# The art of Maps proofreading

When Taco had to turn Yuri's article into the Maps format the problem was that the tutorial about dice was using a MetaFont and as the Maps prefers outlines it made sense to go that way. So, as follow up Taco made the three dimensional dice into a font and wondered if we should add a typescript to the distribution. And who can deny Taco. especially when also Frans wants to use these shapes. While pondering this I realized that we already had dice in ConTEXt so I took a look at the MetaFont code to see what magic was needed for going 3D. The file used by Yuri and Taco is `DICE3D.MF` by Thomas A. Heim and dates from 1998.

The OpenType font that Taco made shows some inaccuracies with respect to the way the three sides are connected. At first I though that there was some issue with the transform but it more looks like it is in the definitions of the paths. I won't go into details but using scaled `fullsquare`'s works well so that is what we do here.

In the code below we start with the simple dice shapes. We just define the six variants as macros. Because we later will reuse the dots we save them in a picture list. The definitions have been simplified a bit because in the MetaFun module we also define the reversed variants two and three as well as dominos.

```
\startMPcalculation{simplefun}
  picture DiceDots[] ;

  pickup pencircle scaled 3/2 ;

  DiceDots[ 1] := image ( draw(4,4) ; ) ;
  DiceDots[ 2] := image ( draw(2,6) ; draw(6,2) ; ) ;
  DiceDots[ 3] := image ( draw(2,6) ; draw(4,4) ; draw(6,2) ; ) ;
  DiceDots[ 4] := image ( draw(2,6) ; draw(6,6) ; draw(2,2) ; draw(6,2) ; ) ;
  DiceDots[ 5] := image ( draw(2,6) ; draw(6,6) ; draw(4,4) ; draw(2,2) ; draw(6,2) ; ) ;
  DiceDots[ 6] := image ( draw(2,6) ; draw(6,6) ; draw(2,4) ; draw(6,4) ; draw(2,2) ;
                                                                          draw(6,2) ; ) ;

  def DiceFrame =
    pickup pencircle scaled 1/2 ;
    draw unitsquare scaled 8 ;
  enddef ;

  vardef DiceOne   = DiceFrame ; draw DiceDots[1] ; enddef ;
  vardef DiceTwo   = DiceFrame ; draw DiceDots[2] ; enddef ;
  vardef DiceThree = DiceFrame ; draw DiceDots[3] ; enddef ;
  vardef DiceFour  = DiceFrame ; draw DiceDots[4] ; enddef ;
  vardef DiceFive  = DiceFrame ; draw DiceDots[5] ; enddef ;
  vardef DiceSix   = DiceFrame ; draw DiceDots[6] ; enddef ;

  vardef DiceBad =
    DiceFrame ; draw (1,7) -- (7,1) ; draw (1,1) -- (7,7) ;
  enddef ;
\stopMPcalculation
```

Next we define a Type3 font. The `lmt_` prefix is used for a collection of macros in LuaMetaFun. It is an example of how we enhance the user interface with parsers written in Lua; these sort of extend the MetaPost syntax. These glyphs all have the

same dimensions. A Type3 font consists of bitmap or outline drawing operators and with some Lua magic we can create these in LuaTEX and LuaMetaTEX. One just has to make sure to plug them into the pdf backend.

```
\startMPcalculation{simplefun}

  lmt_registerglyphs [
    name     = "dice",
    units    = 12,
    width    = 8,
    height   = 8,
    depth    = 0,
    usecolor = true,
  ] ;

  lmt_registerglyph [ category = "dice", unicode = "0x2680", code = "DiceOne;" ] ;
  lmt_registerglyph [ category = "dice", unicode = "0x2681", code = "DiceTwo;" ] ;
  lmt_registerglyph [ category = "dice", unicode = "0x2682", code = "DiceThree;" ] ;
  lmt_registerglyph [ category = "dice", unicode = "0x2683", code = "DiceFour;" ] ;
  lmt_registerglyph [ category = "dice", unicode = "0x2684", code = "DiceFive;" ] ;
  lmt_registerglyph [ category = "dice", unicode = "0x2685", code = "DiceSix;" ] ;

  lmt_registerglyph [ category = "dice", private = "invaliddice", code = "DiceBad;" ] ;

\stopMPcalculation
```

We now will add the three dimensional variants for which we need a few transformations that we borrow from the MetaFont original. It is the only code we had to take but it is also the most magical.

```
\startMPcalculation{simplefun}
  transform t[] ; numeric r ; r := sqrt(1/4) ;

  hide((0,0) transformed t1 = (0,0)) ;
  hide((1,0) transformed t1 = (r,r)) ;
  hide((0,1) transformed t1 = (0,1)) ;

  hide((0,0) transformed t2 = (0,0)) ;
  hide((1,0) transformed t2 = (1,0)) ;
  hide((0,1) transformed t2 = (r,r)) ;

  t3 := t1 shifted (8,0) ; % front to right side
  t4 := t2 shifted (0,8) ; % front to top
\stopMPcalculation
```

Next we define the extra variants. The list of combinations (of three digits) come from the mentioned MetaFont file:

```
\startMPcalculation{simplefun}
  vardef Diced(expr a, b, c) =
    draw image (
      pickup pencircle scaled 1/2 ;
      draw image (
        nodraw unitsquare scaled 8 transformed t4  ;
        nodraw unitsquare scaled 8 transformed t3  ;
        nodraw unitsquare scaled 8 ;
        dodraw unitsquare scaled 8 ;
      ) ;
      draw DiceDots[a]                ;
      draw DiceDots[b] transformed t3 ;
      draw DiceDots[c] transformed t4 ;
```

```
    ) ;
  enddef ;
\stopMPcalculation
```

We use this macro when we register the shapes. The Unicode's are of course wrong but we don't care too much about them here. We could have used private slots.

```
\startMPcalculation{simplefun}
  lmt_registerglyph [ category = "dice", unicode = "123", code = "Diced(1,2,3);", width = 12, height = 12 ] ;
  lmt_registerglyph [ category = "dice", unicode = "135", code = "Diced(1,3,5);", width = 12, height = 12 ] ;
  lmt_registerglyph [ category = "dice", unicode = "142", code = "Diced(1,4,2);", width = 12, height = 12 ] ;
  lmt_registerglyph [ category = "dice", unicode = "154", code = "Diced(1,5,4);", width = 12, height = 12 ] ;
  lmt_registerglyph [ category = "dice", unicode = "214", code = "Diced(2,1,4);", width = 12, height = 12 ] ;
  lmt_registerglyph [ category = "dice", unicode = "231", code = "Diced(2,3,1);", width = 12, height = 12 ] ;
  lmt_registerglyph [ category = "dice", unicode = "246", code = "Diced(2,4,6);", width = 12, height = 12 ] ;
  lmt_registerglyph [ category = "dice", unicode = "263", code = "Diced(2,6,3);", width = 12, height = 12 ] ;
  lmt_registerglyph [ category = "dice", unicode = "312", code = "Diced(3,1,2);", width = 12, height = 12 ] ;
  lmt_registerglyph [ category = "dice", unicode = "326", code = "Diced(3,2,6);", width = 12, height = 12 ] ;
  lmt_registerglyph [ category = "dice", unicode = "351", code = "Diced(3,5,1);", width = 12, height = 12 ] ;
  lmt_registerglyph [ category = "dice", unicode = "365", code = "Diced(3,6,5);", width = 12, height = 12 ] ;
  lmt_registerglyph [ category = "dice", unicode = "415", code = "Diced(4,1,5);", width = 12, height = 12 ] ;
  lmt_registerglyph [ category = "dice", unicode = "421", code = "Diced(4,2,1);", width = 12, height = 12 ] ;
  lmt_registerglyph [ category = "dice", unicode = "456", code = "Diced(4,5,6);", width = 12, height = 12 ] ;
  lmt_registerglyph [ category = "dice", unicode = "462", code = "Diced(4,6,2);", width = 12, height = 12 ] ;
  lmt_registerglyph [ category = "dice", unicode = "513", code = "Diced(5,1,3);", width = 12, height = 12 ] ;
  lmt_registerglyph [ category = "dice", unicode = "536", code = "Diced(5,3,6);", width = 12, height = 12 ] ;
  lmt_registerglyph [ category = "dice", unicode = "541", code = "Diced(5,4,1);", width = 12, height = 12 ] ;
  lmt_registerglyph [ category = "dice", unicode = "564", code = "Diced(5,6,4);", width = 12, height = 12 ] ;
  lmt_registerglyph [ category = "dice", unicode = "624", code = "Diced(6,2,4);", width = 12, height = 12 ] ;
  lmt_registerglyph [ category = "dice", unicode = "632", code = "Diced(6,3,2);", width = 12, height = 12 ] ;
  lmt_registerglyph [ category = "dice", unicode = "645", code = "Diced(6,4,5);", width = 12, height = 12 ] ;
  lmt_registerglyph [ category = "dice", unicode = "653", code = "Diced(6,5,3);", width = 12, height = 12 ] ;
\stopMPcalculation
```

At the ConTEXt (read: TEX) end we define three font features. The digits features will map digits onto dice and the ligatures, when enabled, come from sequences of thee digits. The `metapost` feature pushes the graphics into the font instead of already present characters.

```
\definefontfeature
  [dice:normal] % no reverse in this example
  [default]
  [metapost={category=dice}]

\definefontfeature
  [dice:digits]
  [dice:digits=yes]

\definefontfeature
  [dice:three]
  [dice:three=yes]
```

The `dice:digits` and `dice:three` features are implemented as follows. The ligature definitions have to come before the digit remapping because we process features in order.

```
\startluacode
  fonts.handlers.otf.addfeature("dice:three", {
    type      = "ligature",
```

```
    order     = { "dice:three" },
    nocheck   = true,
    data      = {
      [123] = { 0x31, 0x32, 0x33 }, [135] = { 0x31, 0x33, 0x35 },
      [142] = { 0x31, 0x34, 0x32 }, [154] = { 0x31, 0x35, 0x34 },
      [214] = { 0x32, 0x31, 0x34 }, [231] = { 0x32, 0x33, 0x31 },
      [246] = { 0x32, 0x34, 0x36 }, [263] = { 0x32, 0x36, 0x33 },
      [312] = { 0x33, 0x31, 0x32 }, [326] = { 0x33, 0x32, 0x36 },
      [351] = { 0x33, 0x35, 0x31 }, [365] = { 0x33, 0x36, 0x35 },
      [415] = { 0x34, 0x31, 0x35 }, [421] = { 0x34, 0x32, 0x31 },
      [456] = { 0x34, 0x35, 0x36 }, [462] = { 0x34, 0x36, 0x32 },
      [513] = { 0x35, 0x31, 0x33 }, [536] = { 0x35, 0x33, 0x36 },
      [541] = { 0x35, 0x34, 0x31 }, [564] = { 0x35, 0x36, 0x34 },
      [624] = { 0x36, 0x32, 0x34 }, [632] = { 0x36, 0x33, 0x32 },
      [645] = { 0x36, 0x34, 0x35 }, [653] = { 0x36, 0x35, 0x33 },
    }
  } )

  fonts.handlers.otf.addfeature("dice:digits", {
    type      = "substitution",
    order     = { "dice:digits" },
    nocheck   = true,
    data      = {
      [0x30] = "invaliddice",
      [0x31] = 0x2680, [0x32] = 0x2681, [0x33] = 0x2682,
      [0x34] = 0x2683, [0x35] = 0x2684, [0x36] = 0x2685,
      [0x37] = "invaliddice",
      [0x38] = "invaliddice",
      [0x39] = "invaliddice",
    },
  } )
\stopluacode
```

We now can use these fonts so we define a few font instances with different features:

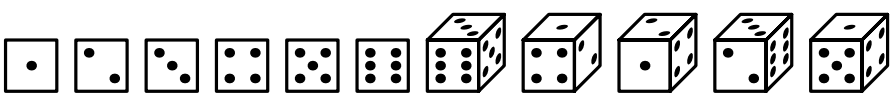
```
\definefont[DiceN][Serif*dice:normal]
\definefont[DiceD][Serif*dice:normal,dice:digits]
\definefont[DiceT][Serif*dice:normal,dice:three,dice:digits]
```

The next few lines give simple two dimensional dice:

```
\DiceN \dostepwiserecurse{"2680}{"2685}{1}{\char#1\quad}%
\DiceD 2\quad5\quad3\quad0
```

As we can see in the next character run, a nice aspect is that the digits are also transformed so there is some perspective in it.

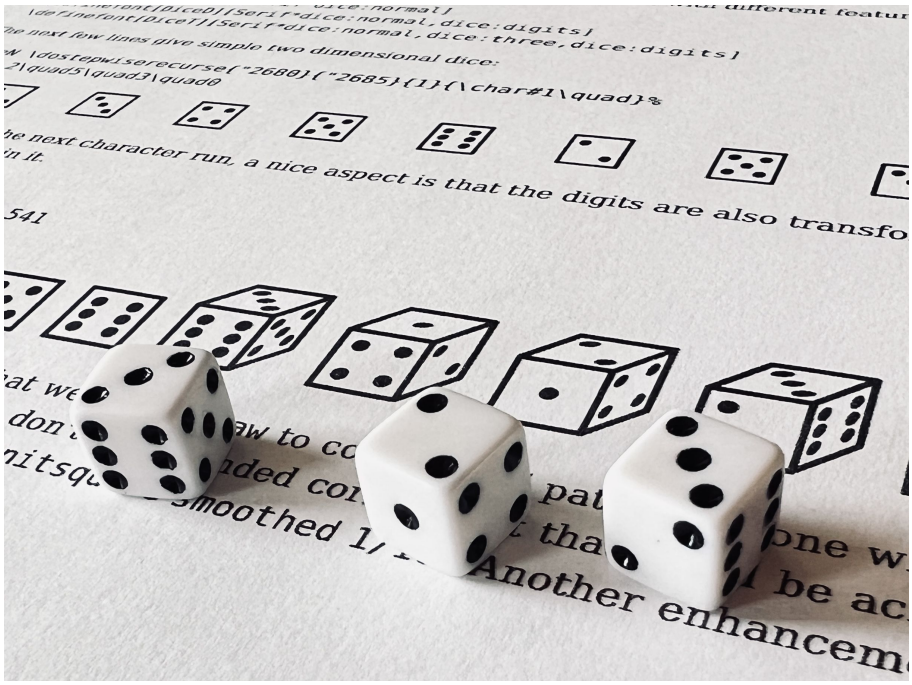
```
\DiceT 1 2 3 4 5 6
\DiceT 653 421 142 263 541
```



One of the probably unseen details is that we use `nodraw` to combine all paths into one which gives nicer shapes. Contrary to the original we don't use rounded corners but that could be achieved by replacing the `unitsquares` by for instance `unitsquare smoothed 1/10`. Another enhancement could be filled dice with white dots.

We come now to the title of this article. When a Maps is being composed, the workflow is as follows. An author send an article, and when it's in pdf format, Frans will read it carefully and feedback issues. When all is okay, Taco kicks in and turns it into the Maps format which involves looking at page breaks, scaling of images, checking fonts and color etc. Then Frans again takes a look at it. It cannot be denied that in my personal case I actually rely on Frans to prevent me from mistakes.

How serious the Maps proofreading is done is demonstrated in the next image. It shows us that not only an article gets printed but that there is some handy work involved too. In this case Frans took real dice as reference. It shows how I get bitten by not showing the complete implementation here.




These show the mirrored variants of two and three and if you really want a consistent set of dice you need to have some use these variants. I will not do that here because we then also have to discuss influencing the variants. It makes sense to use Unicode modifiers to control this. The dice definition in the ConTEXt module actually also have these mirrored shapes and a `dice:reverse` feature:

```
\startMPcalculation{simplefun}
  DiceDots[-2] := image ( draw(6,6) ; draw(2,2) ; ) ;
  DiceDots[-3] := image ( draw(6,6) ; draw(4,4) ; draw(2,2) ; ) ;
\stopMPcalculation
```

But first I have to find me some dice or borrow Frans his reference set. We leave that for the ConTEXt meeting later this year.

Hans Hagen