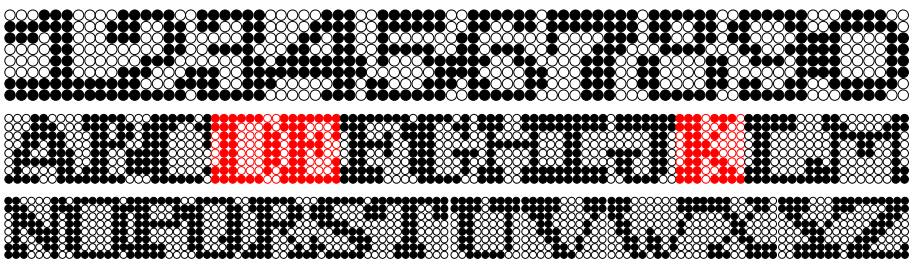
ThreeSix

Don Knuths first colorfont?

In the process of reaching completion and perfection Don Knuth occasionally posts links to upcoming parts of the TAOCP series on his web pages. Now, I admit that much is way beyond me but I do understand (and like) the graphics and I know that Don uses MetaPost. The next example code is just a proof of concept but might eventually become a decent module (with helpers) for making (runtime) fonts. After all, we need to adapt to current developments and TEXies are always willing to adapt and experiment. This chapter was written at the same time as the previous one on Type3 fonts so you might want to read that first.

The font explored here is `FONT36`, used in "A potpourri of puzzles" and flagged as "a special font designed for dissection puzzles" (in fascicle 9b for Volume 4). Playing with and visualizing for me often works better than formulas, which then distracts me from the original purpose, but let's have a closer look anyway.



The font has a fixed maximum height of 8 quantities. There is no depth in the characters. Some characters are wider. In this example we use a tight bounding box. In ConTEXt speak this font is just a regular font but with a special feature enabled.

```
\definefontfeature
  [fontthreesix]
  [default]
  [metapost=fontthreesix]

\definefont[DEKFontA][Serif*fontthreesix]
```

After this the `\DEKFontA` command will set this font as current font. The definition mentions `Serif` as font name. In ConTEXt this name will resolve in the currently defined Serif, so when your document uses Latin Modern that will be the one. The `fontthreesix` will make this instance use that feature set, and the feature definition has the defaults as parent (so we get kerning, ligatures, etc.) but as extra feature also `metapost`. This means that the new glyphs that are about to be defined will actually be injected in the `Serif`! We will replace characters in that instance. So, the following:

```
This font is used in \quotation {The Art Of Computer Programming} by
Don Knuth, not in volume~1, 2 or~3, but in number~4!
```

comes out as:

> This font is used in “The ■rt ■f ■omputer ■rogramming” by ■on ■nuth, not in volume ■, ■ or ■, but in number ■!

But that doesn’t look too good, so we will tweak the font a bit:

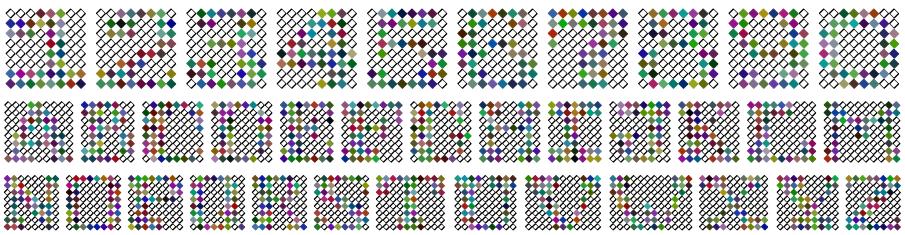
```
\definefontfeature
  [fontthreesix-color]
  [default]
  [metapost={category=fontthreesix,spread=.1}]

\definefont[DEKFontD][Serif*fontthreesix]
```

The `spread` (multiplied by the font unit, which is 12 basepoints here) will add a bit more spacing around the blob:

> This font is used in “The ■rt ■f ■omputer ■rogramming” by ■on ■nuth, not in volume ■, ■ or ■, but in number ■!

Now, keep in mind that we’re talking of a real font here. You can cut and paste these characters. It’s just the default uppercase Latin alphabet plus digits.

Before we go and look at some of the code needed to render this, a few more examples will be given.



In the above example we not only use color, but also a different shape and random colors (that is: random per TEX job). The feature definition for this is:

```
\definefontfeature
  [fontthreesix-color]
  [default]
  [metapost={%
    category=fontthreesix,shape=diamond,%
    color=random,pen=fancy,spread=.1%
  }]
```

Possible shapes are `circle`, `diamond` and `square` and instead of a random color one can give a known color name. Using transparency makes no sense in this font.

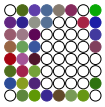
A nice usage for this font are initials:

```
\setupinitial[font=Serif*fontthreesix-initial sa 5]
{\DEKFontB \placeinitial \input zapf\par}
```

The initial feature is defined as:

```
\definefontfeature
  [fontthreesix-initial]
  [metapost={category=fontthreesix,color=random,shape=circle}]
```

We use this in quoting Hermann Zapf, one that for sure is very applicable in a case like this:


oming back to the use of typefaces in electronic publishing: many of the new typographers receive their knowledge and information about the rules of typography from books, from computer magazines or the instruction manuals which they get with the purchase of a  or software. There is not so much basic instruction, as of now, as there was in the old days, showing the differences between good and bad typographic design.  any people are just fascinated by their  's tricks, and think that a widely–praised program, called up on the screen, will make everything automatic from now on.

Some combinations of sub-features are shown in figure 1. We blow up the diamond with fancy pen example in figure 2. Alas, the TEX logo doesn't look that good in such a font. Using it for acronyms is not a good idea anyway, but maybe you can figure out figure 3.

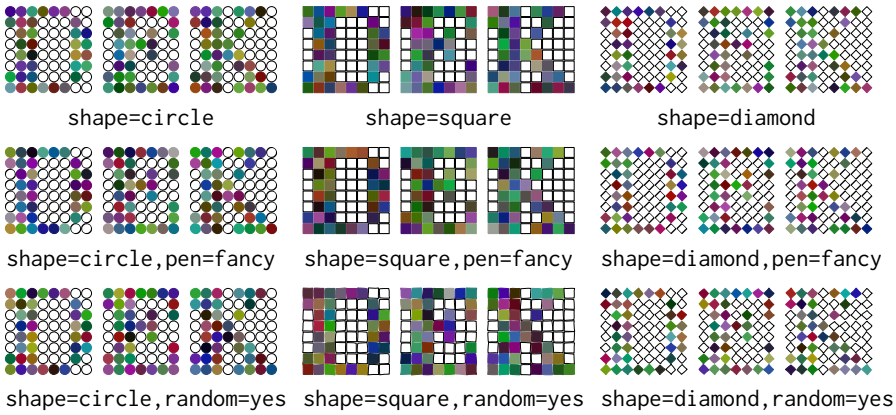


Figure 1.

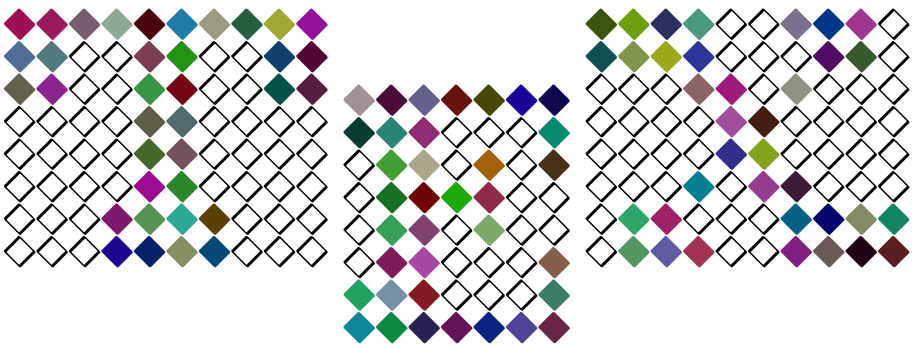


Figure 2.

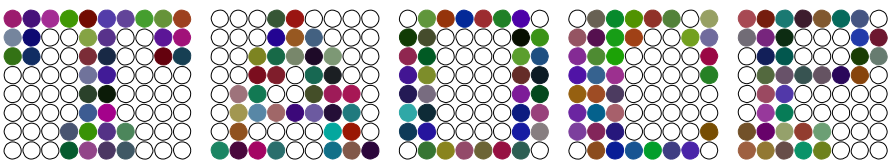


Figure 3.

You can quit reading now or expose yourself to how this is coded. We use a combination of Lua and MetaPost, but different solutions are possible. The shapes are entered (or course) with zeros and ones.

```
\startluacode
local font36 = {
    ["0"] = "00111100 01111110 11000011 11000011 11000011 ...",
    ["1"] = "00011100 11111100 11101100 00001100 00001100 ...",
    .....
    ["D"] = "11111100 11100010 01100011 01100011 01100011 ...",
    ["E"] = "1111111 1110001 0110101 0111100 0110100 0110001 ...",
    .....
    ["K"] = "11101110 11100100 01101000 01110000 01111000 ...",
    .....
}
\stopluacode
```

We also use Lua to register this font. The actual code looks slightly different because it uses some helpers from the ConTEXt Lua libraries. We remap the bits pattern onto MetaPost macro calls.

```
\startluacode
local replace = {
    ["0"] = "N;",
    ["1"] = "Y;",
    [" "] = "L;",
}

function MP.registerthreesix(name)
    fonts.dropins.registerglyphs {
        name     = name,
        units    = 12,
        usecolor = true,
    }
    for u, v in table.sortedhash(font36) do
        local ny     = 8
        local nx     = (#v - ny + 1) // ny
        local height = ny * 1.1 - 0.1
        local width  = nx * 1.1 - 0.1
        local code   = string.gsub(v,".",replace)
        fonts.dropins.registerglyph {
            category = name,
            unicode  = utf.byte(u),
            width    = width,
            height   = height,
            code     = string.format("ThreeSix(%s);",code),
        }
    end
end

MP.registerthreesix("fontthreesix")
\stopluacode
```

So, after this the font `fontthreesix` is known to the system but we still need to provide MetaPost code to generate it. The glyphs themselves are now just sequences of `N`, `Y` and `L` with some wrapper code around it. The definitions are put in the `MP` namespace simply because a first version initialized in MetaPost, and there could create variants, but in the end I settled on the parameter interface at the TEX end.

The next definition looks a bit complex but normally such a macro is stepwise constructed. Notice how we can query the sub features. In order to make that possible

the regular MetaFun parameter handling code is used. We just push the sub-features into to mpsfont namespace.

```
\startMPcalculation{simplefun}

def InitializeThreeSix =
    save Y, N, L, S ; save dx, dy, nx, ny ; save currentpen ;
    save shape, fillcolor, mypen, random, spread, hoffset ;
    string shape, fillcolor, mypen ; boolean random ;
    pen currentpen ;
    dx :=   11/10 ;
    dy := - 11/10 ;
    nx := - dx ;
    ny :=   0 ;
    shape      := getparameterdefault "mpsfont" "shape" "circle" ;
    random     := hasoption           "mpsfont" "random" "true" ;
    fillcolor  := getparameterdefault "mpsfont" "color" "" ;
    mypen      := getparameterdefault "mpsfont" "pen" "" ;
    spread     := getparameterdefault "mpsfont" "spread" 0 ;
    hoffset    := 12 * spread / 2 ;
    currentpen := pencircle
        if mypen = "fancy" :
            xscaled 1/20 yscaled 2/20 rotated 45
        else :
            scaled 1/20
        fi ;
    if shape == "square" :
        def S =
            unitsquare if random : randomized 1/10 fi
            shifted (nx,ny)
        enddef ;
    elseif shape = "diamond" :
        def S =
            unitdiamond if random : randomized 1/10 fi
            shifted (nx,ny)
        enddef ;
    else :
        def S =
            unitcircle if random : randomizedcontrols 1/20 fi
            shifted (nx,ny)
        enddef ;
    fi ;
    def N =
        nx := nx + dx ;
        draw S ;
    enddef ;
    if fillcolor = "random" :
        def Y =
            nx := nx + dx ;
            fillup S withcolor white randomized (2/3,2/3,2/3) ;
        enddef ;
    elseif fillcolor = "" :
        def Y =
            nx := nx + dx ;
            fillup S ;
```

```
        enddef ;
    else :
        def Y =
            nx := nx + dx ;
            fillup S withcolor fillcolor ;
        enddef ;
    fi ;
    def L =
        nx := - dx ;
        ny := ny + dy ;
    enddef ;
enddef ;

vardef ThreeSix (text code) =
    InitializeThreeSix ; % todo: once per instance run
    draw image (code) shifted (hoffset,-ny) ;
enddef ;

\stopMPcalculation
```

This code is not that efficient in the sense that there's quite some MetaPost-Lua-MetaPost traffic going on, for instance each parameter check involves this, but in practice performance is quite okay, certainly for such a small font. There will be an initializer option some day soon. The `simplefun` is a reference to an MPLIB instance that does load MetaFun but only the modules that make no sense for this kind of usage. It also enforces double mode. The calculations wrapper just executes the code and does not place some (otherwise empty) graphic.

Those who have seen (and/or read) "Concrete Mathematics" will have noticed the use of inline images, like dice. Dice are also used in "pre-fascicle 5a" (I need a few more lives to grasp that, so I stick to the images for now!). So, to compensate the somewhat complex code above, I will show how to accomplish that. This time we do all in MetaPost:

This is not that hard to follow. We define some shapes first. These could have been assigned to the `code` parameter directly but this is nicer. Next we register the font itself and after that we set glyphs. We also set the official Unicode slots. So, copying a dice can either result in a digit or in a Unicode slot for a dice. In the example below we switch to a color which demonstrates that our dice can be colored at the TEX end. It's either that or coloring at the MetaPost end as both demand a different kind of Type3 embedding trickery.

We actually predefine three features. The digits one will map regular digit in the input to dice. We accomplish that via a font feature:

```
\startluacode
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
```

This kind of trickery is part of the font machinery used in ConTEXt and permits runtime adaption of fonts, so we just use the same mechanism. The nocheck is needed to avoid this feature not kicking in due to lack of (at the time of checking) yet undefined dice.

```
\definefontfeature
  [dice:normal]
  [default]
  [metapost={category=dice}]
\definefontfeature
  [dice:reverse]
  [default]
  [metapost={category=dice,option=reverse}]
\definefontfeature
  [dice:digits]
  [dice:digits=yes]

\definefont[DiceN] [Serif*dice:normal]
\definefont[DiceD] [Serif*dice:normal,dice:digits]
\definefont[DiceR] [Serif*dice:reverse,dice:digits]

{\DiceD Does 1 it 4 work? And {\darkgreen 3} too?} {\DiceR And how about
{\darkred 3} then? But 8 should sort of fail!}
```

Does ⚀ it ⚃ work? And ⚂ too? And how about ⚄ then? But ☒ should sort of fail!

The six digits and UNICODE characters come out the same:

```
\red  \DiceD \dostepwiserecurse   {`1}   {`6}{1}{\char#1\quad}%
\blue \DiceN \dostepwiserecurse{"2680}{"2685}{1}{\char#1\quad}%
```

It is tempting to implement for instance 7 as two dice (a one to multi mapping in OPENTYPE speak) but then one has to decide what combination is taken. One can also implement ligatures so that for instance 12 results in two six dice. But I think that's over the top and only showing TEX muscles. It is anyway not that hard to do as we have an interface for that already.

So why not do the dominos as well? Because there are so many dominos we predefine the shapes and then register the lot in a loop. We have horizontal and vertical variants. Being lazy I just made two helpers while one could have done but with some rotation and shifting of the horizontal one. The loop could be a macro but we don't save much code that way.

```
\startMPcalculation{simplefun}

picture Dominos[] ;

Dominos[0] := image() ;
Dominos[1] := image(draw(4,4);) ;
```

```
Dominos[2] := image(draw(2,6);draw(6,2););
Dominos[3] := image(draw(2,6);draw(4,4);draw(6,2););
Dominos[4] := image(draw(2,6);draw(6,6);draw(2,2);draw(6,2););
Dominos[5] := image(draw(2,6);draw(6,6);draw(4,4);draw(2,2);draw(6,2););
Dominos[6] := image(draw(2,6);draw(4,6);draw(6,6);draw(2,2);draw(4,2);draw(6,2););

lmt_registerglyphs [
    name     = "dominos",
    units    = 12,
    width    = 16,
    height   = 8,
    depth    = 0,
    usecolor = true,
] ;

def DrawDominoH(expr a, b) =
    draw image (
        pickup pencircle scaled 1/2 ;
        if (getparameterdefault "mpsfont" "color" "") = "black" :
            fillup unitsquare xyscaled (16,8) ;
            draw (8,.5) -- (8,7.5) withcolor white ;
            pickup pencircle scaled 3/2 ;
            draw Dominos[a]
                withpen currentpen
                withcolor white ;
            draw Dominos[b] shifted (8,0)
                withpen currentpen
                withcolor white ;
        else :
            draw unitsquare xyscaled (16,8) ;
            draw (8,0) -- (8,8) ;
            pickup pencircle scaled 3/2 ;
            draw Dominos[a]
                withpen currentpen ;
            draw Dominos[b] shifted (8,0)
                withpen currentpen ;
        fi ;
    ) ;
enddef ;

def DrawDominoV(expr a, b) = % is H rotated and shifted
    draw image (
        pickup pencircle scaled 1/2 ;
        if (getparameterdefault "mpsfont" "color" "") = "black" :
            fillup unitsquare xyscaled (8,16) ;
            draw (.5,8) -- (7.5,8) withcolor white ;
            pickup pencircle scaled 3/2 ;
            draw Dominos[a] rotatedaround(center Dominos[a],90)
                withpen currentpen
                withcolor white  ;
            draw Dominos[b] rotatedaround(center Dominos[b],90) shifted (0,8)
                withpen currentpen
                withcolor white  ;
        else :
            draw unitsquare xyscaled (8,16) ;
            draw (0,8) -- (8,8) ;
```

```
            pickup pencircle scaled 3/2 ;
            draw Dominos[a] rotatedaround(center Dominos[a],90)
                withpen currentpen ;
            draw Dominos[b] rotatedaround(center Dominos[b],90) shifted (0,8)
                withpen currentpen ;
        fi ;
    ) ;
enddef ;

begingroup
    save unicode ; numeric unicode ; unicode := 127025 ; % 1F031

    for i=0 upto 6 :
        for j=0 upto 6 :
            lmt_registerglyph [
                category = "dominos",
                unicode  = unicode,
                code     = "DrawDominoH(" & decimal i & "," & decimal j & ");",
                width    = 16,
                height   = 8,
            ] ;
            unicode := unicode + 1 ;
        endfor ;
    endfor ;

    save unicode ; numeric unicode ; unicode := 127075 ;

    for i=0 upto 6 :
        for j=0 upto 6 :
            lmt_registerglyph [
                category = "dominos",
                unicode  = unicode,
                code     = "DrawDominoV(" & decimal i & "," & decimal j & ");",
                width    = 8,
                height   = 16,
            ] ;
            unicode := unicode + 1 ;
        endfor ;
    endfor ;
endgroup ;

\stopMPcalculation
```

Again we predefine a couple of features:

```
\definefontfeature
  [dominos:white]
  [default]
  [metapost={category=dominos}]

\definefontfeature
  [dominos:black]
  [default]
  [metapost={category=dominos,color=black}]

\definefontfeature
  [dominos:digits]
  [dominos:digits=yes]
```

This last feature is yet to be defined. We could deal with the invalid dominos with some substitution trickery but let's keep it simple.

```
\startluacode
local ligatures = { }
local unicode   = 127025

for i=0x30,0x36 do
    for j=0x30,0x36 do
        ligatures[unicode] = { i, j }
        unicode = unicode + 1 ;
    end
end

fonts.handlers.otf.addfeature("dominos:digits", {
    type      = "ligature",
    order     = { "dominos:digits" },
    nocheck   = true,
    data      = ligatures,
} )
\stopluacode
```

That leaves showing an example. We define a few fonts and again we just extend the Serif:

```
\definefont[DominoW][Serif*dominos:white]
\definefont[DominoB][Serif*dominos:black]
\definefont[DominoD][Serif*dominos:white,dominos:digits]
```

The example is:

```
\DominoW
    \char"1F043\quad \quad
    \char"1F052\quad \quad
    \char"1F038\quad \quad
    \darkgreen\char"1F049\quad \char"1F07B\quad
\DominoB
    \char"1F087\quad
    \char"1F088\quad
    \char"1F089\quad
\DominoD
    \darkred 12\quad56\quad64
```

Watch the ligatures in action:

To what extent the usage of symbols like dice and dominos as characters in the mentioned book are responsible for them being in Unicode, I don't know. Now with all these emoji showing up one can wonder about graphics in such a standard anyway. But for sure, even after more than three decades, Don still makes nice fonts.

A treasure of tiny graphics can be found in "pre-fascicle 5c" and many are in color! In fact, it has dominos too. It must have been a lot of fun writing this! I'm thinking of turning the pentominoes into a font where a GPOS feature can deal with the inter-pentomino kerning (which mighty work out okay for example 36. The windmill dominos also make a nice example for a font where ligatures will boil down to the base form and the (one or more) blades are laid over. It's definitely an inspiring read.

Hans Hagen