‘Hinting’ of scalable outline fonts

Berthold K.P. Horn

‘Hinting’ refers to methods that guide grid fitting of continuous glyph outlines onto a discrete grid, such as those found on a display screen or laser printer.

The need for hinting

Scalable outline fonts — such as fonts in Adobe Type 1 and TrueType format — have continuous shapes described by mathematical curves. These are used to create a discrete raster of dots on a display or hardcopy output device at a specified size. If such a bitmap is made in a simplistic way — such as simply blackening each cell whose center lies within the contour — then a number of visually distracting artifact arise — such as misalignments of features and breaks in shapes, also called ‘drop-outs.’

Typically hinting is used to do things like:

- make sure stems intended to be equally thick appear equally thick
- suppress overshoots — rounded letters (O) are taller than flat ones (X);
- line up features on different glyphs that should be at the same height;
- avoid ‘drop-outs’;
- keep counters between stems open;
- force consistent spacing between sets of parallel strokes;
- compensate for ‘misfeatures’ of the rasterization algorithm such as drop-outs.

A simple example

A simple demonstration of the need for hinting is the following: suppose you have the letter ‘H,’ which — when scaled to the discrete grid of the output device — happens to have vertical stems from $x = 1.2$ to $x = 2.8$ and from $x = 4.8$ to $x = 6.4$. So the original width of both stems is 1.6 pixels.

Now suppose that the rasterizer ‘inks’ every cell whose *center* is inside the outline. Also suppose the discrete grid of pixels has integer coordinates at the corners and that we round continuous values to the pixel centers. Then the discrete version of the left stem will run from $x = 1$ to $x = 3$, while the discretized version of the right stem goes from $x = 5$ to $x = 6$. In the disrete version then, the left stem is twice as fat as the right stem — a visually very noticeable difference.

Hinting is very important at low resolution (on screen, typically 72 dpi or 96 dpi), where it is hard to get an adequate representation of a continuous shape on a discrete grid. It is still significant at medium (laser printer, typically 300 dpi or 600 dpi) resolution, but is not important at very high resolution (image setter). ‘Font smoothing’ or anti aliasing also reduces the importance of hinting, although hinting can reduce the number of ‘fuzzy edges’ that can be distracting in anti-aliased fonts.

Different hinting ‘languages’

Intelligent grid-fitting of continuous shapes onto a discrete grid requires a mapping that involves more than simply rounding of x and y coordinates. Much can be achieved by instead setting up a piece-wise linear mapping for each coordinate axis, with corner points at the edges of stems and alignment zones. An alternative to such a ‘global’ approach is one that depends instead on individually adjusting coordinates of knots. The latter approach is more powerful, but more awkward to use.

The Adobe Type 1 hinting language is declarative, hence easy to use. Since the ‘brains’ behind it is in the rasterizer, rasterizing quality of a font may be improved if an improved rasterizer is release. However, since the Type 1 rasterizer is not described anywhere, the exact effect of various hinting methods can only be ascertained by experiment.

The TrueType hinting language is imperative and powerful, but also painful to use in comparison. The TrueType rasterizer *is* described, so it is possible — in principle — to predict what the effects of a change might be. One reason the TrueType hinting language *needs* to be more powerful, is that the underlying rasterizer has misfeatures (like drop-outs) that need to be ‘patched up’. All the ‘brains’ is in the TrueType font — not in the rasterizer — so it benefits less from improvements to the rasterizer.

METAFONT also provides methods for grid-fitting outlines. There is no separate ‘hinting language’ in this case. The work is done using METAFONT’s built-in mechanisms for enforcing constraints established by equalities or inequalities.

Hint switching and delta hints

Quality hinting code is tightly interlaced with the program that draws the glyph, since different ‘hints’ may apply to different parts of the glyphs. This is called ‘hint switching’.

For a simple example, let us consider the ‘R’ in CMR10. It has a slab lying on the baseline at the bottom of the left leg and a curled shape that drops *below* the baseline on the right leg. If horizontal stem hints are applied to correctly grid fit the slab on the left leg, then the right leg will be drawn below the baseline, even at very small sizes where this looks ‘wrong’ and where pulling the right leg up to the baseline produces a visually more pleasing result. Conversely, if a horizontal stem hint were to be tuned for the curl in the right leg, then the slab at the bottom of the left leg would tend to be rasterized floating above the baseline. To get the best rasterization at all sizes, the glyph has to be split into two parts and the right leg rasterizered with different hints than those used for the left.

While simple shapes (like a sans serif 'H') require no hint switching at all, some complex shapes such as the 'Weierstrass p' ($\wp$) may require many switches.

TrueType provides a mechanism called 'delta hinting' which makes it possible to make corrections specific to particular resolutions (measured in ppem — pixels per 'em'). This is useful in correcting drop-outs which occur over certain size ranges. A glyph has to be checked at all sizes that a user is likely to use ('waterfalls'), and drop-outs patched up using delta hints that apply at those sizes. It is possible to draw a complete bitmap using TrueType hinting code.

Type 1 fonts use a different (undocumented) rasterization algorithm that first creates a continuous outline that has no drop-outs, but that is approximately $1/2$ pixel too wide all the way around. It then erodes this outline using Euler-number-preserving binary image operations — which cannot introduce drop-outs. Hence there is no need to 'patch up' the result.

Hint types and hinting tools

Some hints are font wide, such as information on 'alignment zones,' dominant stem widths, and overshoot suppression. But the bulk of hinting code is at the character level. Often the character level code is designed to interact with the font level code (e.g. so-called 'ghost' stems interacting with font level alignment zones at x-height, cap-height, figure-height, ascender, descender heights and so on).

Some simple types of hints follow systematic rules and can be generated automatically by some applications such as Fontographer, FontLab, Type-Designer, and Ikarus. Others require judgement; quality hinting is more of an art than a science. For example, you get to choose (roughly speaking) between preserving more of the unique character of a particular face at small sizes versus making it more readable.

There are few good tools for doing quality manual hinting. For Type 1 fonts, most font generation software has some support for manipulating hints graphically. Type Designer even provides for hint switching. But without being able to see clearly what hints go where, it isn't really good enough to be useful. For TrueType, the only hinting tools are from Type Solutions in New Hampshire.

The quality of fonts and of rasterizers

Note that the 'hinting' code can be a substantial part of a font, particularly in the case of quality TrueType fonts. If you pass one of the Microsoft core fonts (Arial, Times New Roman, and Courier New) through one of the font generation tools you will see a dramatic drop in size as a result of the fact that the original hints are lost. In some cases quality TrueType fonts shrink by almost 50% (and lose their quality, of course) by this kind of liposuction. Type 1 font hinting is simpler and so rarely adds more than 10% to 20% to file size.

Hinting code is interpreted by the rasterizer and so its effectiveness depends on how well the rasterizer makes use of it. Adobe Type Manager (ATM) does the best job for fonts in Type 1 format, with PostScript interpreters in printers typically not being quite as sophisticated. Various Display PostScript systems mostly do not as well as ATM either. This may change as Adobe upgrades PostScript interpreters under its control. Naturally 'clones' don't do as well.

Moral

Use only quality fonts. Fonts on those discs that offer 'a zillion fonts for $29.95' are mostly poorly made derivatives of the real thing. Typically the original hinting is stripped out (perhaps because it is thought to be better protected by copyright than the rest of the font program).

◇ Berthold K.P. Horn
Y&Y, Inc.
45 Walden St.
Concord, MA 01742 USA
`bkph@ai.mit.edu`