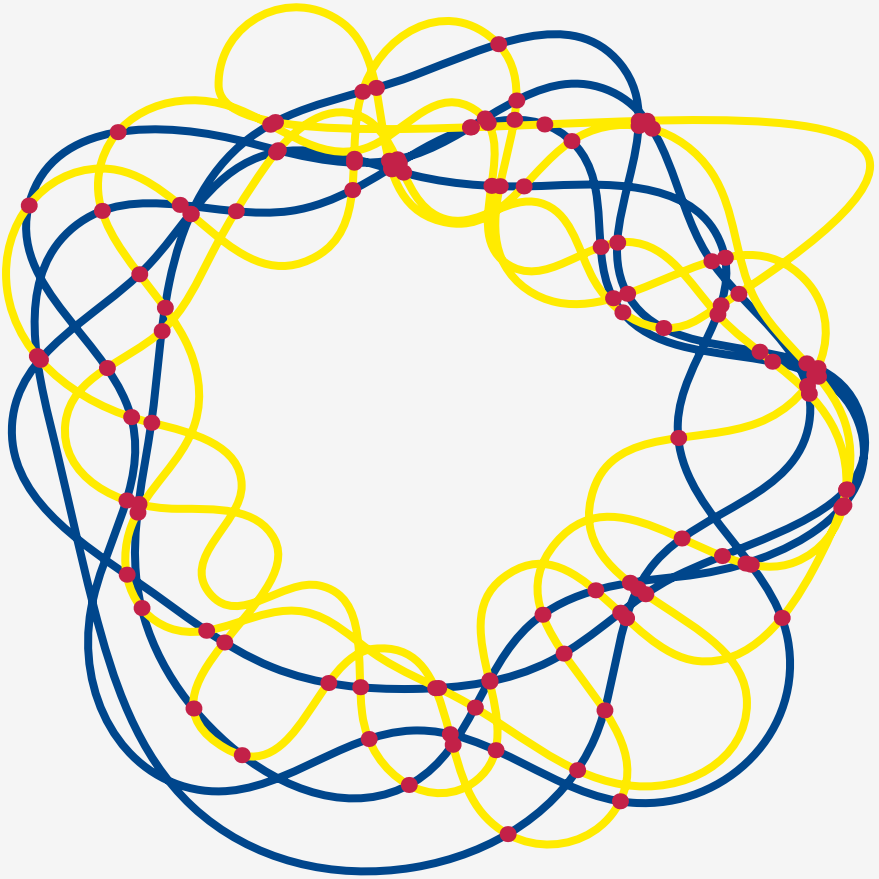


FIND ALL INTERSECTIONPOINTS

```
path p[] ;
p[1] := fulltriangle
        rotated 30
        xsized RightMarginWidth ;
p[2] := p[1] rotated 180 ;

pickup pencircle scaled 5 ;
draw p[1] ;
draw p[2] ;

path ip ;
ip := p[1] firstintersectionpath p[2] ;
drawpoints ip
           withpen pencircle scaled 10
           withcolor "MyColors:3" ;
```



EVEN MORE POINTS

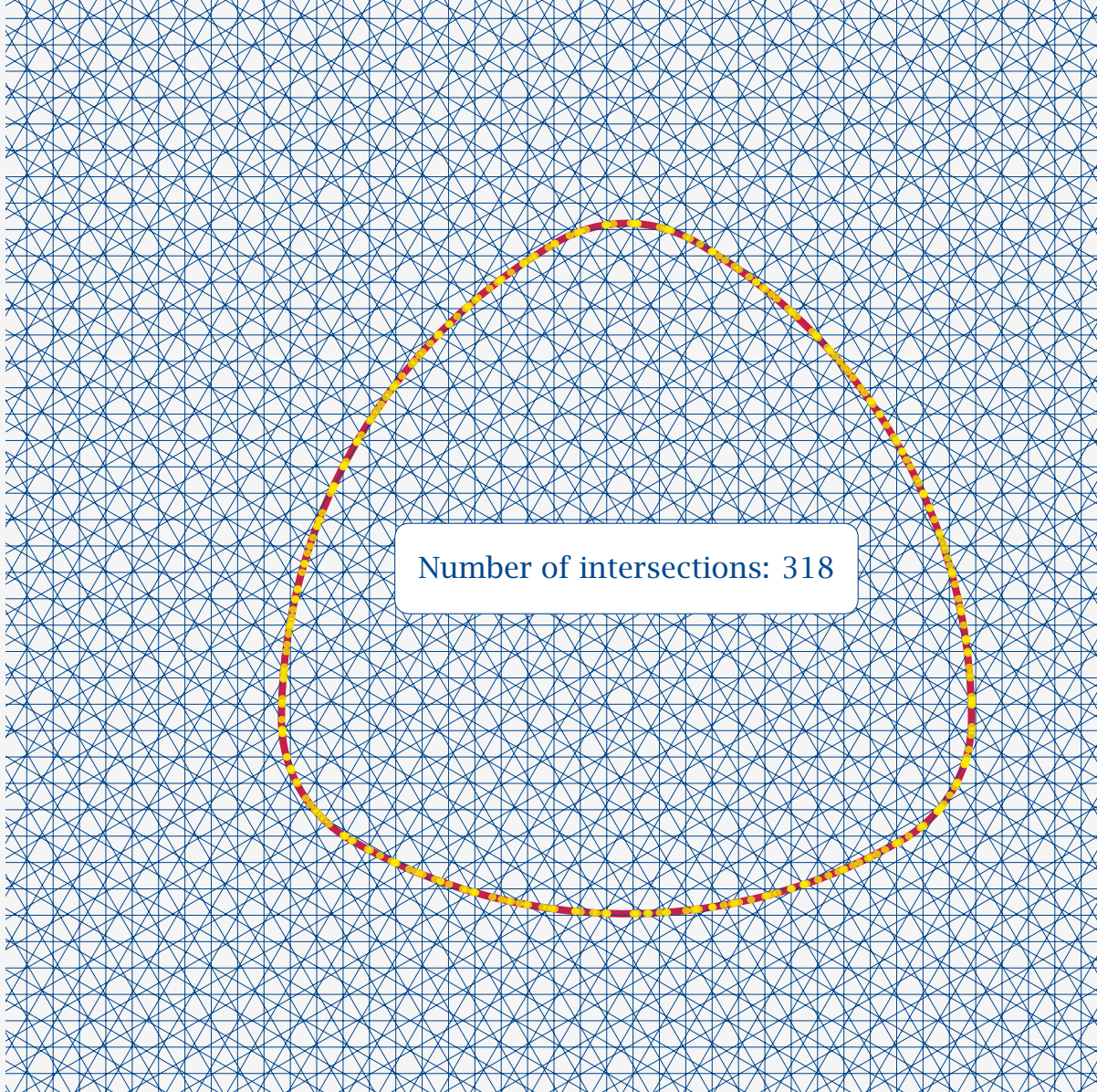


LENGTH

Hugo Steinhaus[1] longimeter

Diameter in the example: $d = 100\,\text{mm}$

Exact length: $\pi d \approx 314\,\text{mm}$



[1] 1887–1972, Polish mathematician

INTERSECTION MACROS

`intersectiontimeslist` This is the basic one. It gives a path with the times of intersections (for the first path in first coordinate, the second in the second). If the paths do not intersect, it returns $(-1,-1)$.

`sortedintersectiontimes` As above, but sorts the intersection-times on the first path.

`firstintersectionpath` Returns a path consisting of the points of intersection of the paths, sorted on time for the first one. Points will belong to the first path.

`secondintersectionpath` As above, but sorted on the second one, and with points from the second.

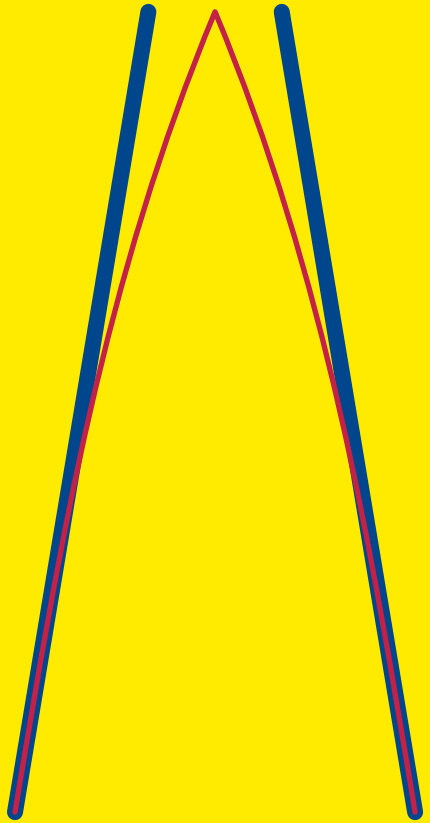
`intersectionpath` Sorted on first, but takes the mean of the corresponding points (so maybe not on any of the paths).

There is also

```
cutbeforefirst
cutafterfirst
cutbeforelast
cutafterlast
selfintersectiontimeslist
```

to play with.

`intersectionprecision` can be used to speed up (or slow down).



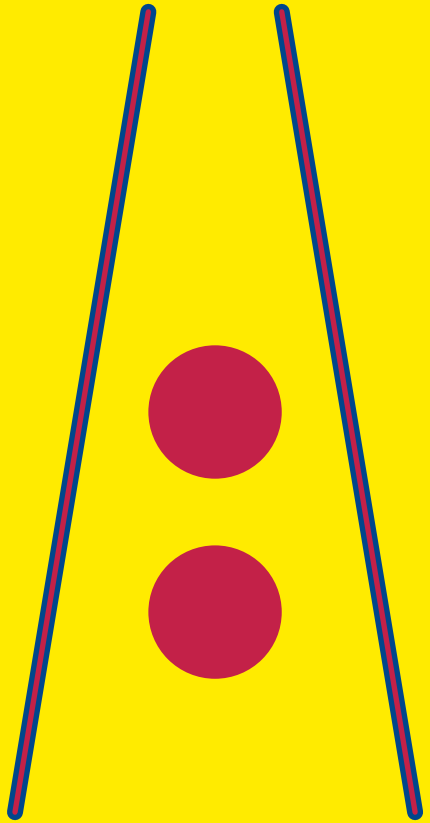
JOINING PATHS WITH &

`&` The traditional joiner. The last point of the first path must coincide with the first point of the second. Otherwise we get an error:

```
error: Paths don't touch; '&' will be changed to '..'

When you join paths 'p & q', the ending point of p
must be exactly equal to the starting point of q.
So I'm going to pretend that you said 'p .. q' instead.
```

Whether they are the same can be controlled with `jointolerance`.

```
path p[] ;
p[1] := (-75,-150) -- (-25, 150) ;
p[2] := ( 25, 150) -- ( 75,-150) ;
draw p[1] withpen pencircle scaled 6 ;
draw p[2] withpen pencircle scaled 6 ;
interim jointolerance := 50 ;
draw p[1] & p[2]
     withpen pencircle scaled 2
     withcolor "MyColors:3" ;
```



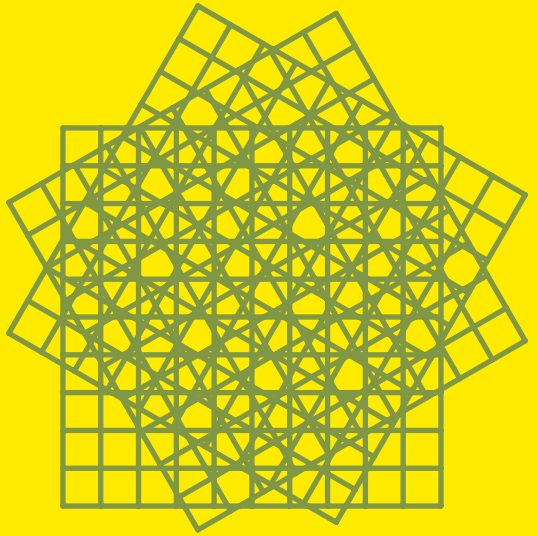
JOINING PATHS WITH &&

`&&` A new way to join paths. This will generate a `moveto` in the PDF file.

```
path p[] ;
p[1] := (-75,-150) -- (-25, 150) ;
p[2] := ( 25, 150) -- ( 75,-150) ;
draw p[1] withpen pencircle scaled 6 ;
draw p[2] withpen pencircle scaled 6 ;
draw p[1] && p[2]
     withpen pencircle scaled 2
     withcolor "MyColors:3" ;

fill (fullcircle scaled 50 shifted (0,-75)
      &&
      fullcircle scaled 50
      &&
      cycle)
     withcolor "MyColors:3" ;
```

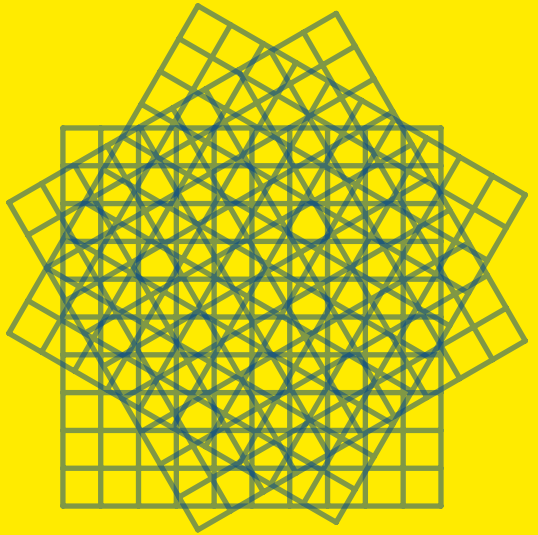
THE GRID IS ONE PATH(!)

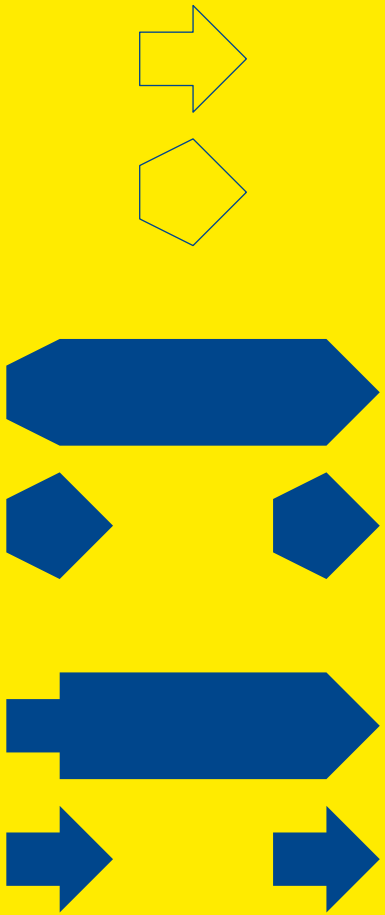
```
path basegrid, grid ;

basegrid :=
  for i = -5 upto 5 :
    ((-5,i) -- (5,i)) scaled 5mm &&
    ((i,-5) -- (i,5)) scaled 5mm &&
  endfor
  nocycle ;

grid := basegrid &&
        basegrid rotated 30
        shifted (2mm,6mm) &&
        basegrid rotated 60
        shifted (2mm,7mm) ;

draw grid
     withpen pencircle scaled 2
     withtransparency (1, 0.5) ;
```



THE GRID WITH MORE PATHS

```
path basegrid, grid[] ;

basegrid :=
  for i = -5 upto 5 :
    ((-5,i) -- (5,i)) scaled 5mm &&
    ((i,-5) -- (i,5)) scaled 5mm &&
  endfor
  nocycle ;

grid[1] := basegrid ;
grid[2] := basegrid rotated 30
                    shifted (2mm,6mm) ;
grid[3] := basegrid rotated 60
                    shifted (2mm,7mm) ;

for i = 1 upto 3 :
  draw grid[i]
       withpen pencircle scaled 2
       withtransparency (1, 0.5) ;
endfor
```





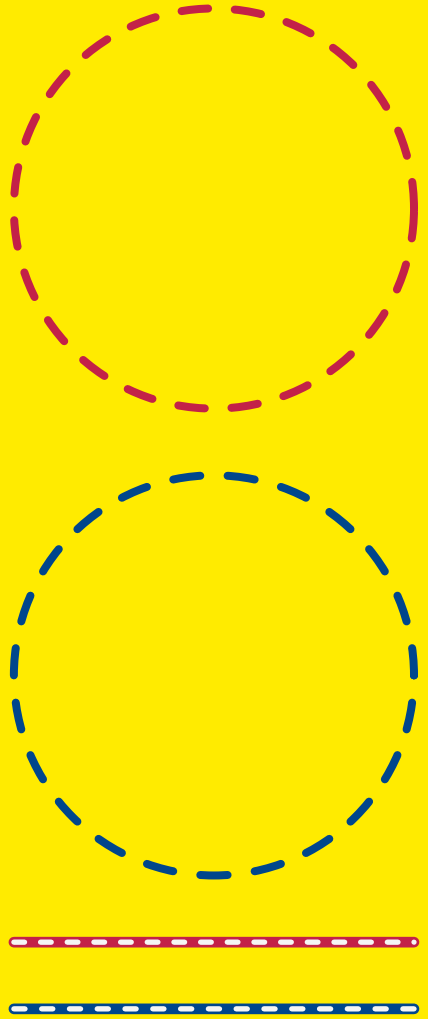
AN ARROW PEN? NEP!

```
path parr ;
parr := (-20,-10) -- (0,-10) -- (0,-20) -- (20,   0)
                  -- (0, 20) -- (0, 10) -- (-20, 10) --
cycle ;
pen parrpen ; parrpen := makepen(parr) ;
nep parrnep ; parrnep := makenep(parr) ;

draw parr           shifted(0,150) ;
draw convexed(parr) shifted(0,100) ;

draw    (-50, 25) -- (50,25) withpen parrpen ;
drawdot (-50,-25)            withpen parrpen ;
drawdot ( 50,-25)            withpen parrpen ;

draw    (-50,-100) -- (50,-100) withpen parrnep ;
drawdot (-50,-150)              withpen parrnep ;
drawdot ( 50,-150)              withpen parrnep ;
```

DASHED EVENLY?

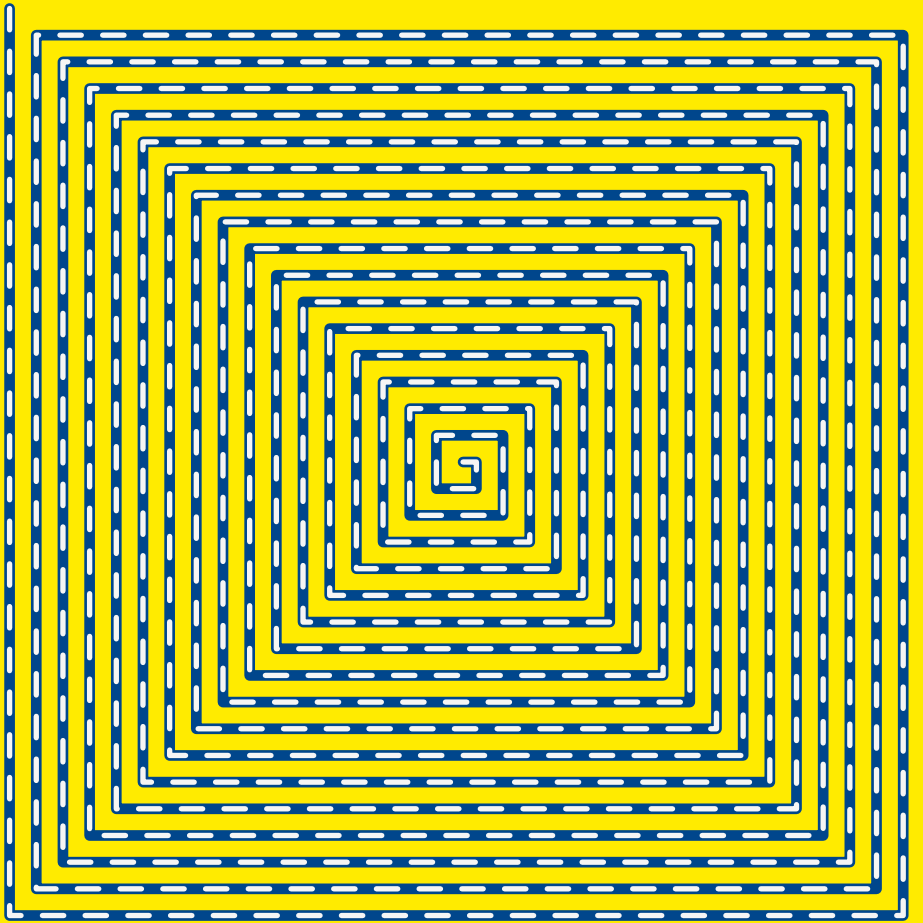
```
draw fullcircle scaled 150
                shifted (0,125)
                dashed dashpattern(on 10 off 10)
                withpen pencircle scaled 3
                withcolor "MyColors:3" ;
draw fullcircle scaled 150
                shifted (0,-50)
                withdashes 10
                withpen pencircle scaled 3 ;
```

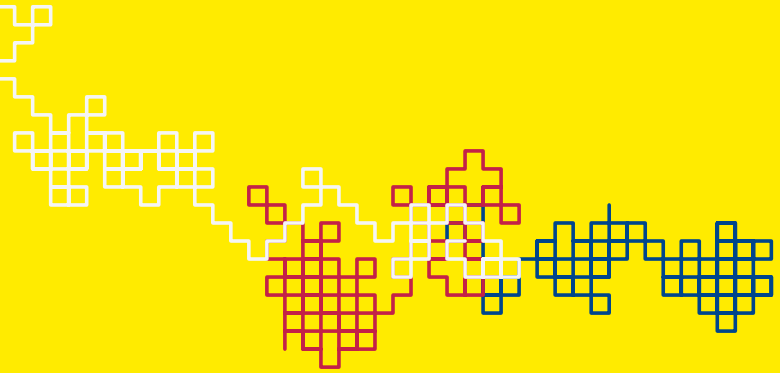
This one is a bit costly, since it has to calculate the arclength of the path.



RELATIVE POINTS

```
fill unitsquare
     scaled PaperHeight
     withcolor "MyColors:2" ;

path p ;

p := origin
     for i = 0 upto 16 :
       -- xrelative   20i +  5
       -- yrelative -(20i + 10)
       -- xrelative -(20i + 15)
       -- yrelative   20i + 20
     endfor ;

draw p withpen pencircle scaled 4 ;

draw p withdashes (8,8)
       withpen pencircle scaled 2
       withcolor "MyColors:4" ;
```





RELATIVE AND ABSOLUTE MOVEMENTS

`xrelative dx` Create a point that is shifted `dx` in horizontal direction relative to the previous point.

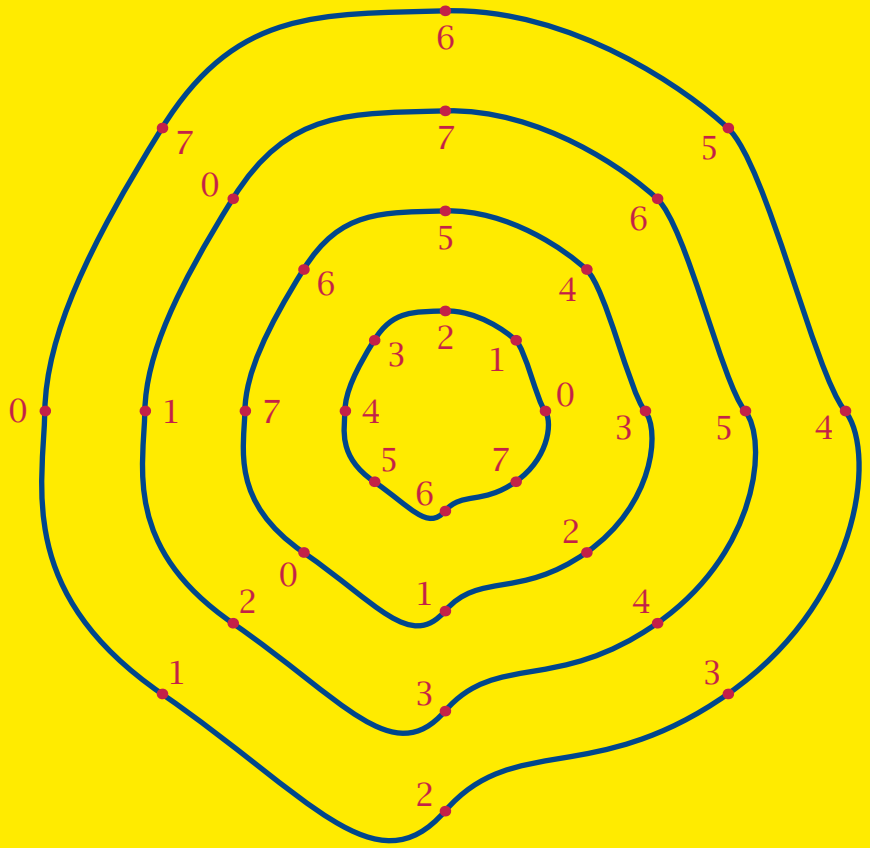
`yrelative dy` Create a point that is shifted `dy` in vertical direction relative to the previous point.

`xyrelative (dx,dy)` Create a point that is shifted `(dx,dy)` relative to the previous point. Shift `(dx,dx)` if the numerical value `dx` is given.

`xabsolute x` Create a point, shifted horizontally compared to the previous point, so that the first coordinate becomes `x`.

`yabsolute y` Create a point, shifted vertically compared to the previous point, so that the second coordinate becomes `y`.

```
draw origin
  for i = 1 upto 100 :
    -- xrelative(5*((-1)**(round(uniformdeviate(1)))))
    -- yrelative(5*((-1)**(round(uniformdeviate(1))))) endfor
  withpen pencircle scaled 2 ;
```

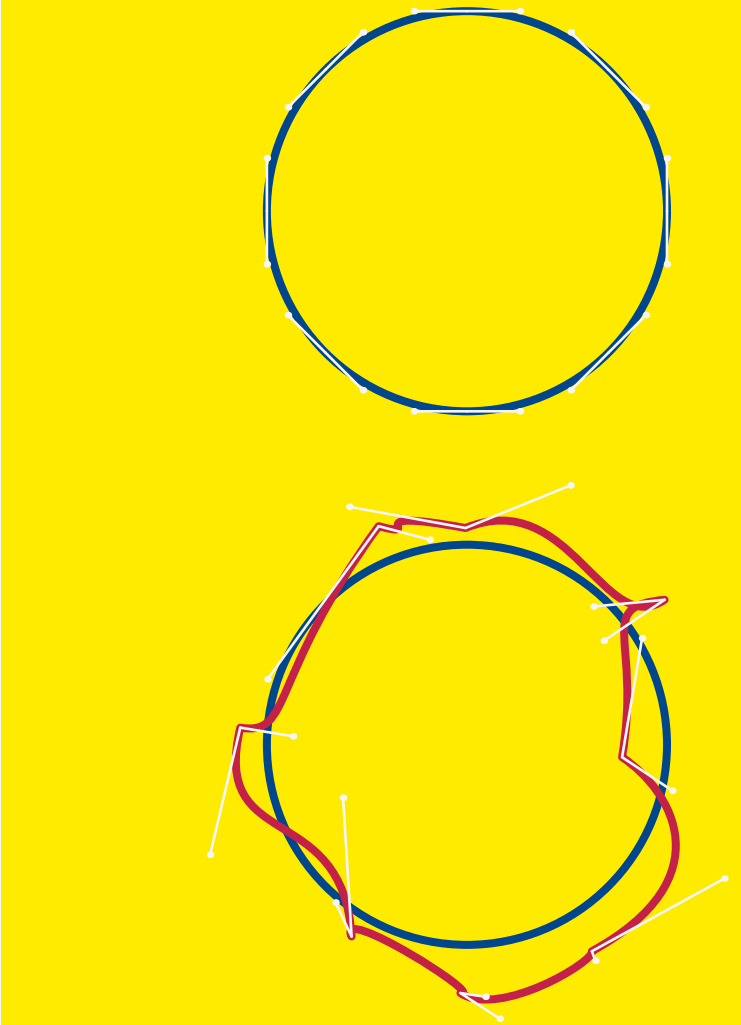


RECYCLED PATH

```
path p[] ;
p[0] := fullcircle
        randomizedcontrols .2 ;
p[1] := p[0] scaled 75                ;
p[2] := p[0] scaled 150 recycled  3 ;
p[3] := p[0] scaled 225 recycled -3 ;
p[4] := p[0] scaled 300 recycled 12 ;
pickup pencircle scaled 2 ;
for i = 1 upto 4 :
  draw p[i] ;
  drawpoints p[i]
    withcolor "MyColors:3" ;
  drawpointlabels p[i]
    withcolor "MyColors:3" ;
endfor
```

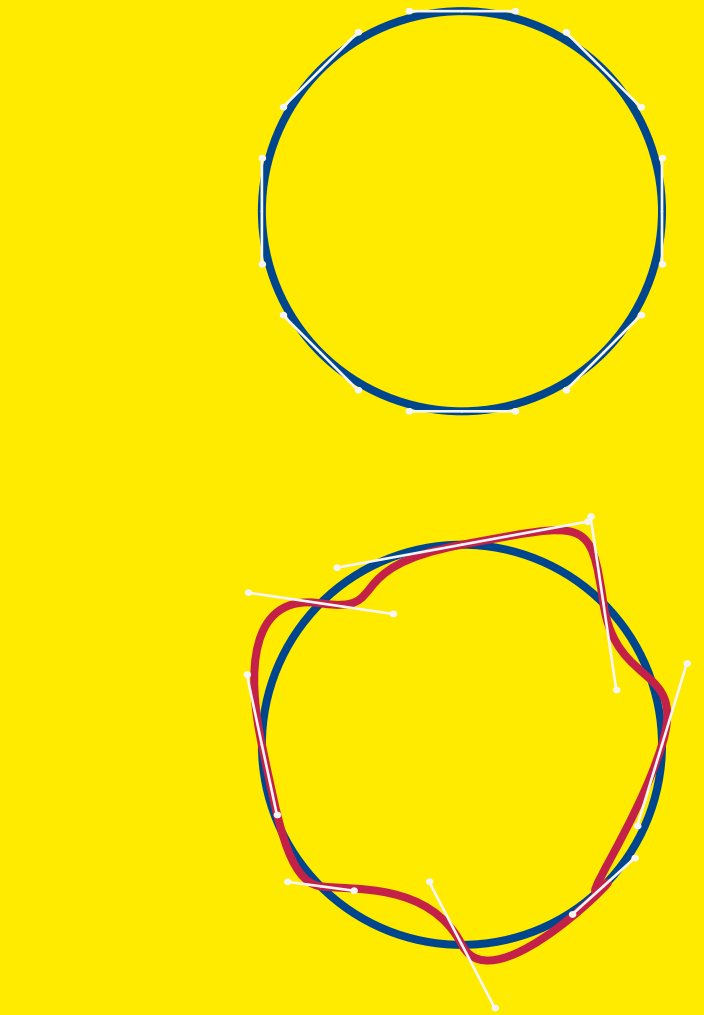
RANDOMIZE A PATH (OLD)

```
path p ;
p := fullcircle scaled 150 shifted (0,100) ;
draw p withpen pencircle scaled 3 ;
drawcontrollines p withcolor "MyColors:4" ;
drawcontrolpoints p withcolor "MyColors:4" ;

p := p shifted (0,-200) ;
draw p withpen pencircle scaled 3 ;

p := p randomized 60 ;
draw p withpen pencircle scaled 3
       withcolor "MyColors:3" ;
drawcontrollines  p withcolor "MyColors:4" ;
drawcontrolpoints p withcolor "MyColors:4" ;
```



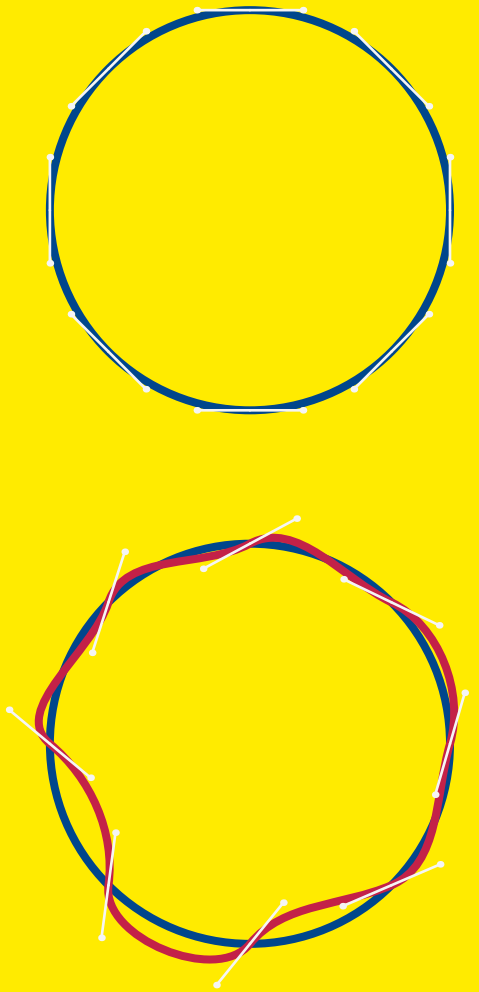
RANDOMIZE CONTROLPOINTS (OLD)

```
path p ;
p := fullcircle scaled 150 shifted (0,100) ;
draw p withpen pencircle scaled 3 ;
drawcontrollines p withcolor "MyColors:4" ;
drawcontrolpoints p withcolor "MyColors:4" ;

p := p shifted (0,-200) ;
draw p withpen pencircle scaled 3 ;

p := p randomizedcontrols 60 ;
draw p withpen pencircle scaled 3
       withcolor "MyColors:3" ;
drawcontrollines  p withcolor "MyColors:4" ;
drawcontrolpoints p withcolor "MyColors:4" ;
```



ROTATING CONTROLPOINTS

```
path p ;
p := fullcircle scaled 150 shifted (0,100) ;
draw p withpen pencircle scaled 3 ;
drawcontrollines p withcolor "MyColors:4" ;
drawcontrolpoints p withcolor "MyColors:4" ;

p := p shifted (0,-200) ;
draw p withpen pencircle scaled 3 ;

p := p randomrotatedcontrols 120 ;
draw p withpen pencircle scaled 3
       withcolor "MyColors:3" ;
drawcontrollines  p withcolor "MyColors:4" ;
drawcontrolpoints p withcolor "MyColors:4" ;
```

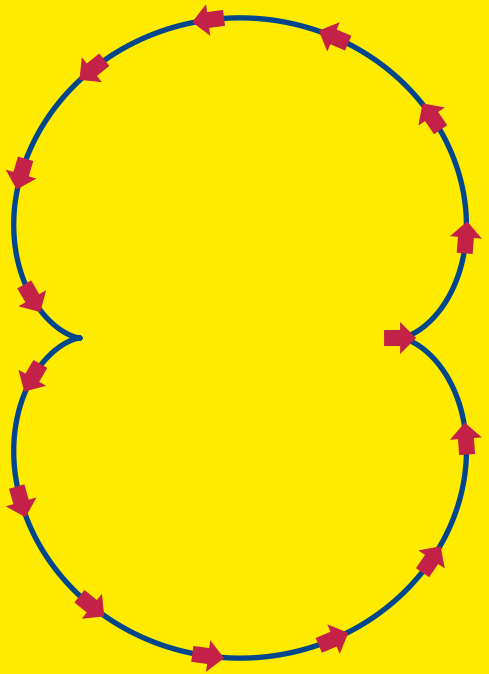


LOOPING OVER PATHS

We take a look at the macro behind rotation of control lines.

```
vardef mfun_randomrotated_path(expr p, s) =
  save r, oldr, newr, firstr, l ;
  l := length(p) ;
  newr := (-1/2+uniformdeviate(1))*s ;
  firstr := newr ;
  for i within p :
    hide (
      oldr := newr ;
      newr := (-1/2+uniformdeviate(1)) * s ;
    )
    pathpoint ..
    controls  (pathpostcontrol rotatedaround(pathpoint, oldr) )
    and      ((deltaprecontrol 1) rotatedaround (deltapoint 1,
              if (i <> l - 1) : newr else : firstr fi)) ..
  endfor if cycle p : cycle else : nocycle fi
enddef ;
```

Notice the `pathpoint`, `deltapoint`, `pathpostcontrol`, `deltaprecontrol` and `nocycle`. They are all new.



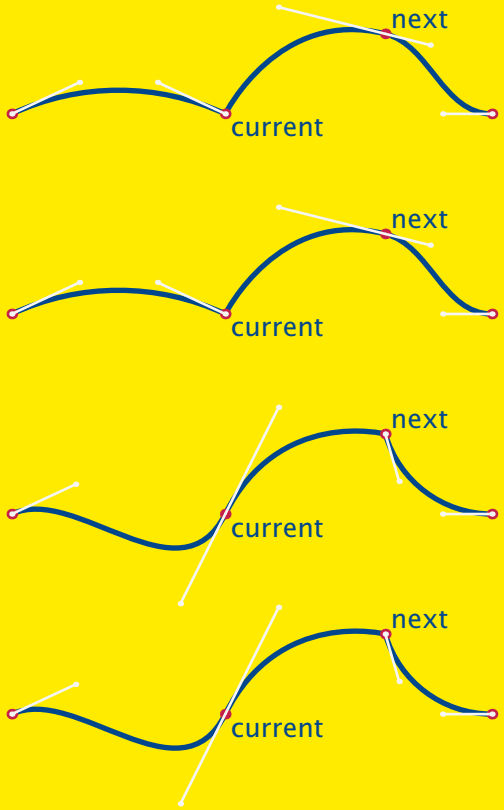
LOOPING OVER PATHS

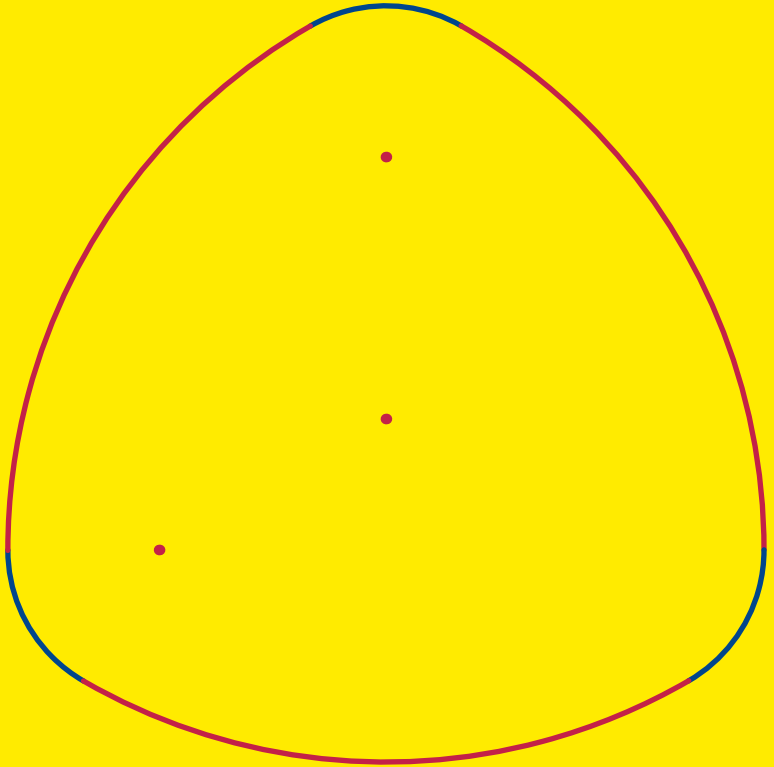
The macro `dashing` is defined via the fast and flexible looping over paths (note the `arcpointlist` and `pathdirection`).

```
vardef dashing (expr pth, shp, stp) =
  for i within arcpointlist stp of pth :
    shp
      rotated angle(pathdirection)
      shifted pathpoint
    &&
  endfor nocycle
enddef ;

path p ;
p := function(1,"3*cos(x) - cos(3*x)", "3*sin(x) - sin(3*x)",
                epsed(0), epsed(2pi), 0.001) scaled 30 ;
draw p withpen pencircle scaled 2 ;

path parr ;
parr := (-20,-10) -- (0,-10) -- (0,-20) -- ( 20, 0)
                  -- (0, 20) -- (0, 10) -- (-20,10) -- cycle ;

fill dashing (p, parr scaled .3, 15) && cycle
     withcolor "MyColors:3" ;
```

TAKE CONTROL!

```
path p[] ;
p[0] = origin {dir 25} .. (80,0) .. controls ( 80, 0) and (100,40)
                       .. (140,30) .. {dir 0} (180,0) ;
p[1] = origin {dir 25} .. (80,0) .. secondcontrol         (100,40)
                       .. (140,30) .. {dir 0} (180,0) ;
p[2] = origin {dir 25} .. (80,0) .. controls (100,40) and (140,30)
                       .. (140,30) .. {dir 0} (180,0) ;
p[3] = origin {dir 25} .. (80,0) .. firstcontrol (100,40)
                       .. (140,30) .. {dir 0} (180,0) ;

for i = 0 upto 3 :
  p[i] := p[i] shifted (-90,75+37.5-75*i) ;
  draw p[i] withpen pencircle scaled 2 ;
  drawpoints p[i] withcolor "MyColors:3" ;
  drawcontrollines p[i]  withcolor "MyColors:4" ;
  drawcontrolpoints p[i] withpen pencircle scaled 2
                         withcolor "MyColors:4" ;
  label.lrt("\small\sans current", point 1 of p[i]) ;
  label.urt("\small\sans next",    point 2 of p[i]) ;
endfor
```





REULEAUX VIA ARC

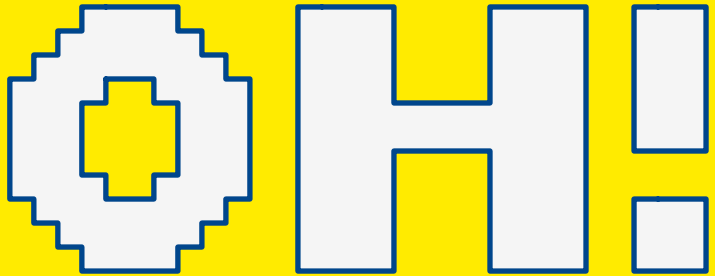
```
z0 = (0,6/sqrt(3)*cm) ;
z1 = z0 rotated 120 ;

path ARC[] ;
ARC[1] := arc(60,120) scaled 4cm
                      shifted z0 ;
ARC[2] := arc(0,  60) scaled 16cm
                      shifted z1 ;
pickup pencircle scaled 2 ;

for i = 0 upto 2 :
  draw ARC[1] rotated 120i ;
  draw ARC[2] rotated 120i
       withcolor "MyColors:3" ;
endfor

drawpoints origin -- z0 -- z1
           withcolor "MyColors:3" ;
```

POTRACING

```
string s ; s :=
  "000011110000011110000111100111
   000111111000011110000111100111
   001111111100011110000111100111
   011110011110011110000111100111
   011100001110011111111111100111
   011100001110011111111111100111
   011100001110011110000111100000
   011110011110011110000111100000
   001111111100011110000111100111
   000111111000011110000111100111
   000011110000011110000111100111" ;

lmt_startpotraced [ bytes = s ] ;

path p ; p := lmt_potraced [
  % polygon = true,
  % threshold = 0.15,
] ;  p := p scaled 9 ; p := p shifted -center p ;

fill p withcolor "MyColors:4" ;
draw p withpen pencircle scaled 2 ;

lmt_stoppotraced ;
```

POTRACING

```
string s ; s :=
  "00001111000001111000011110011100
   00011111100001111000011110011100
   00111111110001111000011110011100
   01111001111001111000011110011100
   01110000111001111111111110011100
   01110000111001111111111110011100
   01110000111001111000011110000000
   01111001111001111000011110000000
   00111111110001111000011110011100
   00011111100001111000011110011100
   00001111000001111000011110011100" ;

lmt_startpotraced [ bytes = s ] ;

path p ; p := lmt_potraced [
  polygon = true,
  % threshold = 0.15,
] ;  p := p scaled 9 ; p := p shifted -center p ;

fill p withcolor "MyColors:4" ;
draw p withpen pencircle scaled 2 ;

lmt_stoppotraced ;
```

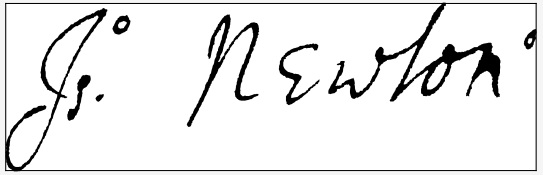


POTRACING

```
string s ; s :=
  "000011110000011110000111100111
   000111111000011110000111100111
   001111111100011110000111100111
   011110011110011110000111100111
   011100001110011111111111100111
   011100001110011111111111100111
   011100001110011110000111100000
   011110011110011110000111100000
   001111111100011110000111100111
   000111111000011110000111100111
   000011110000011110000111100111" ;

lmt_startpotraced [ bytes = s ] ;

path p ; p := lmt_potraced [
  % polygon = true,
  threshold = 0.15,
] ;  p := p scaled 9 ; p := p shifted -center p ;

fill p withcolor "MyColors:4" ;
draw p withpen pencircle scaled 2 ;

lmt_stoppotraced ;
```

POTRACING FROM BITMAP IMAGE

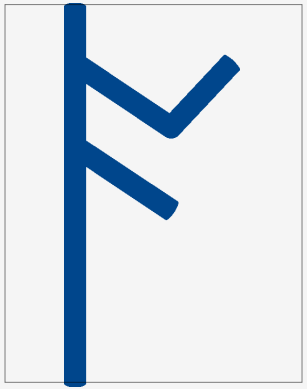
We start from this black and white png image



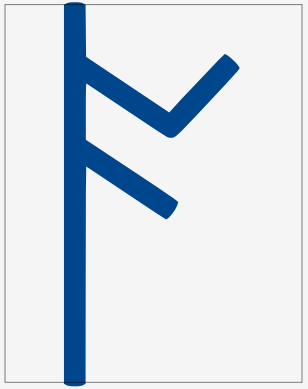
With the code below we get the result in the margin.

```
path p ; p := lmt_potraced [
  filename  = "newton.png",
  criterium = 1,
] ;
p := p xsized 200 ;
p := p shifted -center p ;

path b ;
b := last_potraced_bounds xsized 200 ;
b := b shifted -center b ;

fill b ;
fill p withcolor "MyColors:2" ;
draw b withpen pencircle scaled 1 withcolor "MyColors:3" ;
```

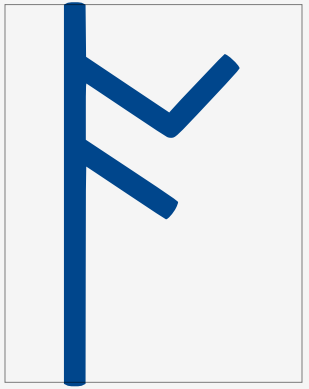
POTRACING FONTS



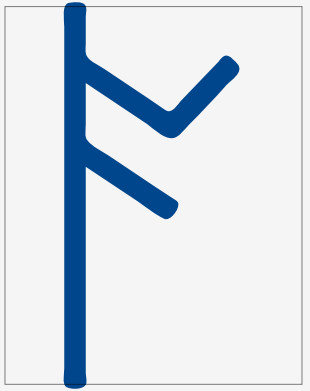
Type3: pk



Type3: MetaPost



Opentype: cff



Type1: pfb

Type3: pk

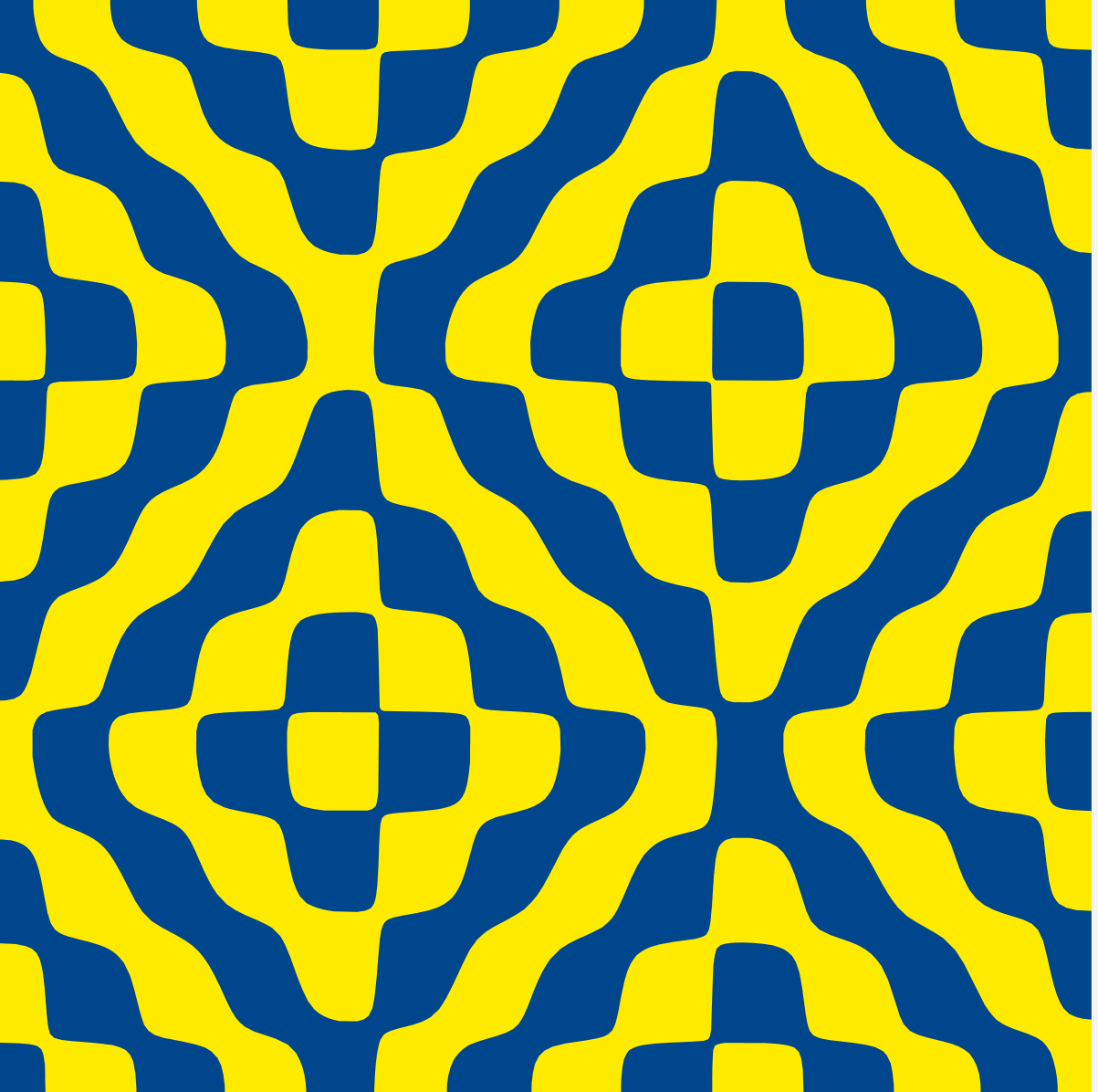
Type3: MetaPost

Opentype: cff

Type1: pfb

POTRACING MATH (LUA)

```
\startluacode
local cos, pi = math.cos, math.pi
local N       = 2000
local pp      = pi/N
local ppA     = 10 * pp
local ppB     = 13 * pp
local cosppAy = 0
local cosppBy = 0
local function f(x,y)
  if x == 1 then
    cosppAy = cos(ppA*y)
    cosppBy = cos(ppB*y)
  end
  local z = cos(ppB*x)*cosppAy + .55 * cos(ppA*x)*cosppBy
  if z > 0 then
    return '1'
  else
    return '0'
  end
end
potrace.setbitmap("mybitmap", potrace.contourplot(N,N,f))
\stopluacode
```

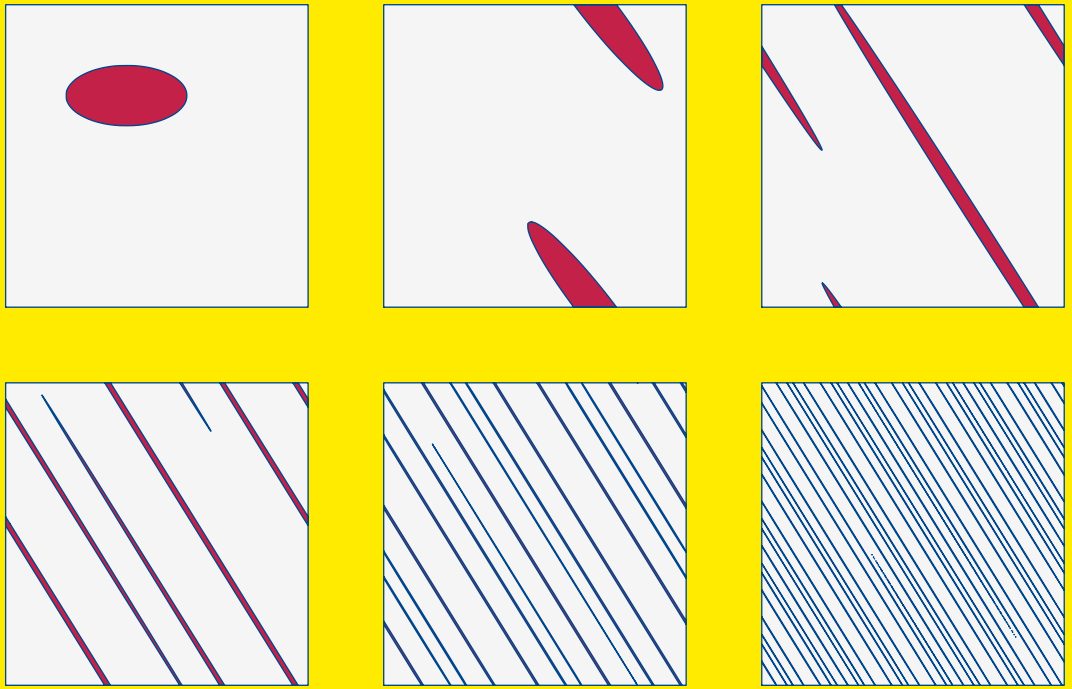


POTRACING MATH

```
path p ; p := lmt_potraced [
  stringname = "mybitmap",
  value      = "1",
  threshold  = 0.25,
  optimize   = true,
] ;

p := p ysized PaperHeight ;
p := p shifted (-center p) ;
fill p ;
```

AN ITERATED MAP

```
\startluacode
local fmod = math.fmod
function MP.SetMyIteratedMap(n,N)
    local function f(x,y)
        x=x/N
        y=y/N
        for i = 1,n do
          x,y = fmod(2*x + y,1),fmod(x + y,1)
        end
        if ((x - 0.4)/0.2)^2 + ((y - 0.7)/0.1)^2 < 1 then
            return '1'
        else
            return '0'
        end
    end
    potrace.setbitmap("iteratedmap", potrace.contourplot(N,N,f))
end
\stopluacode
```

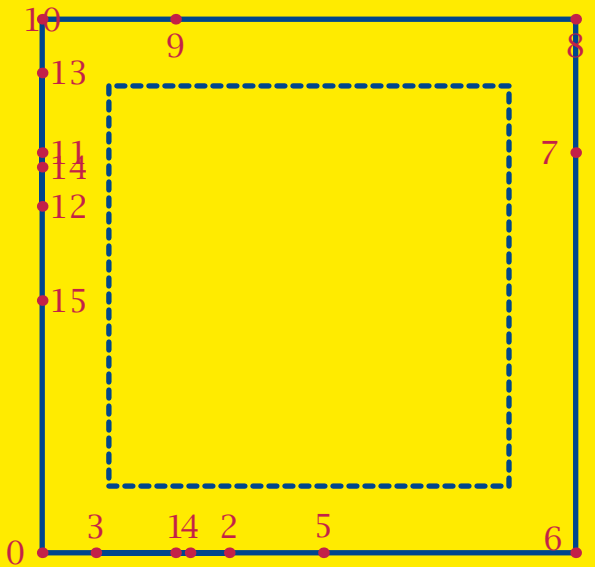


AN ITERATED MAP

```
numeric N ; N := 1000 ;
path p[], b[] ;
for i = 0 upto 5 :
  lua.MP.SetMyIteratedMap(i,N);
  p[i] := lmt_potraced [
    stringname = "iteratedmap",
    value      = "1",
    size       = 0,
  ] ;
  b[i] := fullsquare scaled 4cm ;
  b[i] := b[i]
  shifted ((i - 1)*5cm, 2.5cm ) ;
  p[i] := p[i] scaled (1/N*4cm) ;
  p[i] := p[i] shifted (-2cm,-2cm)
    shifted ((i - 1)*5cm, 2.5cm ) ;
  if i > 2:
  p[i] := p[i] shifted (-15cm,-5cm) ;
  b[i] := b[i] shifted (-15cm,-5cm) ;
  fi
  fill b[i] withcolor "MyColors:4" ;
  fill p[i] withcolor "MyColors:3" ;
  draw b[i] && p[i] ;
endfor
```

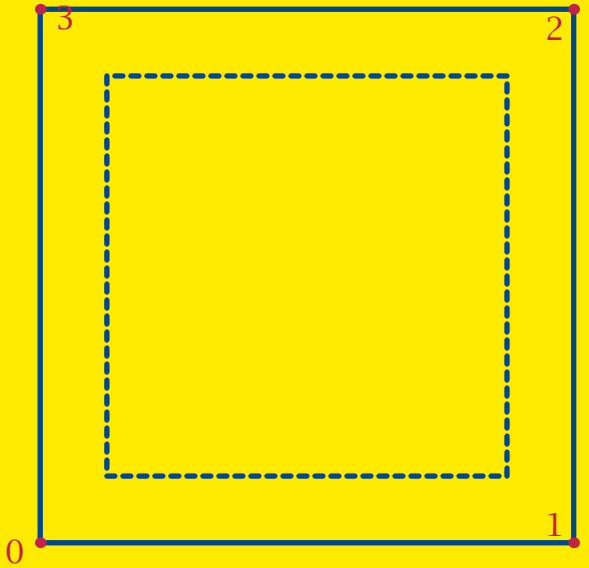
AN ENVELOPED SQUARE

```
path p, ep ;

p  := fullsquare scaled 150 ;
ep := envelope(pensquare scaled 50) of p ;

pickup pencircle scaled 2 ;
draw p  dashed evenly ;
draw ep               ;
drawpoints ep         withcolor "MyColors:3" ;
drawpointlabels ep    withcolor "MyColors:3" ;
```



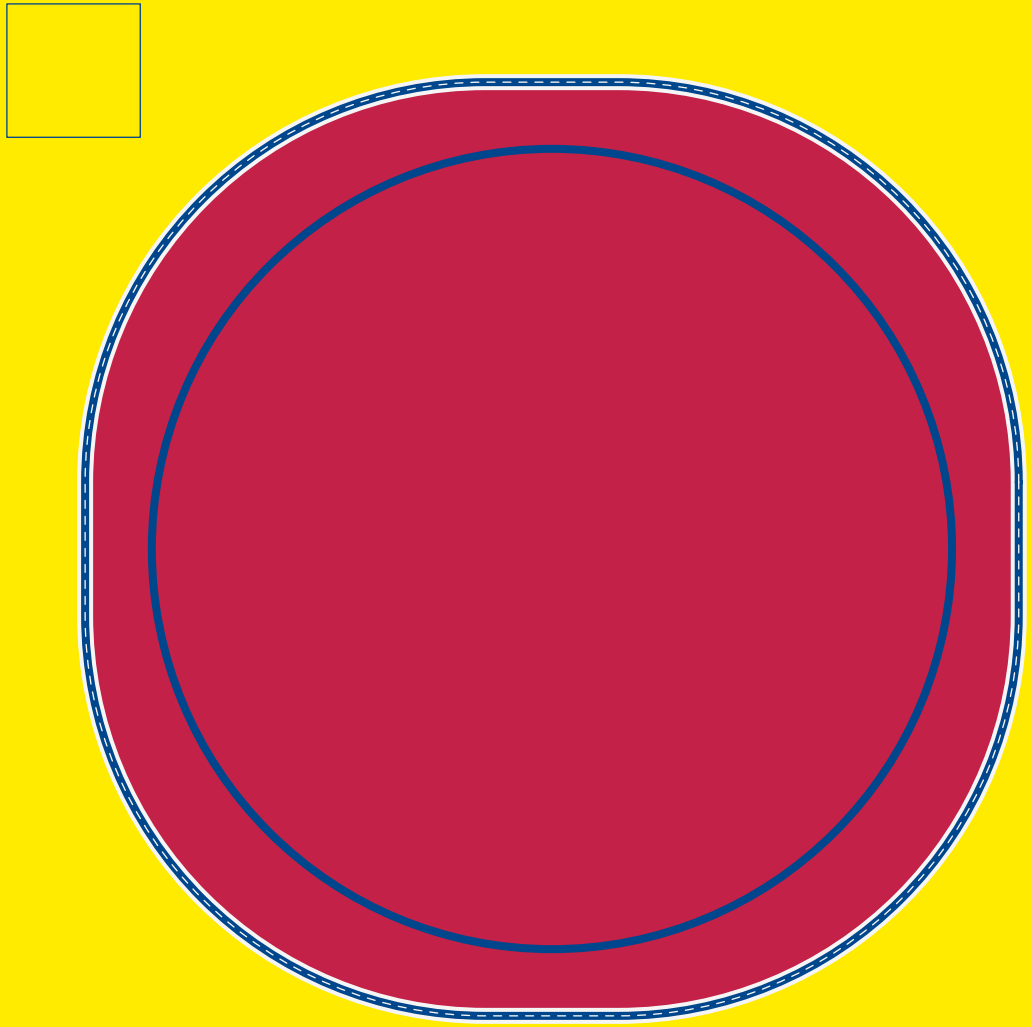
AN ENVELOPED SQUARE (REDUCED)

```
path p, ep ;

p  := fullsquare scaled 150 ;
ep := envelope(pensquare scaled 50) of p ;
ep := reducedpath(ep) ;

pickup pencircle scaled 2 ;
draw p  dashed evenly ;
draw ep               ;
drawpoints ep         withcolor "MyColors:3" ;
drawpointlabels ep    withcolor "MyColors:3" ;
```



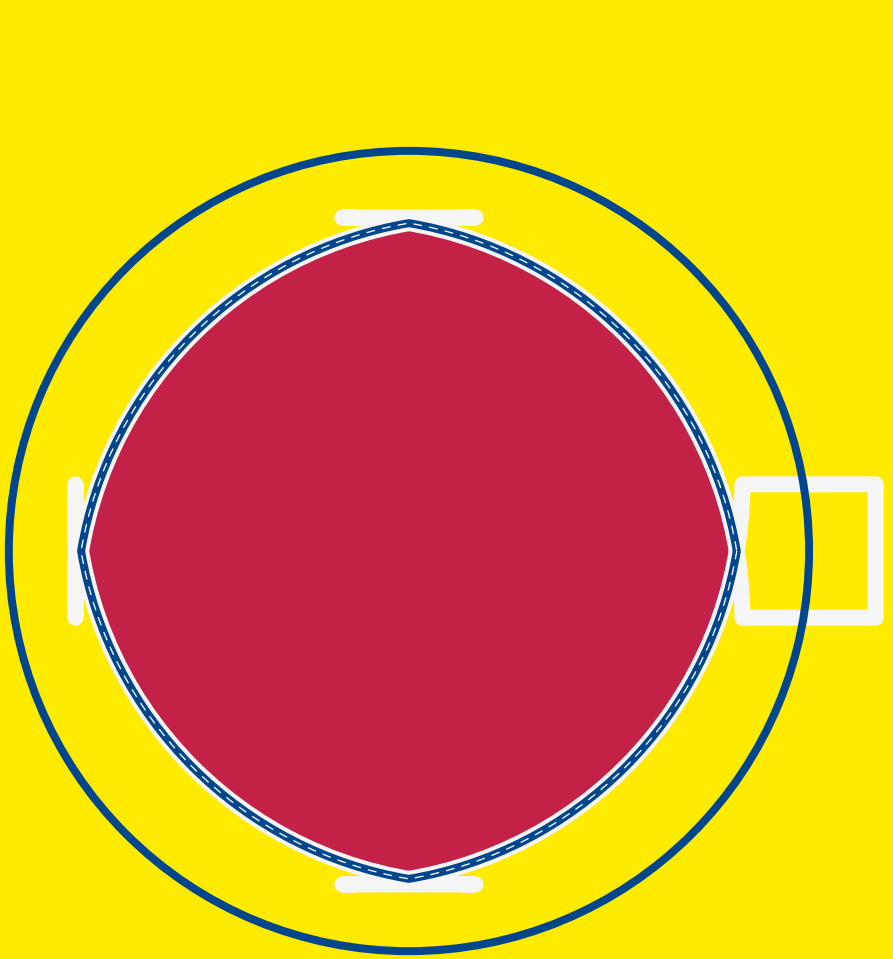
ENVELOPES

```
def DrawEnvelope(expr Path, Pen) =
     save EnvPath ; path EnvPath ;
     EnvPath := Path enveloped Pen ;
     fill EnvPath withcolor "MyColors:3" ;
     draw envelope(Pen) of Path withpen pencircle scaled 6
                                withcolor "MyColors:4" ;
     draw EnvPath withpen pencircle scaled 3 ;
     draw EnvPath dashed evenly withcolor "MyColors:4" ;
     draw Path withpen pencircle scaled 3 ;
     draw makepath(Pen) shifted (-0.4PaperHeight,0.4PaperHeight) ;
enddef ;
```



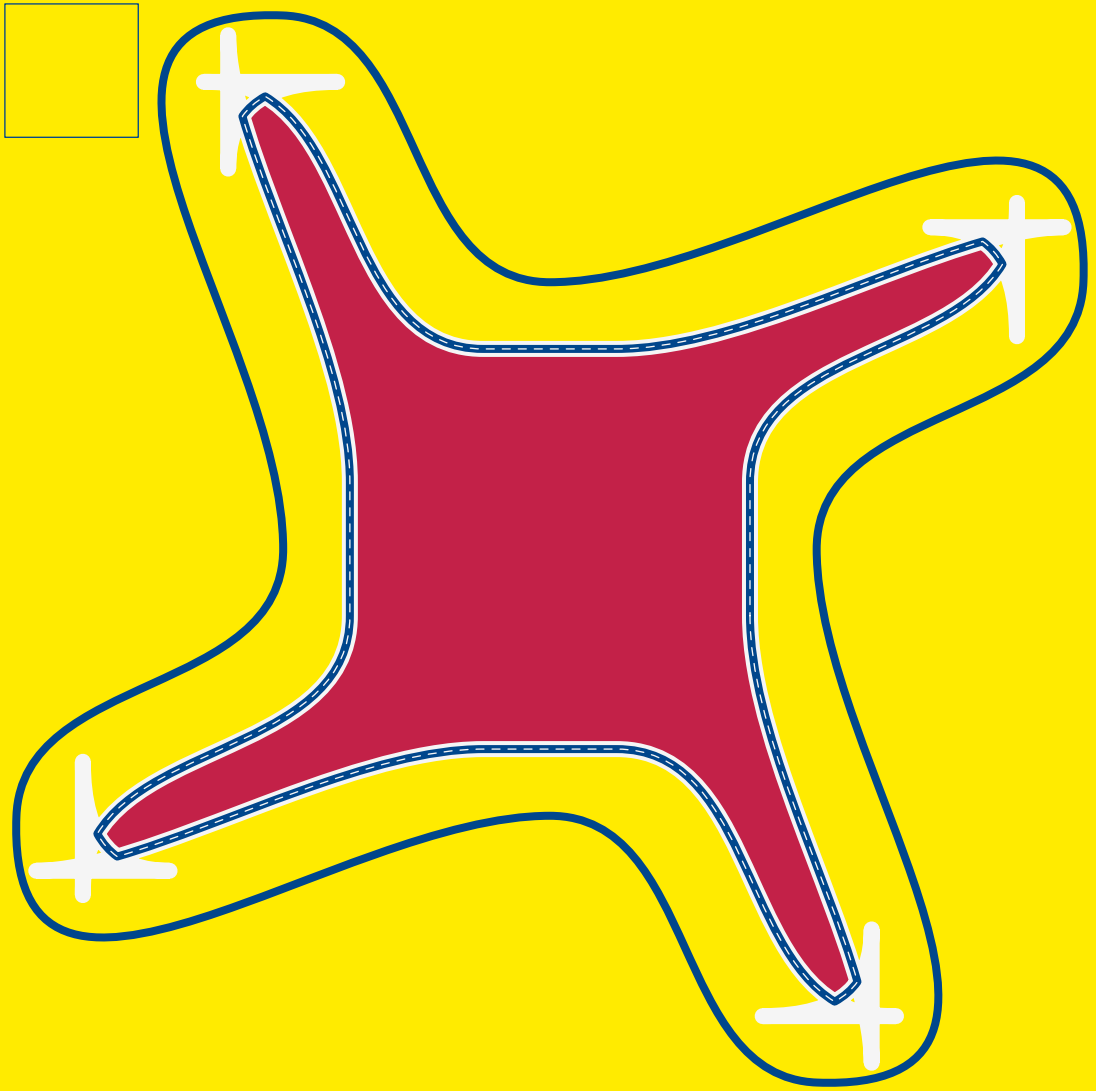
ENVELOPES

```
path p ; pen q ;

p := fullcircle scaled 300 ;

q := pensquare scaled 50 ;

DrawEnvelope(p,q) ;
```



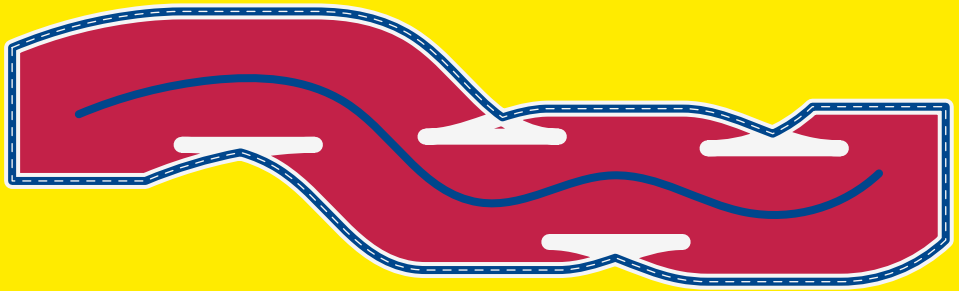
ENVELOPES

```
path p ; pen q ;

p := reverse fullcircle scaled 300 ;

q := pensquare scaled 50 ;

DrawEnvelope(p,q) ;
```



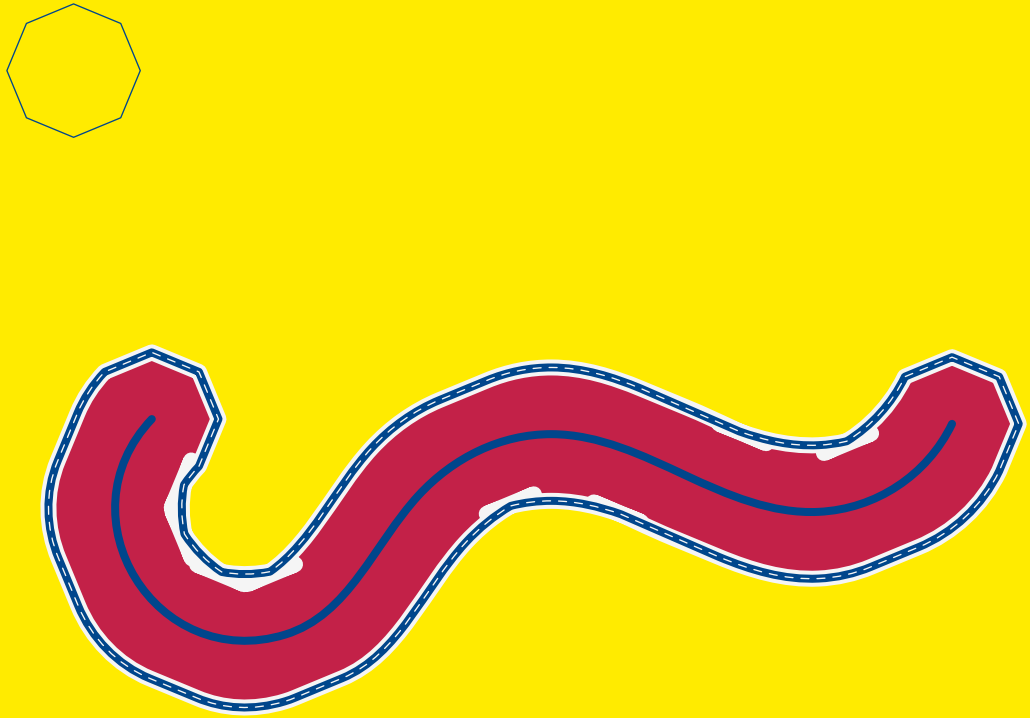
ENVELOPES

```
path p ; pen q ;

p := reverse (
        (100,0){up}
     .. xyrelative (100,100)
     .. {left}(0,100)
     .. xyrelative (-100,100)
     .. {down}(-100,0)
     .. xyrelative (-100,-100)
     .. {right}(0,-100)
     .. xyrelative (100,-100)
     .. {up}cycle );

q := pensquare scaled 50 ;

DrawEnvelope(p,q) ;
```



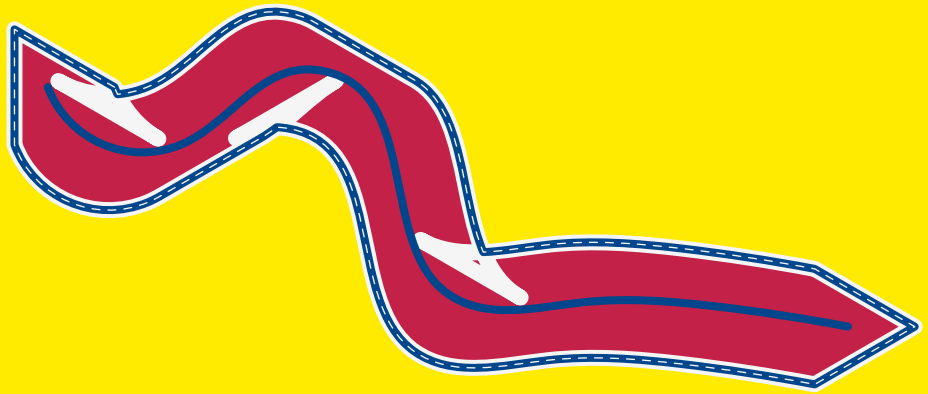
ENVELOPES

```
path p ; pen q ;

p :=
  for i = -150 step 50 until 150 :
    (i, -50 + uniformdeviate(100)) ..
  endfor nocycle ;

q := pensquare scaled 50 ;

DrawEnvelope(p,q) ;
```



ENVELOPES

```
path p ; pen q ;

p :=
  for i = -150 step 50 until 150 :
    (i, -50 + uniformdeviate(100)) ..
  endfor nocycle ;

q := makepen punked fullcircle
                    scaled 50 ;

DrawEnvelope(p,q) ;
```



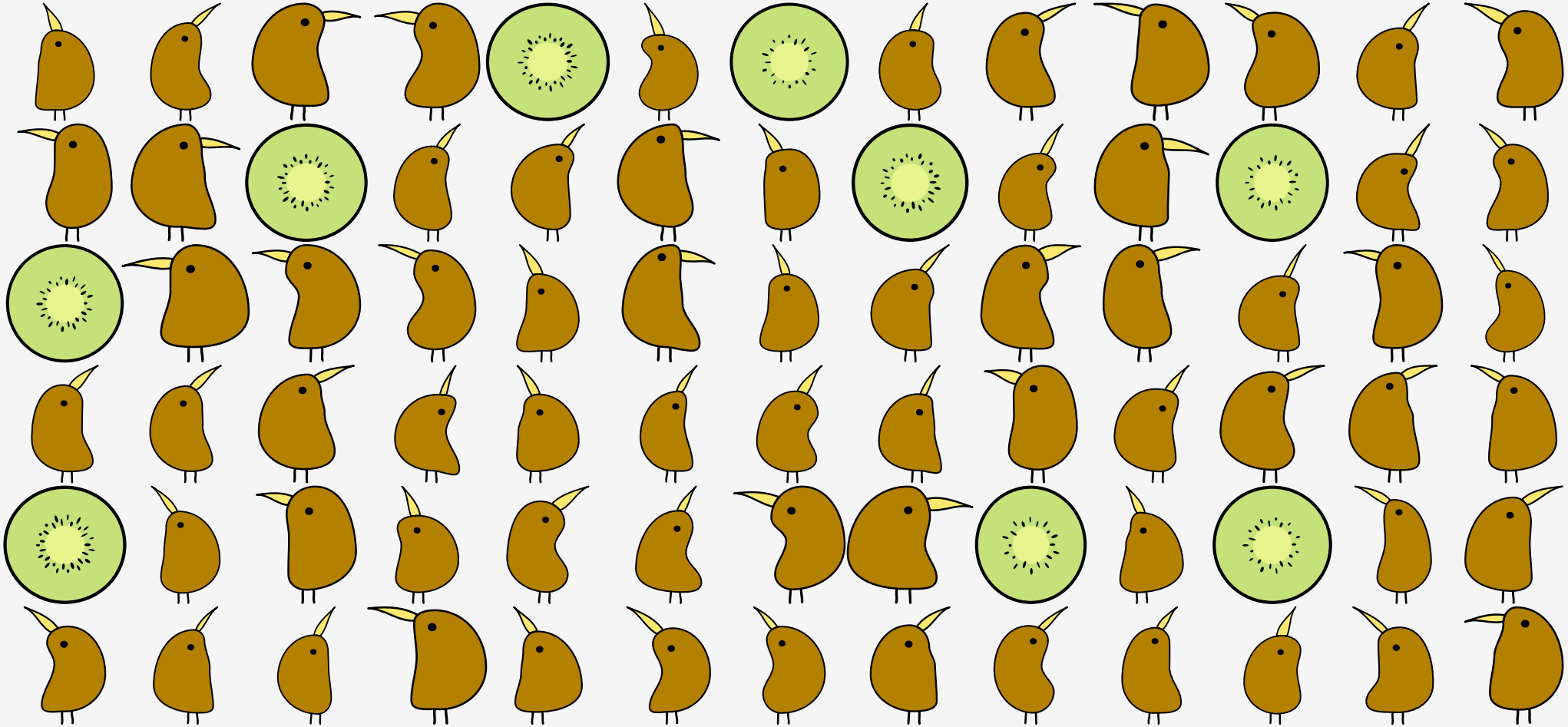
ENVELOPES

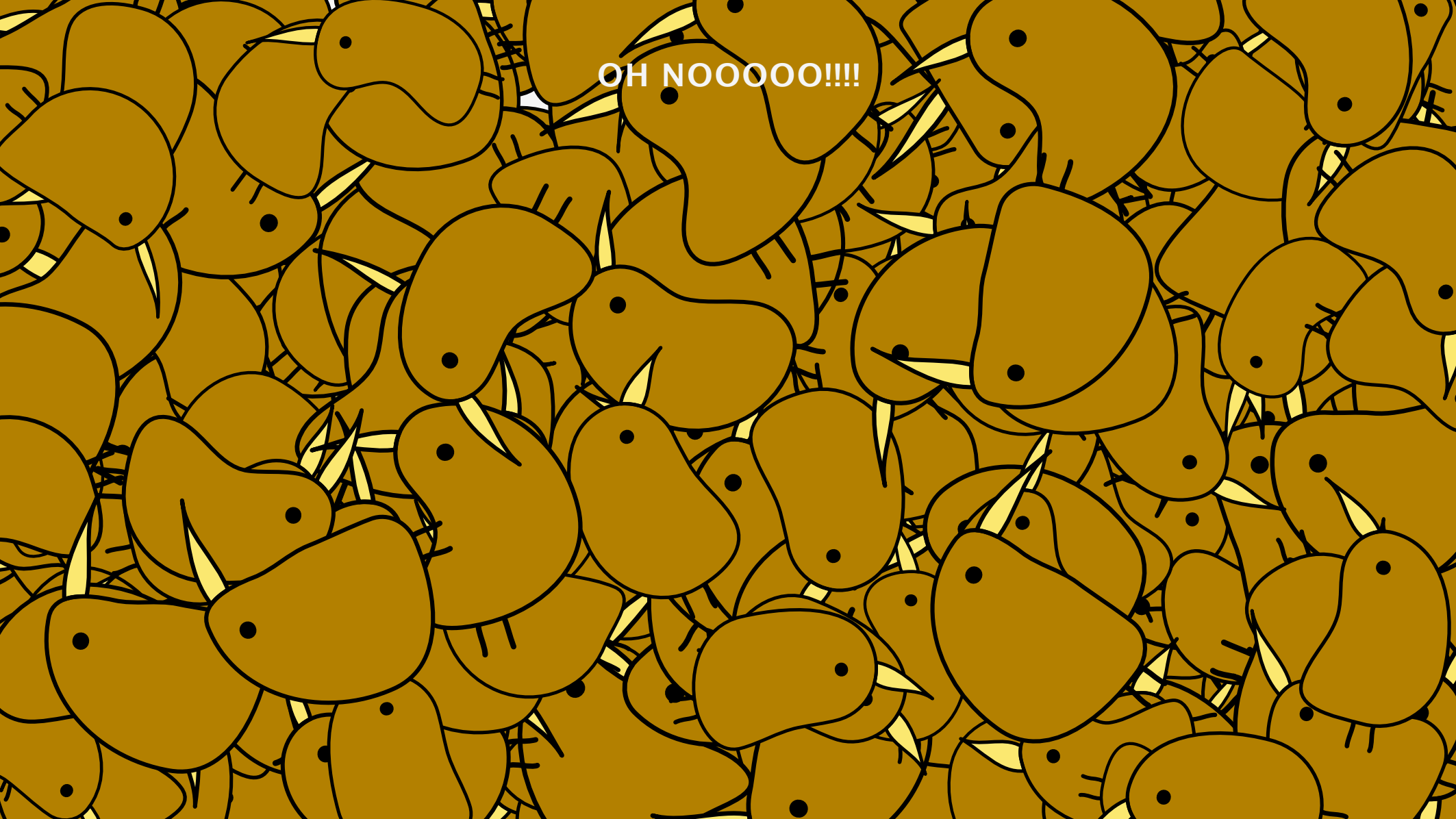
```
path p ; pen q ;

p :=
  for i = -150 step 50 until 150 :
    (i, -50 + uniformdeviate(100)) ..
  endfor nocycle ;

q := makepen fulltriangle scaled 50 ;

DrawEnvelope(p,q) ;
```

A FEW MORE NEWS

- `pathpoint`, `pathprecontrol`, `pathpostcontrol`, `pathfirst`, `pathlast`, `pathdirection`, `pathstate`, `pathindex`, `pathlastindex`, `pathlength`
- `withnestedprescript`, `withnestedpostscript`, `withstacking`, `withlinecap`, `withlinejoin`, `withmiterlimit`, `withcurvature`, `withnothing`
- `uncycle`
- `uncontrolled`
- `nolength`
- `stackingpart`
- `corners`, `xrange`, `yrange`
- `filled`, `clipped`, `grouped`, `bounded`
- `setgroups`, `setbounds`, `setclip`
- `boundingpath`
- `setproperty`
- `subarclength`
- `centerof`, `centerofmass`

A FEW KIWI(BIRD)S FROM YESTERDAY'S WORKSHOP




OH NOOOOO!!!!