Unusual bitmaps

Hans Hagen, Mikael P. Sundqvist

1 Introduction

In the early days of TEX, fonts mostly were bitmaps and when such a bitmap was shown in zeros and ones, the shape was rather recognizable. A recent example of a special-purpose (TAOCP) bitmap font is Don Knuths three–six font. Do you recognize this character?

```
1111111111
1100110011
1100110011
0000110000
0000110000
0000110000
0001111000
0001111000
```

It has a rather low resolution, but can still serve its purpose as we will see later on. One problem is of course that such a low resolution doesn't render too well. Although vector images are often preferred, especially in a MetaPost context, below we will explain how we can also use bitmaps in a constructive way.

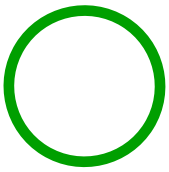
2 Vectorizing bitmaps

The potrace library by Peter Selinger is a nice tool: you feed it a bitmap and are rewarded with an outline specification. Details about the process can be found in `https://potrace.sourceforge.net/potrace.pdf`. When you limit yourself to only the basics, there are not that many source files and therefore I decided to add it to LuaMetaTEX in order to explore if we can do runtime conversions. Possible applications are logos and maybe converted bitmap fonts, although these can best be prepared beforehand because consistent metrics need to be taken care of. There are, however, other applications possible. Here I will discuss usage only in the perspective of Metafun, simply because we have to be visual, and Metafun is all about that. The library is of course accessible from the Lua end, if only because that way we can use it in Metafun.

We start with a simple example that also shows a potential usage:

```
\startMPcode
    string s ; s := "010 111 010";
    draw lmt_potraced [
        bytes = s,
    ] ysized 2cm
      withpen pencircle scaled 4
      withcolor darkgreen ;
\stopMPcode
```

We feed the `lmt_potraced` macro a 3×3 bitmap encoded as a string and this is what we get back:



We expect a cross and get back a circle which is not what we want. This is because we don't have much body in this bitmap and potrace needs more pixels in order to give back a decent outline.

```
\startMPcode
    string s ; s := "010 111 010";
    draw lmt_potraced [
        bytes   = s,
        explode = true,
    ] ysized 2cm
      withpen pencircle scaled 4
      withcolor darkgreen ;
\stopMPcode
```

So, this time we 'explode' the bitmap, that is: we repeat every pixel three times in the horizontal and vertical direction. One can specify `nx` and `ny` but so far using different values doesn't help more than the magic threesome.

We're getting there but need to do a bit more.

```
\startMPcode
    string s ; s := "010 111 010";
    draw lmt_potraced [
        bytes     = s,
        threshold = 0.25,
        explode   = true,
    ] ysized 2cm
      withpen pencircle scaled 4
      withcolor darkgreen ;
\stopMPcode
```

Here we've set a threshold which will clip the paths in a range so that our case will get a better fit:

By now you will have noticed that we get back a path and this is an important feature of potrace: it returns a closed path expressed in lines and curves that one is supposed to fill. That result is converted

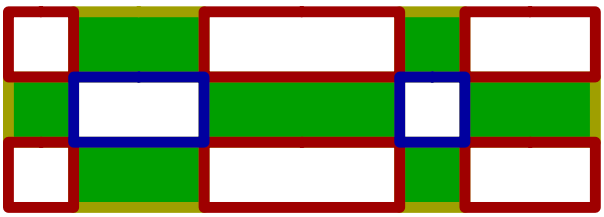
doi.org/10.47397/tb/45-1/tb139hagen-bitmaps

into a Lua table that we then can use for instance to generate a valid MetaPost path. We can do whatever we like with that path: draw or fill it for clipping. Natively, the library might return multiple paths but the user sees only one because we concatenate them which is a feature of the MetaPost library that comes with LuaMetaTEX.

Once we could do this it was no big deal to add support for filtering. After all, we have more than 0 and 1 characters available. Take this example, where we lay out the bitmap a bit differently, for clarity:

```
\startMPcode
  string s ; s := "
      211222122
      133111311
      211222122
  ";
  path p[] ;
  p[1] := lmt_potraced [
      bytes     = s,
      threshold = 0.25,
      explode   = true,
      value     = "1",
  ] ;
  p[2] := lmt_potraced [
      bytes     = s,
      threshold = 0.25,
      explode   = true,
      value     = "2",
  ] ;
  p[3] := lmt_potraced [
      bytes     = s,
      threshold = 0.25,
      explode   = true,
      value     = "3",
  ] ;

  fill p[1] withcolor darkgreen ;
  draw p[1] withcolor darkyellow ;
  draw p[2] withcolor darkred ;
  draw p[3] withcolor darkblue ;

  currentpicture
    := currentpicture xsized TextWidth ;
\stopMPcode
```

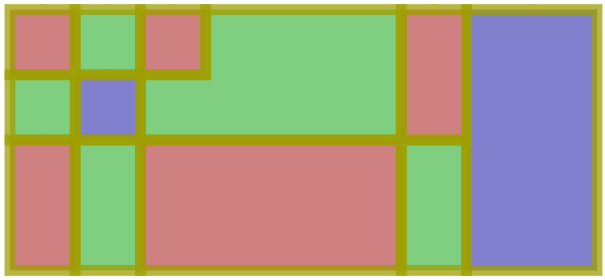
We save the paths so that we can use them multiple times, here for a draw and fill operation on the first path but you could scale, rotate, or manipulate the result before rendering it.



This example demonstrates that a user can define outlines using a bitmap specification and that the amount of code is rather small. At some point we might add a few more helpers that might reduce the amount of code even more.

```
\startMPcode
    string s ; s := "
        212111233
        131111233
        212222133
        212222133   ";
    path p[] ;
    p[1] := lmt_potraced [
        bytes     = s,
        threshold = 0.25,
        explode   = true, value = "1",
    ] ;
    p[2] := lmt_potraced [
        bytes     = s,
        threshold = 0.25,
        explode   = true, value = "2",
    ] ;
    p[3] := lmt_potraced [
        bytes     = s,
        threshold = 0.25,
        explode   = true, value = "3",
    ] ;
    linejoin := butt ;
    fill p[1] withcolor darkgreen
              withtransparency (1,.50) ;
    fill p[2] withcolor darkred
              withtransparency (1,.50) ;
    fill p[3] withcolor darkblue
              withtransparency (1,.50) ;
    draw p[1] withcolor darkyellow
              withtransparency (1,.75) ;
    draw p[2] withcolor darkyellow
              withtransparency (1,.75) ;
    draw p[3] withcolor darkyellow
              withtransparency (1,.75) ;
  currentpicture :=
    := currentpicture xsized TextWidth ;
\stopMPcode
```

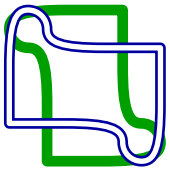
Because we have single paths we can safely apply properties like transparency, as shown below: crossing lines come out right instead of with accumulated transparent colors.



Sometimes you want to swap the rows and columns so we provide a feature for doing that:

```
\startMPcode
    string s[] ;

    s[1] := "1110 0110 0110 0111";
    s[2] := "1000 1111 1111 0001";

    draw lmt_potraced [
        bytes   = s[1],
        explode = true,
    ] ysized 2cm withcolor darkgreen
                withpen pencircle scaled 4 ;

    draw lmt_potraced [
        bytes   = s[2],
        explode = true,
    ] ysized 2cm withcolor darkblue
                withpen pencircle scaled 4 ;

    draw lmt_potraced [
        bytes   = s[1],
        explode = true,
        swap    = true,
    ] ysized 2cm withcolor white
                withpen pencircle scaled 2 ;
\stopMPcode
```

When one uses Lua input (as we will see later), one can do that when generating the bitmap. At any rate, this is what we get from the above:



The previous examples demonstrate that not much code is needed in order to achieve nice effects. It also illustrates that one needs to twist the mind a little and think of bitmap specifications as actually efficient outline definitions. One could argue that such rectangular shapes are easy to program in MetaPost anyway, and going via bitmaps is kind of strange. So, let's move on to a more attractive example.

3 How about fonts

In order to demonstrate building fonts we need a decent bitmap font and it happens that Don Knuth's 'Font36' is a good candidate (`https://erikdemaine.org/fonts/dissect`). We have discussed that one elsewhere and its usage can be found in the Metafun `threesix` library. We happily borrow the definitions from that library (actual source strings are all on one line):

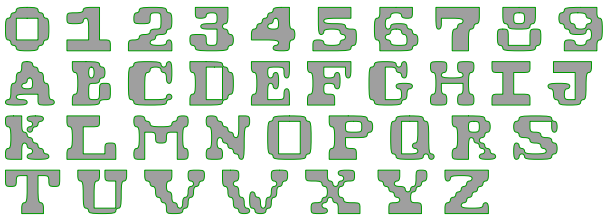
```
\startMPdefinitions
string dekthreesix[] ; path shapes[] ;

def DEK(expr n, b) =
    dekthreesix[utfnum(n)] := b ;
enddef ;

DEK("0", "00111100 01111110 11000011 11000011
          11000011 11000011 01111110 00111100");
...
DEK("Z", "11111111 10000111 00001110 00011100
          00111000 01110000 11100001 11111111");
\stopMPdefinitions
```

We use these definitions in the MetaPost code below. Helpers like `utfnum` are part of LuaMetafun and we use (named) colors defined at the ConTEXt end.

```
\startMPcode
def shapethem(expr first, last) =
    for i = utfnum(first) upto utfnum(last) :
        shapes[i] := lmt_potraced [
            bytes   = dekthreesix[i],
            explode = true,
          % threshold = .25, % more rectangular
            value   = "1",
        ] ;
    endfor ;
enddef ;

def drawthem(expr first, last, dx, dy) =
    numeric d ; d := 0 ;
    numeric f ; f := utfnum(first) ;
    numeric l ; l := utfnum(last) ;
    for i = f upto l :
        fill shapes[i] shifted (d,dy)
            withcolor "middlegray" ;
        draw shapes[i] shifted (d,dy)
             withcolor "darkgreen" ;
      % draw boundingbox shapes[i] shifted (d,dy);
        d := d + bbwidth(shapes[i]) + dx;
    endfor ;
enddef ;

shapethem("0","9") ;
shapethem("A","Z") ;

drawthem("0", "9", 10, -00) ;
drawthem("A", "J", 10, -30) ;
drawthem("K", "S", 10, -60) ;
drawthem("T", "Z", 10, -90) ;

currentpicture
  := currentpicture xsized TextWidth ;
\stopMPcode
```

Here we don't integrate it as a font but just show the characters as they come out, which hopefully is easier to understand. You can take a close look at the

bitmaps in order to see what we get rendered below. Setting a lower threshold will give more rectangular results.



Because we're not going to use this font for typesetting here, a simple example of a definition will do. We first load an already existing module so that we don't need to define the bitmap definitions; basically they are the same as above.

```
\useMPlibrary[threesix]

\startMPcalculation{simplefun}
    vardef ThreeSixPotraced
                  (expr code, spread, lift) =
        draw lmt_potraced [
            bytes   = code,
            explode = true,
            value   = "1",
        ] scaled (1/3) shifted (spread,lift) ;
    enddef ;
\stopMPcalculation
```

Next we register a Metafun font using similar trickery as in that module. There we define a few variants but that is not needed here.

```
\startluacode
  local utfbyte = utf.byte
  local f_code = string.formatters
           ['ThreeSixPotraced("%s",%s,%s);']

  function MP.registerthreesixpotraced(name)
    fonts.dropins.registerglyphs {
        name     = name,
        units    = 12,
        usecolor = true,
    }
    for u, data in table.sortedhash(MP.font36) do
        local ny     = 8
        local nx     = ((#data + 1) // ny) - 1
        local height = ny * 1.1 - 0.1
        local width  = nx * 1.1 - 0.1
        local spread = 0.9
        local lift   = 0.3
        fonts.dropins.registerglyph {
            category = name,
            unicode  = utfbyte(u),
            width    = width + spread,
            height   = height,
            code     = f_code(data,spread,lift),
        }
    end
  end
  MP.registerthreesixpotraced
    ("fontthreesixpotraced")
\stopluacode
```

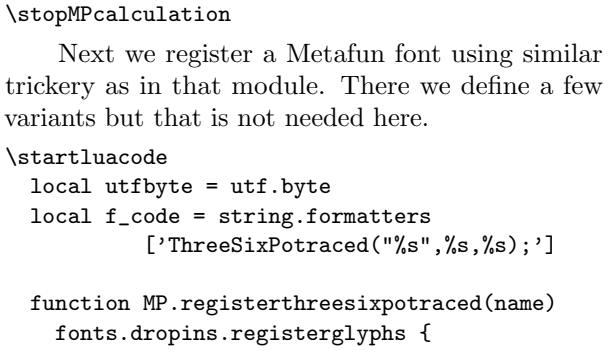
Finally we define a font feature that will hook the previous code into the font handler. There a Type 3 font will be constructed.

```
\definefontfeature
  [fontthreesixpotraced]
  [default]
  [metapost=fontthreesixpotraced,
   spacing=.375 plus .2 minus .1 extra .375]

\definefont[DEKFontP][Serif*fontthreesixpotraced]
```

We now show an example of usage (abridged). This font only has uppercase characters so one might consider duplicating these into the lowercase slots. Because we replace glyphs in the serif font used, we still have a complete font, albeit of mixed design.

```
\DEKFontP \WORD{\samplefile{knuth}}
```

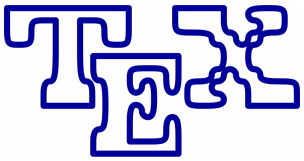
This gives us a rendering that is quite readable, especially when you consider how small the bitmaps are (the red bar is the overfull box marker):



Just in case Don Knuth runs into this example, we need to cheat a little here and redefine the TEX logo definition:

```
\protected\def\TeX
  {\dontleavehmode
   \begingroup
   T%
   \kern-.40\fontcharwd\font`T%
   \lower.45\fontcharht\font`X\hbox{E}%
   \kern-.15\fontcharwd\font`X%
   X%
   \endgroup}
```

A real font would have proper font kerns but if needed you can set that up using the OpenType feature plug in mechanism that ConTEXt provides. A font like this can also be colored:



4 External bitmaps

A convenient way to include traced images is to use the potrace command line tool. However, we can also include grayscale single-byte PNG-encoded images:

```
\startMPcode
  path p ; p := lmt_potraced [
      filename  = "mill.png",
      criterium = 100,
  ] ;

  fill last_potraced_bounds
       withcolor "middlegray" ;
  fill p withcolor "darkgreen" ;
  draw p withcolor "darkred"
         withpen pencircle scaled 1.5 ;
  setbounds currentpicture
            to last_potraced_bounds ;

  currentpicture
    := currentpicture xsized TextWidth ;
\stopMPcode
```

This is not the best image but the photograph happens to be part of the ConTEXt distribution so why not use it. You can of course apply a different criterion and overlay a subsequent trace in a different color. The result is shown in figure 1.

In addition to the `last_potraced_bounds` path variable we also have `last_potraced_width` and `last_potraced_height` numeric available.

5 Using Lua-generated bitmaps

It is tempting to see what can be done with a bitmap generated by Lua, but let's first give a simple example. Here we register a bitmap:

```
\startluacode
local s = [[
    00000001000000000000010000000
    00000010100000000000101000000
    00000100010000000001000100000
    00001000001000000010000010000
    00010000000100000100000001000
    00100000000010000100000000100
    01000000000001001000000000010
    10000000000000110000000000001
]]

potrace.setbitmap("mybitmap",s)
\stopluacode

\startMPcode
    path p ; p := lmt_potraced [
        stringname = "mybitmap",
    ] ;
    fill p ysized 2cm withcolor "darkblue" ;
\stopMPcode
```

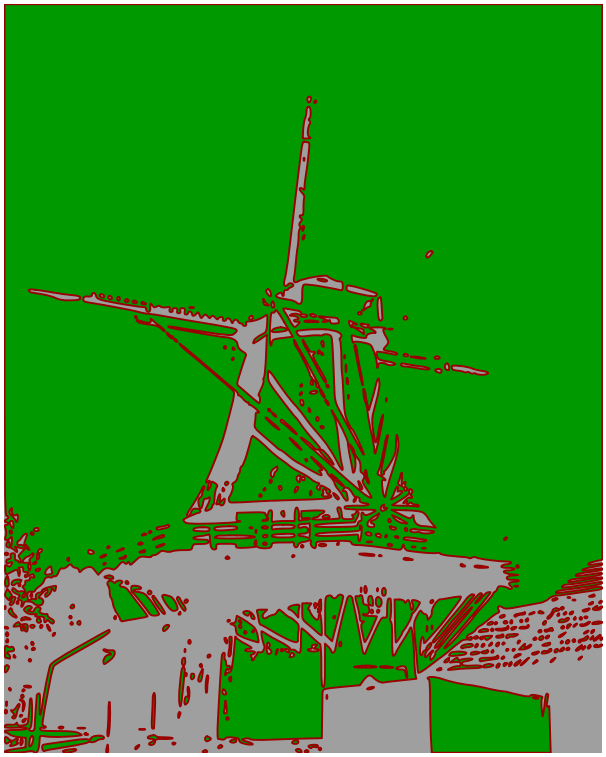
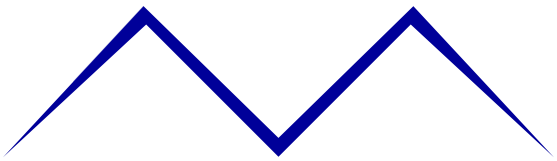


Figure 1: A traced PNG image

We could play with the parameters but what we get is an outline that is supposed to be filled:

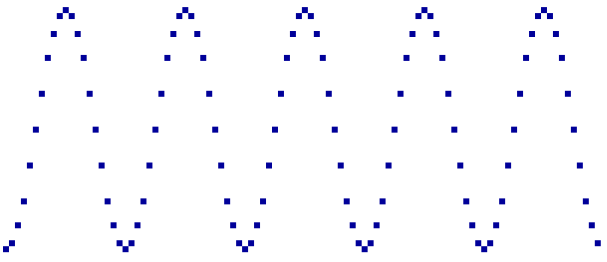


Next we create a bitmap from a dataset; in this case, we draw a function. Of course we could create some handy helpers to do this:

```
\startluacode
local d = table.setmetatableindex("table")

local t    = { }
local step = 100
local ymin = 0
local ymax = 0

local x    = 0
local dx   = 10*math.pi/step

for i=1,step do
    local y = math.round(math.cos(x)*20)
    x = x + dx
```

```
    if y > ymax then
        ymax = y
    end
    if y < ymin then
        ymin = y
    end
    t[i] = y
end

for i=1,step do
    for j=ymin, ymax do
        d[j][i] = '0'
    end
end

for i=1,step do
    d[t[i]][i] = '1'
end

for y=ymin,ymax do
    d[y] = table.concat(d[y])
end

potrace.setbitmap("mybitmap",
                  table.concat(d," ",ymin,ymax))
\stopluacode
\startMPcode
    path p ; p := lmt_potraced [
        stringname = "mybitmap",
        explode   = true,
      % tolerance = 0.5,
        threshold = 0,
      % optimize  = true,
    ] ;
    fill p withcolor "darkblue" ;
\stopMPcode
```

To what extent this is a useful application is to be decided:



Later we will see that it is less work if we stay in the first quadrant, using $(1, 1)$ as lower left corner. Here we explicitly need to provide `concat` the range because we have a negative $y_{\min}$.

It quickly gets more interesting when we use an intermediate bitmap for (a kind of) surface plots.

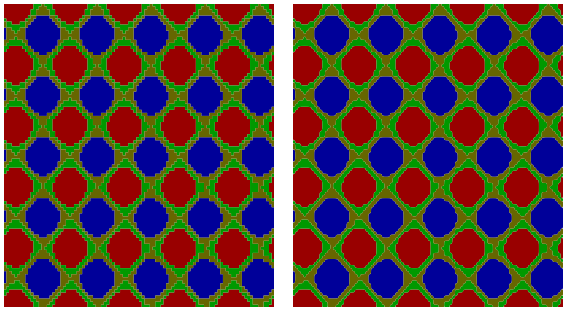
```
\startluacode
local sin, cos, pi = math.sin, math.cos, math.pi

local pp = pi/10

local function f(x,y)
    local z = sin(pp*x) + cos(pp*y)
    if z > 0.5 then
        return '1'
    elseif z > 0 then
        return '2'
    elseif z < -0.5 then
        return '3'
    else
        return'4'
    end
end
potrace.setbitmap("mybitmap",
                  potrace.contourplot(100,100,f))
\stopluacode
```

Here we use a helper that runs the given function over the maxima and collects the results in a bitmap. More such helpers will be provided when users come up with more demands.

```
\startMPcode
    fill lmt_potraced [
        stringname = "mybitmap",
        value      = "1",
        explode    = true,
        threshold  = 0.25,
      % tolerance  = 0.1,
      % threshold  = 1.0,
        optimize   = true,
    ] withcolor "darkred" ;

    fill lmt_potraced [
        stringname = "mybitmap",
        value      = "2",
        explode    = true,
        threshold  = 0.25,
      % tolerance  = 0.1,
      % threshold  = 1.0,
        optimize   = true,
    ] withcolor "darkgreen" ;

    fill lmt_potraced [
        stringname = "mybitmap",
        value      = "3",
        explode    = true,
        threshold  = 0.25,
      % tolerance  = 0.1,
      % threshold  = 1.0,
        optimize   = true,
    ] withcolor "darkblue" ;

    fill lmt_potraced [
        stringname = "mybitmap",
        value      = "4",
        explode    = true,
        threshold  = 0.25,
      % tolerance  = 0.1,
      % threshold  = 1.0,
```

```
        optimize   = true,
    ] withcolor "darkyellow" ;

    clip currentpicture to
      last_potraced_bounds enlarged -1;
    currentpicture := currentpicture
                  xysized (.45*TextWidth,4cm) ;
\stopMPcode
\startMPcode
    fill lmt_potraced [
        stringname = "mybitmap",
        value      = "1",
        explode    = true,
      % threshold  = 0.25,
        tolerance  = 0.1,
        threshold  = 1.0,
        optimize   = true,
    ] withcolor "darkred" ;

    fill lmt_potraced [
        stringname = "mybitmap",
        value      = "2",
        explode    = true,
      % threshold  = 0.25,
        tolerance  = 0.1,
        threshold  = 1.0,
        optimize   = true,
    ] withcolor "darkgreen" ;

    fill lmt_potraced [
        stringname = "mybitmap",
        value      = "3",
        explode    = true,
      % threshold  = 0.25,
        tolerance  = 0.1,
        threshold  = 1.0,
        optimize   = true,
    ] withcolor "darkblue" ;

    fill lmt_potraced [
        stringname = "mybitmap",
        value      = "4",
        explode    = true,
      % threshold  = 0.25,
        tolerance  = 0.1,
        threshold  = 1.0,
        optimize   = true,
    ] withcolor "darkyellow" ;

    clip currentpicture to
      last_potraced_bounds enlarged -1;
    currentpicture := currentpicture
                  xysized (.45*TextWidth,4cm) ;
\stopMPcode
```

Here we are only exploring the possibilities so there is no interface like we have with contour plots. One can imagine that we provide this as a plugin, which is not that hard but whether it eventually happens depends on user demand or rainy days. So to summarize: here we generate the bitmap, we call out to potrace for a vector representation, that gets fed into MetaPost and which gives us back a result that can be converted to PDF. Of course we could go directly from potrace output to PDF if we want to, but now we get full control over the final result. In case you wonder about performance: it compiles real fast!



Some work is needed to scale the image to the proportions that reflect the input ranges but because we have a vector image that does not affect the quality of the outcome. Here we also show the effects of `tolerance` which influences the optimizing and `threshold` that determines the accuracy (number of points). In the second rendering: we use the commented values that work quite well with this kind of more mathematical images.

6 Experimenting

You can use bitmaps as a design tool but it needs a little experimenting to get the idea. Take these examples:

We started with a simple X-like symbol:

```
\startMPcode
    fill lmt_potraced [ bytes = "
        00100000001
        01010000010
        10001000100
        01000101000
        00100010000
        00010101000
        00001000100
        00010100010
        00100010001
        01000001010
        10000000100
    " ] xsized 2cm
        withcolor darkgreen ;
```

```
\stopMPcode
```

Next we started filling the shape a bit and tried to make it less spiked:

```
\startMPcode
    fill lmt_potraced [ bytes = "
        00100000001
        01010000011
        10001000100
        01000101000
        00100010000
        00010101000
        00001000100
        00010100010
        00100010001
        11000001010
        10000000100
    " ] xsized 2cm
        withcolor darkblue ;
\stopMPcode
```

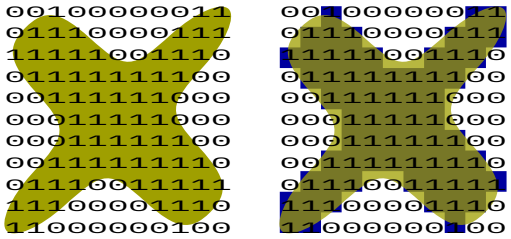
And finally we add even more ones to the bitmap. You need to fill a bitmap area in order to get an efficient fill.

```
\startMPcode
    fill lmt_potraced [ bytes = "
        00100000011
        01110000111
        11111001110
        01111111100
        00111111000
        00011111000
        00011111100
        00111111110
        01110011111
        11100001110
        11000000100
    " ] xsized 2cm
        withcolor darkyellow ;
\stopMPcode
```

If you're in doubt you can also render the bitmap, assuming that it has reasonable proportions.

```
\startMPcode
    string s ; s := "
        00100000011
        01110000111
        11111001110
        01111111100
        00111111000
        00011111000
        00011111100
        00111111110
        01110011111
        11100001110
        11000000100
    " ;
    fill lmt_potraced [ bytes = s ]
        xsized 3cm
        withcolor darkyellow ;
    draw lmt_potraced [ bytes = s,
                        alternative = "text" ]
        shifted (.25,.25)
        xsized 3cm ;
\stopMPcode
```

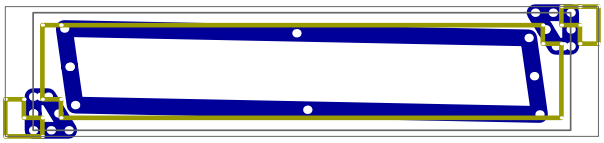
The article that we mentioned in the introduction explains how potrace looks at a bits in relation to its neighbors.



You can save some runtime (and coding) by using the start-stop wrappers that keep the (intermediate) potrace object available. This permits for instance showing the 'original' polygon that serves as basis for successive steps in the library towards to the final curve(s).

```
\startMPcode
    string s ; s :=
        "01111111111111111111111111111100
         11000000000000000000000000000110
         11000000000000000000000000000011
         11000000000000000000000000000011
         11000000000000000000000000000011
         01100000000000000000000000000011
         00111111111111111111111111111110";

    lmt_startpotraced [ bytes = s ] ;

        p := lmt_potraced [
            value     = "0",
            threshold = 0,
            tolerance = 0,
            optimize  = true,
        ] ;

        draw image (
            p := p shifted - center p ;
            draw p
                withpen pencircle scaled 1
                withcolor "darkblue" ;
            drawpoints p
                withpen pencircle scaled .5
                withcolor "white" ;
            draw boundingbox p
                withpen pencircle scaled .1
                withcolor "darkgray" ;
        ) ysized 3cm ;

        path p ; p := lmt_potraced [
            value   = "0",
            polygon = true,
```

```
    ] ;

    draw image (
        p := p shifted - center p ;
        draw p
            withpen pencircle scaled .25
            withcolor "middleyellow" ;
        drawpoints p
            withpen pencircle scaled .175
            withcolor "white" ;
        draw boundingbox p
            withpen pencircle scaled .05
            withcolor "darkgray" ;
    ) ysized 3cm ;

  lmt_stoppotraced ;
\stopMPcode
```



The above is just a bit of exploring the possibilities so eventually there will be a chapter on this in the LuaMetafun manual, because it is definitely fun to play with this in the perspective of MetaPost.

7 Contour plots

We end with showing how we can do rather nice contour plots and region plots with help of the potracer. Let us start with the latter type of graphics. In a recent math paper Mikael was counting nodal domains of Neumann eigenfunctions to the Laplace operator in a square. These eigenfunctions are built from cosines. One example is given by

$$\Psi(x,y) = \cos(8\pi x)\cos(3\pi y) + \cos(3\pi y)\cos(8\pi x) \ .$$

The related graphic of interest is to fill the part of the unit square where Ψ is positive. In the article, Wolfram Mathematica was used to produce this graphic, with its built-in function `RegionPlot`. The result was indeed satisfactory (Figure 2(a)).

If one looks closely at the graphics one will see that the filled regions are made up of a mesh. With potrace we instead receive one (!) path. To generate the corresponding graphic we first use Lua to define a function that takes a point as input and returns 1 if Ψ is positive at the point and 0 otherwise. We then use it to generate a 1000×1000 bitmap image of zeros and ones accordingly.

```
\startluacode
    local cos, pi = math.cos, math.pi

    local N       = 1000
    local pp      = pi/N
    local pp3     = 3 * pp
    local pp8     = 8 * pp
    local cospp8y = 0
    local cospp3y = 0

    local function f(x,y)
        if x == 1 then
            cospp8y = cos(pp8*y)
            cospp3y = cos(pp3*y)
        end
        local z = cos(pp8*x)*cospp3y
                  + cos(pp3*x)*cospp8y
        if z > 0 then
            return '1'
        else
            return '0'
        end
    end

    potrace.setbitmap("mybitmap",
                      potrace.contourplot(N,N,f))
\stopluacode
```

Once this is done, we can use the result in `lmt_potraced`.

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

    p := p xsized .9TextWidth ;
    fill p withcolor "darkred" ;
\stopMPcode
```

This results in Figure 2(b), which looks very similar to the one generated by Wolfram Mathematica. Note that `p` is one (disconnected) path.

We give one example of a contour plot. By a contour plot, here we mean a plot of the curve that is described as the solution to an equation $F(x,y) = 0$. With $F(x,y) = y - f(x)$ we realize that function graphs $y = f(x)$ provide a particular example, but we can also handle more complicated curves here, for example the unit circle ($F(x,y) = x^2 + y^2 - 1$).

Let us draw a part of the curve that goes under the name the Trisectrix of Maclaurin, a curve that can be used to trisect angles (named after Colin Maclaurin in 1742). We use the function

$$F(x,y) = 2x(x^2+y^2) - (3x^2 - y^2) \ .$$

Let us construct the path and draw it.

```
\startluacode
    local N   = 1000
    local xx  = 3/N
    local yy  = 3/N
```

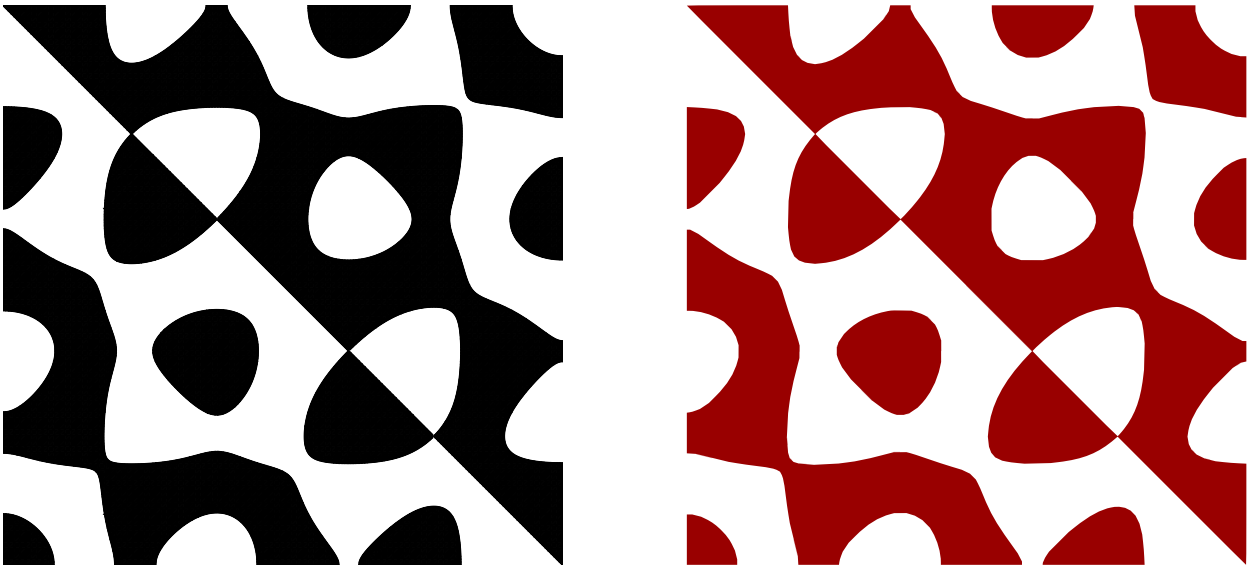


Figure 2: (a) A region plot generated by Wolfram Mathematica.
(b) The same, generated by potrace and Metafun.

```
    local function f(x,y)
        local x = xx*x - 1
        local y = yy*y - 1.5
        local z = 2*x*(x^2 + y^2) - (3*x^2 - y^2)
        if z > 0 then
            return '1'
        else
            return '0'
        end
    end

    potrace.setbitmap("mybitmap",
                      potrace.contourplot(N,N,f))
\stopluacode
\startMPcode
  path p ; p := lmt_potraced [
      stringname = "mybitmap",
      value      = "1",
      tolerance  = 0.1,
      threshold  = 1,
      optimize   = true,
  ] ;
  p := p xsized TextWidth ;
  draw p withcolor "darkred" ;
  drawpoints p withcolor "orange" ;
  drawpointlabels p ;
  currentpicture
    := currentpicture xsized min(8cm, TextWidth);
\stopMPcode
```

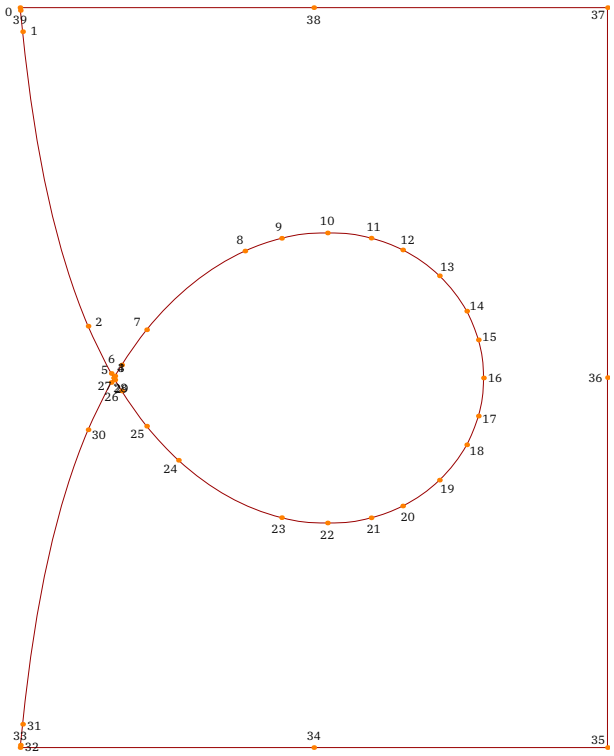


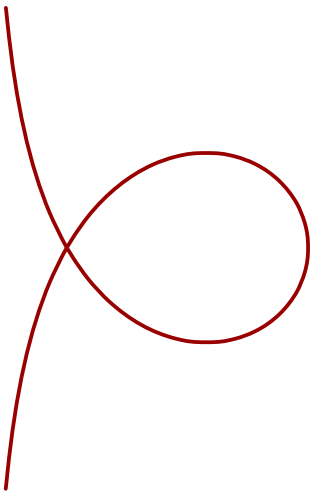
Figure 3: The Trisectrix of Maclaurin, with points and labels.

We also draw the points and the point labels. This way we can easily find out which part of the path to draw. In this case it seems that we need the first 34 points (Figure 3).

```
\startMPcode
```

```
    path p ; p := lmt_potraced [
        stringname = "mybitmap",
        value      = "1",
        tolerance  = 0.1,
        threshold  = 1,
        optimize   = true,
    ] ;
    p := subpath(0,33) of p ;
    p := p xsized 4cm ;
    draw p
        withcolor "darkred"
        withpen pencircle scaled .5mm ;
\stopMPcode
```

You need to keep the resolution (determined by the input) and therefore scaling in mind and choose the pen accordingly:

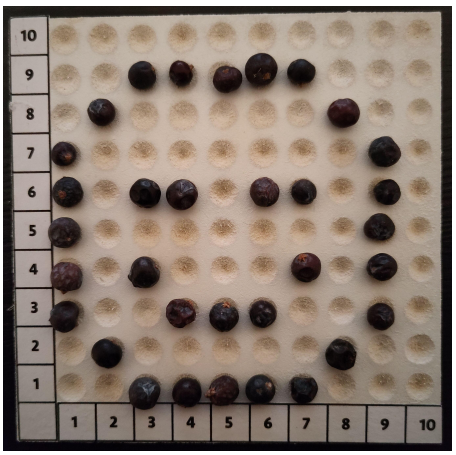


This method of drawing contour plots is efficient, and it gives, in contrast to some traditional methods, curves that look smooth, with relatively few points. It also has weaknesses, one of them being that the inequality $F(x, y) > 0$ does not always single out the curve $F(x, y) = 0$; it might be the case that the function F is positive on both sides of the curve $F(x, y) = 0$.

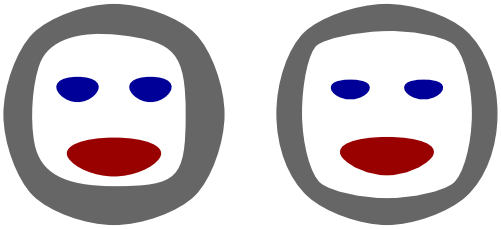
We invite the reader to be creative and play with this (to us) new toy. Have fun!

Coda

And, in the spirit of having fun, here are some final images. This first one is a photo of a "tool" that we came up with at the ConTEXt meeting. Willi Egger made a kit for the attendees, so there was the usual cutting and glueing involved. The dots are seeds.



Finally, here are dots laid out from an emoji that Mikael's children made when we were playing with this feature. The first image has the rendering of that input, while the second "explodes" the pixels in the x direction (so columns are duplicated), resulting in a more symmetric image.



◇ Hans Hagen
Pragma ADE

◇ Mikael P. Sundqvist
Department of Mathematics
Lund University
mickep (at) gmail dot com