Scrutinized Paths

A new path transformation in MetaFun

Mikael P. Sundqvist

In this article, I discuss a problem that occurred while trying to find direction-points of joined paths in MetaPost. We found that when joining two paths where the final point of the first path is the same as, or at least very close to, the first point of the second path, numerical problems might appear. Finally, we present different solutions that appeared on the ConTEXt mailing list on how to avoid the numerical problem, the most generic ones of which work by sanitizing the joined graph. In fact, this resulted in a new path transformation called `scrutinized`.

Curves of constant width and Barbier's theorem

A curve of constant width in the plane is a simple closed convex curve for which every pair of parallel supporting lines are equally distant apart. This distance is called the diameter of the figure. The simplest such curve is the circle. In figure 1 we show some other examples. Once the dashed lines are given, the curves can be constructed with a compass, with the needle point located at the blue dots. Note that the curve in (a) has corners while the other two curves are smooth.

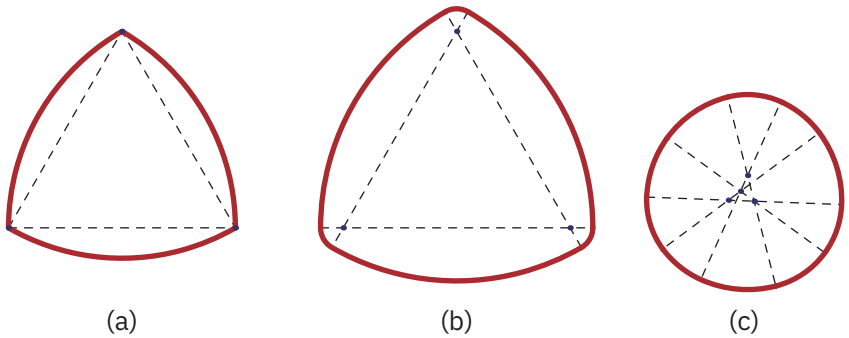

Figure 1.

It is well known that the circumference of a circle with diameter d equals πd. A theorem by Barbier states that this is true for any curve of constant width.

Theorem (Barbier) *Any curve of constant width d has length πd.*

There are several different proofs of this property. I recently had the pleasure to supervise a group of students in a project course in mathematical communication at Lund University, where they were supposed to understand and write about these proofs. One of the proofs of Barbier's theorem is based on the so-called Minkowski sums. Given two sets A and B of points in the plane, the *Minkowski sum* $A + B$ is defined to be the set:

$$A + B = \{a + b \mid a \in A,\ b \in B\}\,.$$

It was while I was presenting to the students an idea about how to possibly draw those figures that I myself encountered a problem with MetaPost, to which I wrote to the ConTEXt mailing list about. It is the result of this inquiry that I will now discuss below.

The problematic figure

It is a fact that if we start with a figure A of constant width, and add a copy B of it that is rotated 180° around some point O, then the Minkowski sum $A + B$ will be a disk. Moreover, if the width of A (and thus also of B) is d, then the diameter of the disk will be $2d$.

In fact, more is true. If we walk around the boundaries of A and B (in an anti-clockwise direction, say), then the point on the circle for which the tangent has a given direction is given by adding the position vectors of the two points on A and B where their tangents have the same direction. Here we have considered the point O as origin. This property is illustrated in figure 2, and this is the image that we wanted to draw.

The idea of drawing such a figure was to first construct the red curves, and then, thanks to the property explained above, loop through the angles and find the corresponding points on the curves for which the direction is the correct `directionpoint`.

Given that the two curves of constant width were constructed as `p[1]` and `p[2]`, we tried to construct their sum `p[3]` as:

```
p[3] := for phi=0 step 30 until 360:
          ((directionpoint dir(phi) of p[1])
           shifted
           (directionpoint dir(phi) of p[2]))
           ..
        endfor cycle;
```

However, instead of giving us the expected result of figure 2, we ended up with the strange result in figure 3.

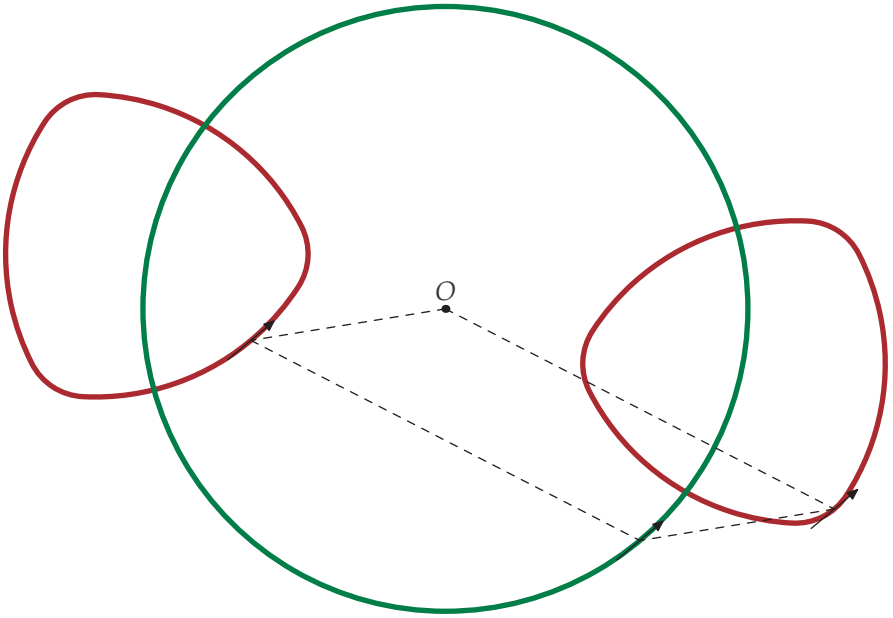

Figure 2.

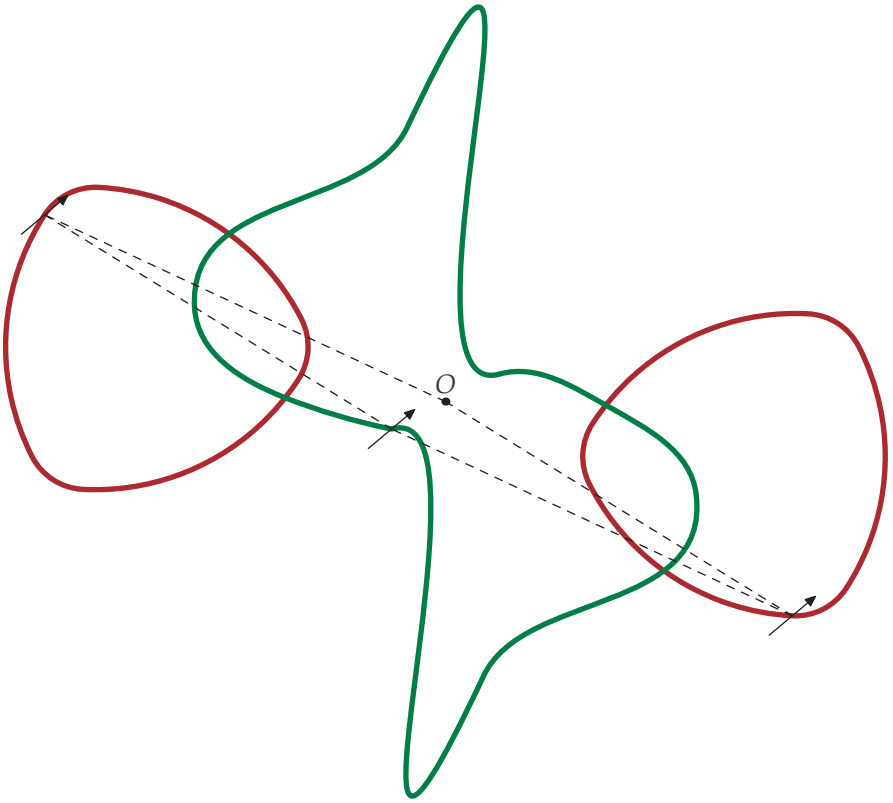

Figure 3.

Debugging

Looking at figure 3, we can see that the problem seems to be that the arrows are not tangential to the curves. Could it be that `directionpoint` is responsible for the wrong result? In the MetaPost manual we can read that `directionpoint` is supposed to give the *first* point on a path where a given direction is achieved.

I got a quick reply on the mailing list from Taco Hoekwater, indicating that the problem is not really with the macro `directionpoint` *per se*, but rather with the construction of the curve of constant width, so let us focus on that for a while.

```
\startMPcode{doublefun}
u:=0.5cm;
path cl,cs,p[];
z0 = (0,6/sqrt(3)*u);
z1 = z0 rotated 120;
cs := (fullcircle scaled 16u) shifted z1;
cs := cs cutafter point 1/6 along cs;
cl := (fullcircle scaled 4u) shifted z0;
cl := cl cutbefore point 1/6 along cl
         cutafter point 2/6 along cl;

p[0] = cs ..
       cl ..
       (cs rotated 120) ..
       (cl rotated 120) ..
       (cs rotated 240) ..
       (cl rotated 240) ..
       cycle;

draw p[0] withpen pencircle scaled 2bp withcolor darkred;
drawpoints p[0];
drawpointlabels p[0];
\stopMPcode
```

The resulting image (figure 4) does not look strange. We glued several arcs together to form a path. We can see that the path `p[0]` has a pair of points that are very close to each other near where the two arcs are joined. Mathematically, these two points should be the same, but they are not. This is where the problem lies. As Taco noticed, the paths `cs` and `cl` are given as below.

```
cs := (141.73224999999996,-49.097491614210789) ..
      (75.312386775380347,111.25424516116959) ..
      (28.347427842053655,147.2925755432174);
cl := (28.346108531095332,147.29283827977969) ..
      (0,154.88788322842163) ..
      (-28.346108531095332,147.29283827977969);
```

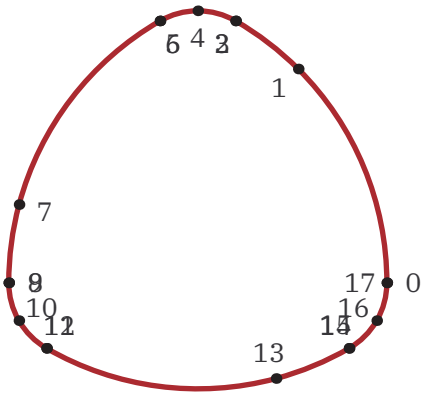

Figure 4.

Note that the last point of `cs` is *almost* the same as the first point of `c1`. In figure 5, we have drawn the curve `p[0]` together with its control points (with the first point slightly bigger). We have also tried to add tangent vectors pointing in directions $n \times 30°$ with the code:

```
for phi=0 step 30 until 330:
  drawarrow(((-u,0)--(u,0)) rotated phi)
            shifted (directionpoint dir(phi) of p[0]);
  freelabel("$" & decimal phi & "$\unit{degree}",
            directionpoint dir(phi) of p[0],
            origin);
endfor;
```

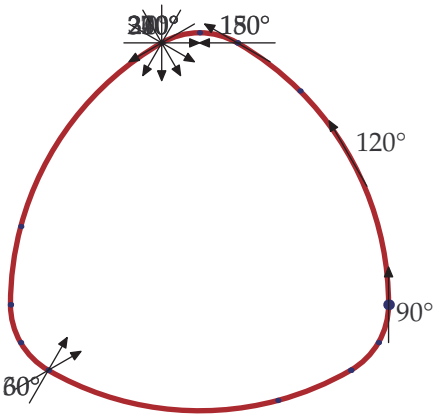

Figure 5.

As you can see, the tangent vectors are not placed correctly. It seems that they tend to get stuck exactly at the points where the different paths are glued together. The problem is simply that these points are *too close to each other* and this confuses `directionpoint`, which apparently interprets the curve to have tangents that it does not have, or at least should not have (remember that `directionpoint` returns the

first point of the path where it finds a point it interprets as having the correct direction). Now that the problem is identified, we can start to look for solutions.

Solutions

Amusingly, when Taco had identified the problem, three solutions appeared more or less immediately. I will present them here, and I will start with the simplest but least general one that I came up with myself, and then move on to the more generic solutions given by Taco Hoekwater and Hans Hagen.

A simple but naïve solution

The first solution that came to my mind was to shorten the paths `cs` and `cl` slightly. This would move the problematic points a bit further away from each other, which would help `directionpoint` to interpret the directions correctly. Thus, I changed my definitions of `cs` and `cl` into the following:

```
cl := (fullcircle scaled 4u) shifted z0;
cl := cl cutbefore point (1/6+epsilon) along cl
        cutafter point (2/6-epsilon) along cl;
cs := (fullcircle scaled 16u) shifted z1;
cs := cs cutafter point (1/6-epsilon) along cs;
```

Note that we use an `epsilon` here to cut the circles just a bit more. The constant `epsilon` is defined to be $1/256/256$ (sic!). Apparently this was enough in this case. These are indeed the definitions used to produce figure 2.

In figure 6, we have implemented this change for the paths `cs` and `cl`, and we noticed that the curve `p[0]` still had 18 points (left-hand image), but now the almost duplicate points are sufficiently far away from each other not to confuse `directionpoint` (right-hand image).

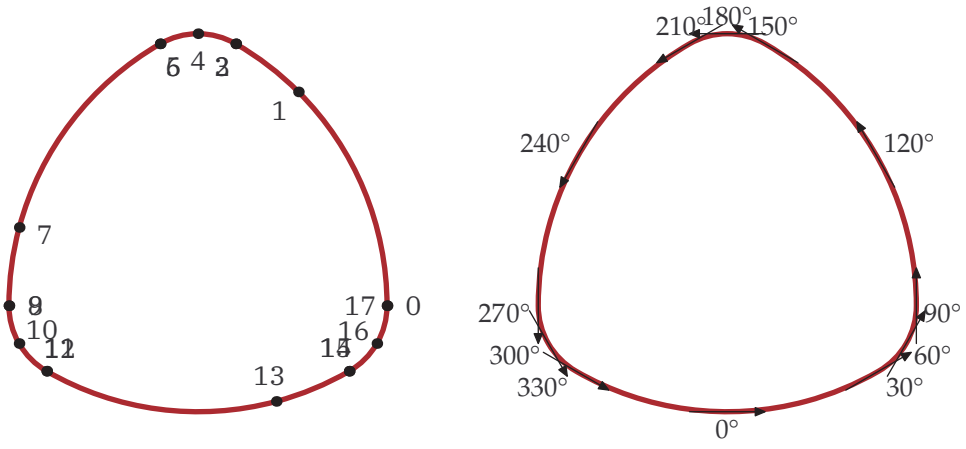

Figure 6.

A more general solution by Taco Hoekwater

Both Taco and Hans had the idea of cleaning the joined paths by removing the superfluous points that are almost exactly at the same place. Taco's solution reads like this:

```
\startMPdefinitions{doublefun}
def clean_path(suffix p) =
  begingroup;
  save q,precontrols,postcontrols,points,i,j;
  pair precontrols[],postcontrols[],points[] ;
  j := 0;
  for i = 0 upto length p:
    if arclength (point i of p--point i+1 of p) > 0.01:
       points[j] := point i of p;
       postcontrols[j] := postcontrol i of p;
       precontrols[j+1] := precontrol i+1 of p;
       j := j + 1;
    fi
  endfor;
  if arclength (point 0 of p--point length p of p) < 0.01:
     j := j - 1;
  fi
  p := for i=0 upto j-1: points[i] ..
          controls postcontrols[i] and precontrols[i+1] ..
       endfor cycle;
  endgroup;
enddef;
\stopMPdefinitions
```

Now it is a matter of adding:

```
clean_path(p[0]);
```

after the definition of `p[0]`. We draw the resulting curve twice in figure 7. To the left we show its point, and we notice that the number of points of the path has decreased from 18 to 12. The six near duplicates were indeed merged. To the right we have also drawn some tangent vectors and, as expected, the correct points are now found by `directionpoint`.

The implemented solution by Hans Hagen

Only minutes after Taco's solution appeared on the mailing list, Hans announced a solution in the form of a path modifier implemented in Lua. After some tuning, it looks like this:

```
\startluacode
registerscript("scrutinized", function()
    local pth  = scanpath()
```

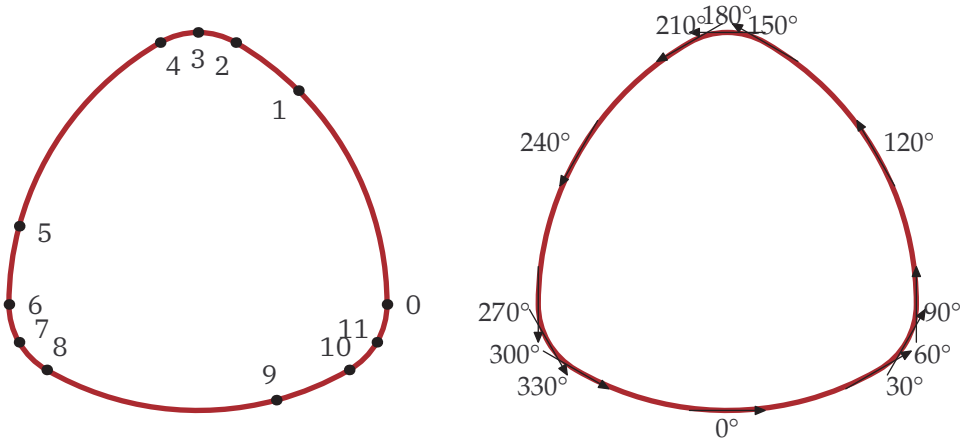

Figure 7.

```
    local d    = 1/10^scannumeric() -- decimals
    local p1   = pth[1]
    local x1   = p1[1]
    local y1   = p1[2]
    local res  = { pth[1] }
    local r    = 1
    for i=2,#pth do
        local pi = pth[i]
        x2 = pi[1]
        y2 = pi[2]
        if abs(x1-x2) > d or abs(y1-y2) > d then
            r = r + 1 res[r] = pi
            x1 = x2
            y1 = y2
        else
            res[r][5] = pi[5]
            res[r][6] = pi[6]
        end
    end
    if pth.cycle then
        res.cycle = true
        if abs(x1-p1[1]) > d or abs(y1-p1[2]) > d then
            -- keep
        else
            res[r] = nil
        end
    end
    injectpath(res)
end)
\stopluacode
```

It is defined in MetaFun like this:

```
\startMPdefinitions{doublefun}
newscriptindex mfid_scrutinized ;
mfid_scrutinized := scriptindex "scrutinized" ;

primarydef p scrutinized  n =
  runscript mfid_scrutinized p n
enddef;
\stopMPdefinitions
```

With these definitions, we need to redefine the path `p[0]` with for example:

```
p[0] := p[0] scrutinized 3;
```

where 3 is chosen to compare the points up to three decimal places. We see the curves redrawn with this solution in figure 8. Again, the number of points in the path has decreased from 18 to 12.

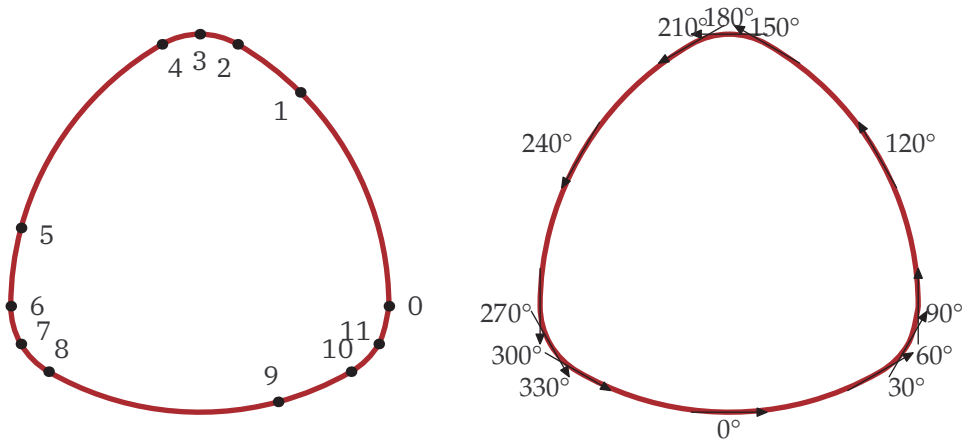

Figure 8.

Acknowledgments

I would like to thank Taco Hoekwater for suggesting to write this article, and Hans Hagen for giving me the support I needed during my writing. As usual, it was great fun to play with MetaFun.