Superbly Legible Fonts: An Improved Bitmap Typewriter

Don Lancaster
Synergetics, Box 809, Thatcher, AZ 85552
copyright c2005 as GuruGram #53
http://www.tinaja.com
don@tinaja.com
(928) 428-4073

Back in **GuruGram** #37 of **BMFONTS.PDF** and over in our **Precision Bitmap Fonts Repository**, we saw that the secrets to ultra small typography of limited resolution were to carefully **lock the fontshapes** to the pixels being displayed; to use a **post process anti-aliasing** to make each pixel represent a true gray-shaded average of its letter content; to provide a **full transparency** blended style character overlay; and to **work at 1:1 final project size**.

We further saw that such precision small bitmaps were exceptionally useful for photo retouching. Particularly on such **eBay** items as electronic test equipment panels. Besides being of interest for display research, working on **eBook readers**, or doing ultra small signage as might be needed on a model railroad.

I've recently completed the beta of a new **AUTOBM1.PSL** bitmap typewriter that nearly completely automates the once tedious process of creating suitable very small bitmapped fonts. Because font characters are now created **on demand** as needed, any font can be used and there is no longer any need for extensive precompiled bitmap libraries.

A quick **demo** can be found **here**. Some real world projects appear **here**, **here**, and **here**, and especially **here**. Be sure to reclick for full size.

How it works

A precision font family size can be specified by the **height** and **width** of its pixels for an upper case letter "A". While our technique works with any **PostScript** font, the choice of **MyriadPro-Bold** seems to work extremely well. Besides its being included in Adobe's **Acrobat 7** package.

The object of our code is to create a **bitmap** of the chosen lettering in the chosen sizes **properly blended** over a chosen background. This "scratch" bitmap can then be cut and pasted into your main bitmap image being post processed.

A scratch bitmap size of 400 x 400 is presently in use. It is split into **four** horizontal stripes to let you work up to four backgrounds at any given time.

AUTOBM1.PSL is written in raw **PostScript**. While this was developed by using my **Gonzo Utilities**, Gonzo is not absolutely needed during runtime as several internal procs have been included. To use your bitmap typewriter, you bring your code up in **Wordpad** or any other editor, make desired changes, and then send over it to **Acrobat Distiller**. Distiller in turn uses **PostScript-as-Language** to write a new scratch bitmap to disk.

The bitmap typewriter has **three** independent internal array-of-arrays group of strings representing individual pixels. One each for red, blue, and green. Only after all characters are properly entered do these string groups get converted to a conventional **.BMP** combined RGB 8-bit bitmap and are written to disk for you.

One or more font families will get held in a separate array-of-arrays group of data values. With the exception of our initial "A" character and a space, **characters are added only when and as needed**. Reuse of any character during imaging is thus much faster.

After analysis, a typical letter "**A**" might look like...

```
[  [1.0000 0.8888 0.0000 0.0000 0.9166 1.0000]
   [1.0000 0.6111 0.1111 0.0000 0.6666 1.0000]
   [1.0000 0.3888 0.3055 0.1666 0.3888 1.0000]
   [1.0000 0.1111 0.5277 0.4166 0.1111 1.0000]
   [0.8888 0.0000 0.4722 0.4166 0.0000 0.8888]
   [0.6111 0.0000 0.0000 0.0000 0.0000 0.6111]
   [0.3888 0.1111 1.0000 1.0000 0.0555 0.3888]
   [0.1111 0.3611 1.0000 1.0000 0.2500 0.1111]  ]
```

Each data value is in the same relative position as its generated pixels. Each data value represents **an average** of 36 or more samples of a high resolution font. The data value rules are...

A 0.0000 data value is fully OPAQUE.

A 1.0000 data value is fully TRANSPARENT.

This is similar to a **0 setgray** being maximum black ink. You can see how the center top and lower corners of the above array clearly show an uppercase "**A**".

When entering the character onto the bitmap, the data value decides how much the "new" character pixel will overwrite its "old" background. Ferinstance, if the old blue background pixel is color 64 and the new blue character pixel is color 128 and the data value is 0.2500, the new blue value will be 112. Because the new color is one quarter transparent and three-quarters opaque.

We'll note that the **MyriadPro-Bold** "A" is not quite symmetric, so some values do not exactly mirror each other. The right foot is apparently slightly wider.

Analyzing the Font Characters

The tricky part is starting with a large high resolution font character and stuffing it down into an exact number of properly offset horizontal and vertical pixels. From which we can derive the needed opacity data values for each pixel.

We always start with an upper case "**A**". And use it to define our font size. Rather than messing with font metric tables, we will redo everything the hard way. After a large 100 point imaging of a character, a vertical sample line is first started to the left and slowly moved right. Points are sampled using **PostScript's** little known and incredibly powerful **infill** operator. On the first hit of an opaque sample, the character's left boundary **clb** can be determined.

A leftgoing sweep is used to find the **crb** right boundary, while vertical sweeps are used for **ctb** and **cbb**. The "**A**" values are then saved as **gcwide**, **gchigh**, and **gctb** to be used as **normalized values** that will determine how many vertical and horizontal pixels will be needed for the other characters.

The upper case "**A**" will be shown in its called-for horizontal and vertical pixel size. By using the normalized "A" values, the other characters are auto adjusted to the needed pixel count widths, heights, and vertical offsetting. Thus, a "W" will be wider than an "f", and the top pixels of a lower case "c" will start below its upper case equivalent.

Descenders are handled automatically, stopping when you run out of array.

Two rows of invisible **[0.0000 0.0000 0.0000 ... 0.0000]** padding are added to the **top** of the upper case "A" and many other characters. This allows for the lower case "h", etc... whose ascender may be slightly **higher** than a capital letter.

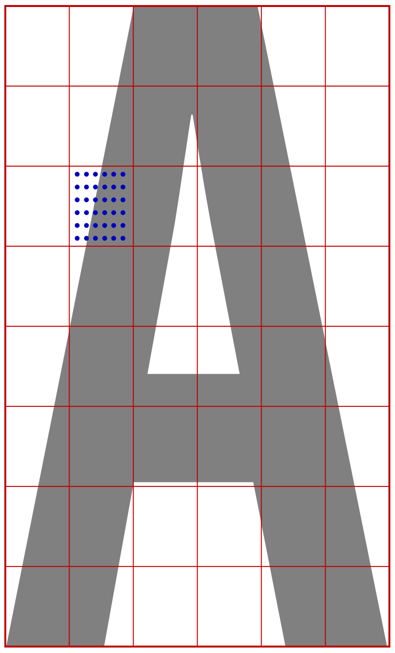
As is apparently the case with **MyriadPro-Bold**.

Your scratch bitmap vertical positioning will thus normally be **two pixels above** the **top** of most characters.

A convention...

All characters build from the TOP down, starting TWO PIXELS ABOVE the top of an upper case "A".

Let's look at an example of how the post anti-aliased and averaged data values are gotten...



Grossly oversized pixels are carefully mapped to **exactly** fit a large high resolution plot of the character to be evaluated. Each pixel is **spatially** sampled 36 or more times as shown in blue. An average is created by using the **infill** operator to determine opaque or clear. In this example, we seem to have 14 white hits and 22 black ones, for a transparency of 14/36 = 0.3811. Which exactly matches the data value in our previous array.

This also is the key secret to the extreme legibility. Other characters are similarly mapped by first relating them to the "A" character defaults.

Getting Started

You normally begin a bitmap typewriter session by loading **AUTOBM1.PSL** into **Wordpad** or a similar editor. Most of your changes will normally be at the very **end** of the document. Your first order of business is to alter **targetfilename** and **targetfilenameprefix** so your bitmap will end up where you expect it to.

Current color values may be defined when needed by a **/curcolor [83 89 74] store**, or predefined to a name such as **/teklight** or **/tekdark**. The integer values range from 0 to 240 and may be read directly from the color palette in **Paint** or similar programs. 0 is black, 240 is maximum luminance.

Your background color(s) are put down next by using **setbackA** on up through **setbackD**. With **setbackA** taking care of the **bottom quarter** of your screen.

A foreground color is then chosen as is the desired current font size. Such as with a **tekdark** followed up by a **6 8 setbmsize**. The first digit is the bitmap **width**.

You are now ready to type away. Enter a **3 280 (Your String of Characters) setgraystring** to continue. The first integer is the horizontal start position, while the second is two pixels **above** the top pixel of your upper case characters. Keep repeating string entry till all needed sizes, colors, and values are complete.

To complete your session, save **AUTOBM1.PSL** under a new filename and send it to **Acrobat Distiller**. Your new scratch bitmap should be automatically generated and appear where you expected it to. Note that there is no actual **.PDF** output; our main goal here is a **.BMP** scratch file. A "**no file produced**" error can be normal and expected.

Bells and Whistles

The bitmap typewriter automatically moves you down to the next line if you run out of character room. Breaks are on full characters, but **not** on whole words. Carriage returns work as expected. Their size is set by the **yinc** variable.

If you enter a **0 0 (new character string) setgraystring** instead of using position values, you will go to a **default** upper left corner (of 3 390) on your **first** string and will **continue where you left off** on successive strings.

The global intercharacter spacing between characters is set by **globalkern** and typically will be one pixel for smaller fonts or two for larger. Note that the space character will be the **sum** of **one** space character width and **twice** the value of your **globalkern**. Thus, spaces may appear wider than expected.

Two characters are optionally reserved by **kern-char** and **kern+char** for individual pair kerning. A redefined "‘" backs you up one pixel column to the left, while a "~" moves you forward one to the right.

The code includes an optional **shrinkfactor** that lets you clip the characters somewhat before analysis. This is possibly useful to darken and improve smaller font sizes but looks awful on the larger ones. Try using a shrinkfactor of **0.04** below 7 pixels of height. But leave it at zero otherwise.

There are some diagnostics hidden in the code that can be activated by uncommenting. Others are easily added. These give you views of the actual characters being sampled, outputs of the generated arrays, and various timing options. Error trapping is modest at present.

Each new generated character is currently reported to the logfile, giving you an estimate of progress. A typical hundred character bitmap annotation should take around nine seconds or less. These routines can be further sped up significantly by reducing boundary precision and averaging samples, along with some new code optimization.

For More Help

Lots of additional uses and examples of true antialiasing are in our **Fonts and Images** and **Precision Bitmapped Fonts** library pages. More on **PostScript** and **Acrobat** in their separate resource areas. Free **Gonzo Utilities** and many use examples are found **here**.

Additional consulting services are available per our **Infopack** services and on a contract or an hourly basis. Additional **GuruGrams** are found **here**.

Further **GuruGrams** await your ongoing support as a **Synergetics Partner**.