Cowfont (koeieletters) update

Taco Hoekwater, Hans Hagen

Abstract

After ten years, the ‘koeieletters’ font is ready for an update. The new version uses OpenType technology to combine the existing four PostScript Type 1 fonts into a single TrueType font. It’s sort of a coincidence that at the tenth ConTEXt meeting, the font also celebrates its tenth birthday.

1 A bit of history[1]

1.1 The artful beginnings

At TUG 2003 in Hawaii, Hans Hagen met with Duane Bibby. Hans was looking for some small images to enliven the ConTEXt manuals and Wiki. A cutout of a very early sketch can be seen in figure 1, but it was soon agreed that consecutive drawings were going to be an alphabet.

Nothing much happened after that initial meeting until the beginning of 2006 when Hans picked up the thread and got Duane started drawing. The alphabet quickly progressed. Starting in a rather naturalistic style like Duane’s ‘normal’ TEX drawings, but later progressing toward a much more cartoon-like style, as can be seen from the drawings in figure 2.

For ease of use, it was clear that these drawings should ideally become a computer font. Taco Hoekwater agreed to take care of the digitization, and luckily the drawings were already prepared for that. As can be seen from the leftmost closeup in figure 3, the cows are drawn inside a grid. This ensures that they are all the same size, which is a vital requirement for a font design. But of course this is a proportional font in the end; it even has kerning and ligatures!

The center drawing in figure 3 is a still rather roughly inked version of one of the in-between drawings (there were many). In this particular one you can see that the mouth of the cow was originally more or less oval, but in the final form (on the right) it became much more hexagonal.

1.2 Digitization

The original sheets were sent to Pragma ADE by regular mail in the beginning of March 2006. Hans scanned the original sheets at 1200 dpi and then forwarded the images to Taco. There were four sheets in all, containing an alphabet with some accents,

[1] This section is an abbreviated version from our article ‘The making of a (TEX) font’, MAPS 34 (2006), pages 51–54. `http://www.ntg.nl/maps/34/11.pdf`

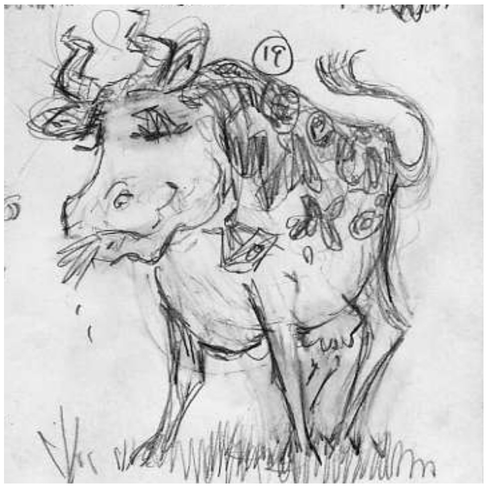


Figure 1: The first drawing

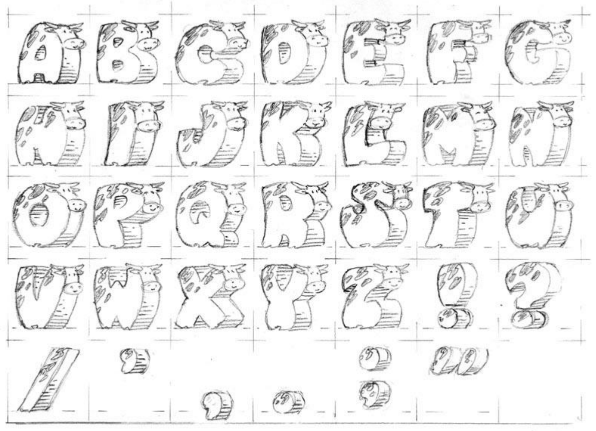


Figure 2: Rough design

Latin punctuation, and a number of TEX-related logos and a few (mathematical) symbols.

The four sheets were digitally cut up into many smaller pieces, each containing a single glyph for the font. This being intended as a decorative font, the character set does not even contain the complete ASCII range. Nevertheless, almost a hundred separate images were created.

These were then imported into FontForge. The autotracer in FontForge, which is actually the standalone `autotrace` program, does quite a good job of tracing the outlines. But, interestingly enough, only at a fairly low resolution. At higher resolutions it gets confused and inserts more than a quadratic amount of extra points as the resolution is increased. Based on empirical tests, the images were scaled to 40% of their original scanned size, resulting in bitmaps that were precisely 1000 pixels high.

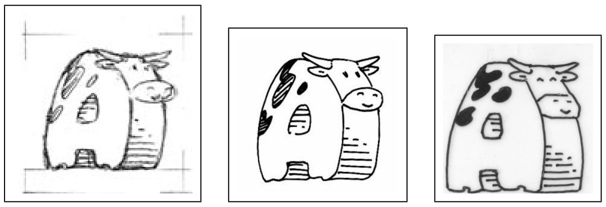


Figure 3: Closeups of the progressive design stages of the letter 'A'.

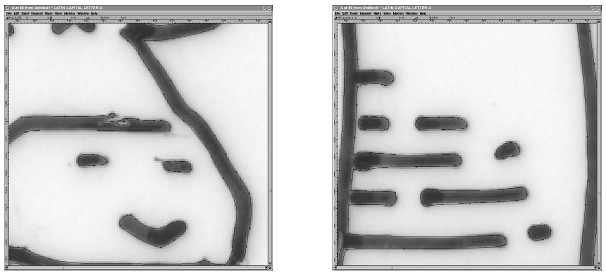
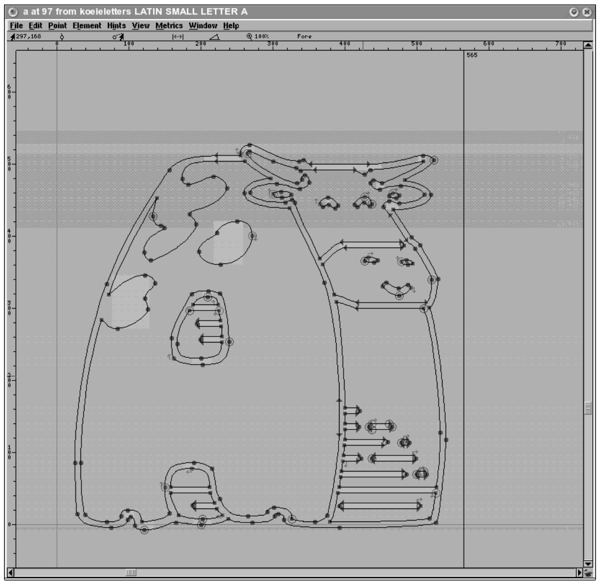


Figure 4: Close-ups of autotracer output


a at 97 from koeieletters LATIN SMALL LETTER A
File Edit Point Element Hints View Metrics Window Help
Figure 5: Finished outline

As was to be expected, the autotracer brought out many of the impurities in the original inked version, as you can see in the left image of figure 4. Luckily, the number of places where manual corrections like this were needed was not so great to force us to reconsider the digitization process.

A more severe problem can be seen in the right-hand image of figure 4. The drawings contain hardly any straight lines. For a font of this complexity, it turned out to be absolutely necessary to simplify the curves. Without simplification, the rendering speed in PDF browsers became unbearably slow. All of the near-horizontal stripes in the bellies were manually removed and replaced by geometric straight lines.

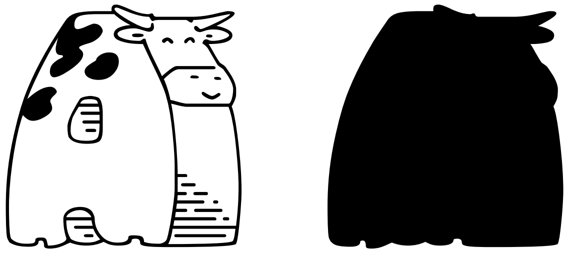
The final stage in the font editor is to add the PostScript hinting. A screenshot of a manually hinted letter is visible in figure 5.

1.3 Finishing the font

The font was saved as two separate PostScript Type 1 fonts, one with the text glyphs and one containing the logo glyphs. The text font is named 'koeieletters', the logo font 'koeielogos'. 'Koeieletters' literally translates from Dutch to English as 'cowcharacters', but the word 'koeieletter' is also used to indicate an enormous character, as in a billboard, for instance.

Eventually it turned out that we needed a second set of two fonts. Sometimes you want to have text in the cowfont but on top of a colored background. The background would then shine right through the hide of the cow and that was of course unacceptable. Hence, we also have the fonts 'koeieletters-contour' and 'koeielogos-contour'.

Here is the final 'A', in the normal and the contour version:



2 Updated version

In ConTEXt MkIV, we prefer not to use Type 1 fonts, and definitely not the tfm-based trickery that was needed to get the 'koeieletters' font performing at its best. Advances in font technology have made it possible to combine all glyphs into a single OpenType font, which goes by the name `koeielettersot`.

2.1 Mathematics

The original Type 1 font already had a math companion but the new font supports math via its 'MATH' table, allowing it to be used for math typesetting just like the other OpenType math fonts that ConTEXt uses, with only a few minor differences:

- There are far fewer glyphs, due to a lack of original artwork. You can imagine that providing the full repertoire of Unicode math would be a bit of a challenge.
- ConTEXt has to do some extra tweaking for the horizontal extensible rules, including those that are appended to radicals.

- There are no accented characters but much can be achieved by enabling the `compose` feature.

2.2 Ligatures for logos

In this font, there is no ‘fi’ ligature. In fact there are no ‘normal’ ligatures at all. However, there is a `dlig` feature in the font which replaces words by hand-drawn versions of those words, and the `ss02` feature can be used to convert these further, into nicer versions with a drop-shadow below.

2.3 Sheep

The numbers and plus and minus in the font can be replaced by versions that resemble a sheep instead of a cow, by enabling the `ss01` feature.

2.4 Colorization

In mid-2016, the ConTEXt font loader started supporting color fonts. Such fonts normally contain emoji characters and for achieving the desired effect two methods are available: overlays and SVG. The first method is cleaner and naturally fits ‘koeieletters’.

The trick is in splitting a glyph into overlaying snippets that each can have a color from a palette. Emoji fonts can provide multiple palettes so that culturally-based colors can be supported. So eventually we could have black Frisian cows and brown ones from the southern part or our country.

The implementation uses virtual fonts. This is straightforward but the current way to inject the needed color directives and information to cut-and-paste the right character can interfere with the way the backend flushes characters. As we managed it with some hackery eventually the virtual font technology might be extended a bit for this purpose.

More challenging was to get math working. Not so much math itself but where regular math fonts use rules for extending radicals, over- and underbars and fractions, we need to use something cowish. Possible solutions are:

- Build the radicals from scratch using snippets: this is cumbersome.
- Preroll with normal rules that get replaced in the node list later: one has to know in what ways TEX constructs glyphs because not every rule is a radical one.
- Patch the math engine to support complex radicals: after some experiments this was considered too dangerous and messy.
- Make the math rules pluggable: adding more callbacks makes no sense for this one exception.
- Make the math rules be (optional) user rules that can be postprocessed: this was relatively easy.

It should be clear that the last solution was chosen. Of course it was not as trivial as we make it sound. First, for radicals we need to register what font we are dealing with so that we can get the right snippets to construct a rule. For the other rules we need to know the font as well and it happens that no such information is available: rules don’t come from fonts. The solution is in two new primitives:

```
% use math specific user nodes:
\mathrulesmode = 1
% the family to take rules from:
\mathrulesfam  = \fam\textstyle
```

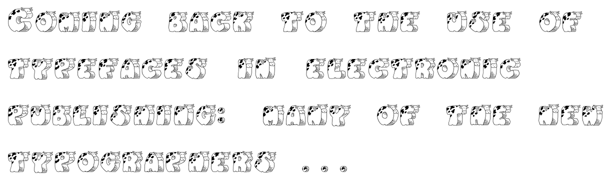
When set, special rules will be constructed that carry the current size (text, script or scriptscript) and family-related font. In the backend the serialization of these rule nodes will trigger a callback (when set) that can inject whatever is reasonable. Of course these extensions are still somewhat experimental and should be used with care.

2.5 Using the font

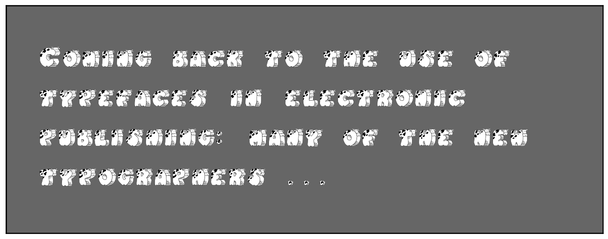
So how is this new font used? Although it is a special kind of font that will seldom be used for a whole document, you need to load it anyway. The easiest way (in ConTEXt) is:

```
\loadtypescriptfile[koeielettersot]
\setupbodyfont[cows,12pt]
```

Please take a look at `type-imp-koeielettersot` to see how these fonts get set up. The beginning of ConTEXt’s usual example Zapf quote (“Coming back to the use of typefaces ...”) comes out as follows:


Coming back to the use of typefaces in electronic publishing: many of the new typographers ...

If you want a colored variant a bit more work is needed. By default the cows are black and white. If you enable color you will see the difference when you show them on a background:


Coming back to the use of typefaces in electronic publishing: many of the new typographers ...

When a font is loaded its color properties are frozen because the backend needs to deal with it.

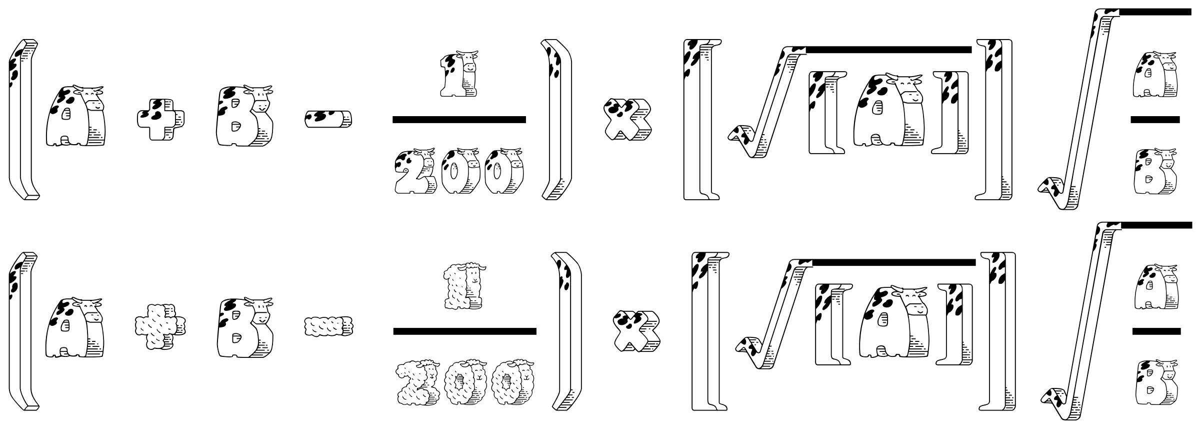


Figure 6: A math formula rendered in 'koeieletters'; cows above, sheep below. The standard black rules in fractions and radicals are fixed in the next figure.

You can, however, influence the color with the `colr` property before a font gets defined. This happens just after loading the typescript file.

```
\definecolor[cowred]    [r=.50]
\definecolor[cowgreen]  [g=.50]
\definecolor[cowblue]   [b=.50]
\definecolor[cowyellow][y=.25]
\definefontcolorpalette[cows]
  [cowgreen,cowyellow,cowblue,cowred]
\adaptfontfeature[sheepcolored] [colr=cows]
```

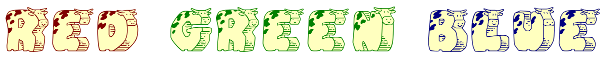
In the example below we show the sheep with colors because we already defined the cows as black and white. You can mix colors by defining fonts explicitly. Note that we only use the second and fourth color in these glyphs.

```
\usetypescript[all][cowsotf]

\definefontcolorpalette[cows-1][cowgreen,
                     cowyellow,cowblue,cowred]
\definefontcolorpalette[cows-2][cowred,
                   cowyellow,cowblue,cowgreen]
\definefontcolorpalette[cows-3][cowgreen,
                     cowyellow,cowred,cowblue]

\definefontfeature[cows-1]
  [cowscolored][colr=cows-1]
\definefontfeature[cows-2]
  [cowscolored][colr=cows-2]
\definefontfeature[cows-3]
  [cowscolored][colr=cows-3]

\definedfont[Cows*cows-1 at 30pt]red\quad
\definedfont[Cows*cows-2 at 30pt]green\quad
\definedfont[Cows*cows-3 at 30pt]blue
```



2.6 Math

As said, we can do math. Take this formula:

```
$\left( a + b - \frac1{200} \right) \times
 \left[\sqrt{[A]}\right] \sqrt{\frac{a}{b}}$
```

This renders as shown in figure 6, cows above, sheep below. The standard rules there don't work well, but figure 7 shows we can do better (implemented with the `\mathrulesmode` mentioned above).

2.7 Logos

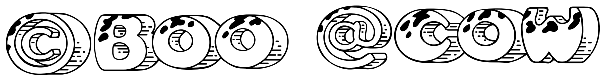
There's a bunch of logos available. You can directly request them but they can also be set automatically.

```
\definefont        [CowsLogo]
  [koeielettersot*cowslogos     sa c]
\definefont        [CowsLigs]
  [koeielettersot*cowsligatures sa c]
\definefontsynonym[CowsOnly]
  [koeielettersot]
```

These definitions can be used to get the logos shown in 8. The last two columns in the table are typeset using:

```
\getnamedglyphdirect{CowsOnly}{contextlogo}
\getnamedglyphdirect{CowsOnly}{c_o_n_t_e_x_t}
```

There are two more ligatures:



and we leave it to you to figure out how to get them.

We end with the best of all: a colored logo.

```
\definefontsynonym
  [CowsColored]
  [koeielettersot*default,cowscolored]

\getnamedglyphdirect{CowsColored}{contextlogo}
```

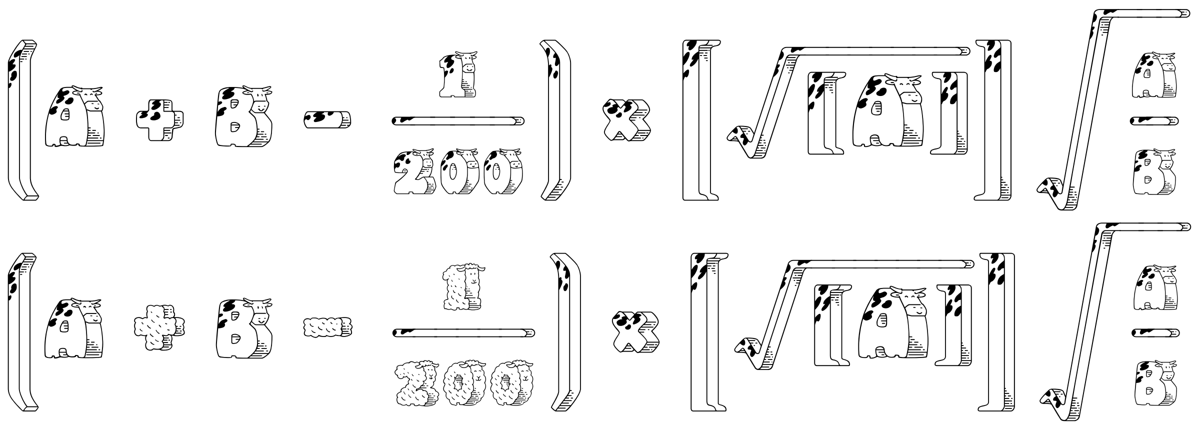


Figure 7: The same math formula as the previous figure, with matching rules created using `\mathrulesmode`.

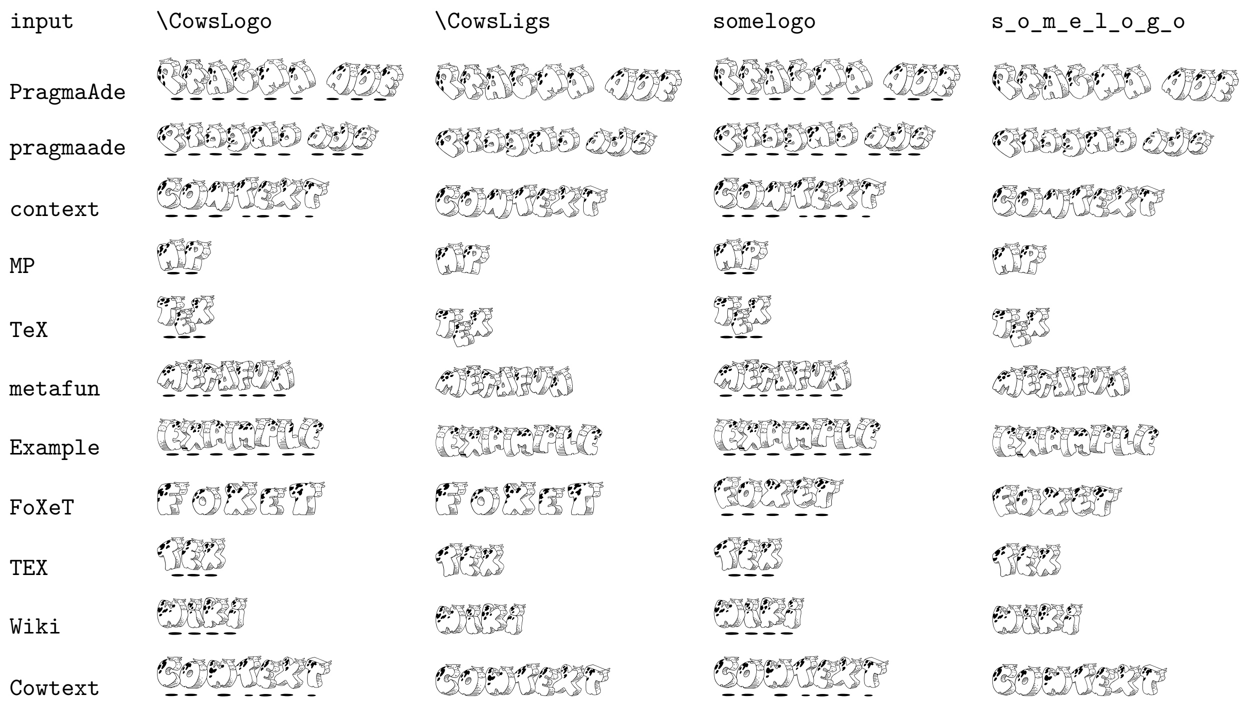
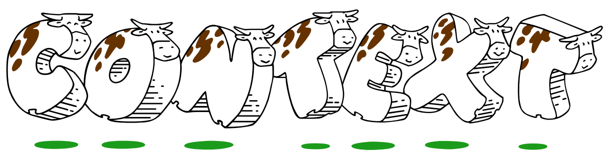

input
\CowsLogo
\CowsLigs
somelogo
s_o_m_e_l_o_g_o
PragmaAde
pragmaade
context
MP
TeX
metafun
Example
FoXeT
TEX
Wiki
Cowtext

Figure 8: Logos in ‘koeieletters’.



To make a quick start with these fonts, you can use one of:

```
\setupbodyfont[koeieletters]
\setupbodyfont[cows]
\setupbodyfont[coloredcows]
\setupbodyfont[sheep]
\setupbodyfont[coloredsheep]
```

where the `koeieletters` variant equals `sheep`. This is possible because we aliased the typescriptfiles to the predefined typeface setups in the typescript file.

◇ Taco Hoekwater, Hans Hagen
ConTEXt Group
http://contextgarden.org