Technical Note TE530

TrueType Q&As

CONTENTS

This Technical Note contains a collection of archived Q&As relating to a specific topic - questions sent the Developer Support Center (DSC) along with answers from the DSC engineers. Current Q&A's can be found on the Macintosh Technical Q&A's web site.

[Sep 01 1993]

Rotating large KanjiTalk characters on PostScript printers

How can I rotate Japanese font text correctly on the LaserWriter? I'd tried to rotate text using `PicComment(TextBegin/TextEnd)`, but in System 7.1 (KanjiTalk7) I couldn't rotate alphabets in Japanese font correctly. Is there any way to either (1) work around the text rotation (using `TextBegin/TextEnd`) problem itself or (2) hide the unrotated text drawing between `RotateBegin` and `RotateEnd` from QuickDraw, such as "magic pen mode" (23) mentioned on page 20 of the Macintosh Technical Note "Picture Comments--The Real Deal"?

—

The problem with KanjiTalk is that large TrueType characters aren't drawn as normal text. If the font is deemed too big to send to the printer, because the printer might not be able to image it, KanjiTalk images the characters in the Macintosh at device resolution and sends the bitmaps to the printer. Bitmaps aren't rotated between `TextBegin` and `TextEnd`, so those calls fail when trying to rotate Kanji text.

You get duplicated text because there's no `TextMode(23)` to go along with `PenMode(23)`, unfortunately.

You can work around this for printing by seeing whether you're printing to a PostScript printer (`wDev` == 3) and not supplying the QuickDraw representation only in this case. Note that with the next LaserWriter driver (the Level 2 driver currently on the Developer CD), that means the rotated text won't be included in EPS Previews (if you don't draw it, it's not included in the preview), but the printout will be correct.

You may also, if you wish, draw the text to an off-screen pixel map at device resolution (using `PrGeneral`) and just use `CopyBits` to image it to both PostScript and QuickDraw printers. This is a nice workaround as well, but it has a definite disadvantage: the LaserWriter driver always reports the resolution of the printer as 300 dpi even if you're printing to an imagesetter. If your application is printing to a printer with higher resolution than 300 dpi, your text will still be at only 300 dpi resolution. That may be an acceptable trade-off to you.

Back to top

SetPreserveGlyph and font glyph preservation

Date Written: 7/22/91

Last reviewed: 6/14/93

TrueType's `SetPreserveGlyph` call with `preserveGlyph` set to "true" works as you'd expect for Times and Helvetica, but it has no effect on New York, Geneva, Monaco, or Chicago. Why does `SetPreserveGlyph` work this way?

—

Glyph preservation is a function of the font. It turns out that Times and Helvetica have glyphs that extend above the ascent line, thereby enabling `SetPreserveGlyph` to have an effect on a particular glyph. New York, Geneva, Monaco, and

Chicago's characters all fit between the ascent and descent lines, and so do not need to be compressed to fit if `preserveGlyph` is true. A font must have glyphs extending above or below the ascent or descent lines for `SetPreserveGlyph` to have an effect.

Back to top

RealFont and TrueType

Date Written: 9/4/91

Last reviewed: 6/14/93

What's the story with RealFont and TrueType? I'm finding that, of the standard System 7 TrueType fonts, only Symbol and Courier get a TRUE result from RealFont for 7-point.

___

You're correct in your observation of RealFont for the 7-point size of certain TrueType fonts. The explanation is hidden in TrueType Spec--The TrueType Font Format Specification (APDA #M0825LL/A), page 227.

The font header table contains a `lowestRecPPEM` field, which indicates the "lowest recommended pixel number per em," or, in other words, the smallest readable size in pixels. As it turns out, the Font Manager in its wisdom uses this information for the value it returns from `RealFont`. Note that for higher-resolution devices, a point size of 7 does not correspond to 7 pixels; but since the unit "point" is 1/72 inch and the screen resolution is (approximately) 72 dpi, the result corresponds to reality in this case.

The value for `lowestRecPPEM` can be arbitrarily set by the font designer. We all know that small point sizes on low-resolution devices never look great, and even less so for outline fonts. Courier and Symbol have `lowestRecPPEM` = 6, while the other outline fonts in the system have `lowestRecPPEM` = 9. This doesn't mean that Courier and Symbol (TrueType) in 7-point size look better than Timesreg. or Helvetica under the same conditions. It means the font designer had higher standards (or was in a different mood) when choosing `lowestRecPPEM` = 9.

Back to top

SetOutlinePreferred affects only calling application

Date Written: 3/9/92

Last reviewed: 6/14/93

Is the property that is managed by `SetOutlinePreferred` and `GetOutlinePreferred` kept on an application-by-application basis? In other words, will calling `SetOutlinePreferred` affect only my application? As the current value of `outlinePreferred` is saved in a PICT, will playing a PICT affect the playing application's current `outlinePreferred` setting? I assume that `outlinePreferred` is not an attribute of a `GrafPort`; is this true?

___

The `outlinePreferred` setting is stored as a low-memory global value for your application. It is saved and restored during context switches, so it only effects your application and no one else's.

`DrawPicture` should not alter the state of the global for you. While `DrawPicture` internally may set or reset this value, it's supposed to put it back the way it found it when it is done. So, playing a PICT will not affect the current application's outline settings.

And no, `OutlinePreferred` is not part of a `grafport` (as mentioned above.)

Back to top

QuickDraw doesn't draw ASCII 32 ($20) character

Date Written: 3/18/92

Last reviewed: 6/14/93

My TrueType font has all 256 characters defined with a unique glyph. I've been unable to draw the $20 (space) character. `DrawChar`, `DrawText`, `DrawString`, and `DrawJust` all ignore this character in the font and draw a blank character. How can I draw it?

Unfortunately, the problem with the space character not being drawn is hard-coded into the text-drawing routine in the core of QuickDraw. ASCII 32 is always "optimized away," regardless of the font being used or of the particular circumstances. The only workaround is to put the corresponding character elsewhere in the ASCII character encoding (or, if this isn't possible, to use an additional font).

You're lucky that TrueType fonts always render the ASCII code 13 (carriage return) if it has a glyph in the font; for bitmapped fonts, if the character drawing happens with scaling, or with foreground/background colors different from black/white, even the CHR(13) drawing is disabled.

Back to top

Downloadables

Acrobat version of this Note (K). Download
