Are fonts programs or data?

Dave Crossland
dave@lab6.com

January 27, 2008

Contents

1 Introduction **2**

2 A broad definition of a font **2**
2.1 What is a computer? . . . 2
2.2 What is software? . . . 2
2.3 What is a bitstream? . . . 3
2.4 What is a programming language? . . . 3
2.5 What is Lisp? . . . 3

3 Sourcecode **4**
3.1 What is compilation? . . . 4
3.2 What is an encoding? . . . 5
3.3 What are file formats? . . . 5
3.4 What are standards? . . . 5
3.5 What is a typeface? . . . 6
3.6 What is a typeface designer? . . . 6
3.7 What is a typeface family? . . . 6
3.8 What is font engineering? . . . 7
3.9 What is PostScript? . . . 7
3.10 What is a bitmap? . . . 7
3.11 What is a vector? . . . 7
3.12 What is a font? . . . 8
3.13 How are fonts made? . . . 8
3.14 Metafont; programs to generate font data; back to the future . . . 8
3.15 visual font editors . . . 8
3.16 font editor source file formats . . . 9
3.17 How did fonts become programs? . . . 9
3.18 TrueType hinting programs . . . 9
3.19 Complex Script Logic . . . 10
3.20 AAT layout programs . . . 10
3.21 OpenType layout tables . . . 10
3.22 SIL Graphite . . . 10

4 Conclusion **10**

1 Introduction

Stallman (1992) makes a strong case that all programs should be free. Are fonts programs? To answer this I consider what fonts represent, the nature of various font technologies, in the context of what computers, software, data and programs are, and legal views of the issue.

2 A broad definition of a font

A font is commonly understood to mean a unit of digital information that describes a set of letter shapes in a way that can be used to typeset text with computers.

2.1 What is a computer?

A 'computer' is something that copies and modifies information by following a list of instructions. They have two essential parts, 'memory' to store information and 'processors' to follow the modification instructions.

In the 19th century computer processors were people who used mechanical calculators and stored information on paper (Grier 2005) and in the 20th century computers became electronic devices, based on the concept that a series or 'circuit' of electrical parts - diodes, capacitors and switches - can store and modify information.

Then three inventions came to define the contemporary computer were transistors, magnetic cores, and integrated circuits.

Transistors are electric switches that change their state quickly, invented by Julius Lilienfeld in Germany in 1925 and first realised at Bell Labs by William Shockley, John Bardeen and Walter Brattain in 1947. (Riordan & Hoddeson 1998) Electric circuits with complex designs to arrange many transistors in a way suited for computation are the basis of contemporary computer processors.

[flex capacitor image from BTTF, with the 3 parts of transistor]

Magnetic core memory was a complex design for arranging a different kind of electric switch, one that maintains its state even when it has no electrical current running through it through magnetism. It was invented at MIT by Jay Forrester in 1951.

Integrated circuits or 'chips' are electric circuits fashioned out of a single material, such as silicon, that contain all the parts of an electric circuit. They were invented at the British Ministry of Defence by Geoffrey Dummer in 1952 and first realised at Texas Instruments by Jack Kilby in 1959. They meant the two main parts of computer hardware, processor chips and storage memory, could be made smaller faster and cheaper, year on year.

Moore described the rate in 'Moore's Law,' which is that the number of transistors that can be packed into a square inch of silicon chips doubles every 18 months.

2.2 What is software?

Programs and Data / Programs do processing and data is processed

At that time, 'hardware' was commonly understood to refer to all the physical parts of computers; 'software' inversely correponded to the intangible parts: the information to copy, and the instructions to modify the information with.

These became known as data and programs. Processor and memory hardware directly correspond to these two kinds of software, in its original sense.

Today, the meaning of "software" has shifted to refer to only to programs. [when did this happen?] The contemporary term for both is a 'bitstream.'

"Software is "run." Data are "processed."" (The Computer Language Company 2007)

2.3 What is a bitstream?

When information is stored by a computer, the information is encoded as digits.

The English word for a symbol that represents a rational number is 'digit' because in ancient Latin the word for 'finger' is 'digita' and the ten fingers of the human man are useful for expressing small numbers.

Any rational number can be expressed in any number base. While ten humans fingers mean the natural base for humans is 10, electric switches are in either an on or off state, so the natural base for electic computers is 2. This is a binary value, 0 or 1, and a 'binary digit' became concatenated into 'bit.'

Information is digitised using some form of computer hardware, from very basic inputs of the first home computers that had a panel of on-off switches that corresponded to the states of single bits, to the common keyboards that soon followed and convert human alphanumeric symbols to one of tens of thousands of corresponding numbers, to sophisticated hardware like the sensor behind the lens of a digital camera that converts light values into millions of picture elements or 'pixels.'

A collection of bits that forms a unit of digital information is a 'bitstream.'

2.4 What is a programming language?

Raw program bitstreams or 'machine languages' are not an expressive language for humans to write programs in, so programming languages were developed. These were initially simply mnemonic codes for writing direct processor instructions in, 'assembly languages,' but more powerful (efficient, convenient) ways of expressing programs were quickly developed: 'Programming languages.'

Although new programming languages have been constantly designed, and continue to be, Graham (2001) argues that the most powerful programming language possible was 'discovered' at MIT by John McCarthy in 1958: Lisp.

2.5 What is Lisp?

Graham suggests that Lisp was 'discovered' rather than 'invented' or designed. We say scientists "discover" science rather than "invent" it because science describes natural phenomena. Lisp is akin to a natural phenomena of computation, based on the concept of "a simple data structure called a list for both code and data."

Powerful features naturally arose from not distinguishing code and data, such as metaprogramming where a program writes a program that is run. The

powerful features came at the price of popularity though: Since Lisp is thoroughly unlike other programming languages at a syntactic level, it is very alien to people who have already learned other languages.

[the other most powerful language, C, why its powerful (as close to machine code as possible but is portable) and how it uses very diff..] All other programming languages use very different syntactic structures for code and data so it is impossible to treat code and data interchangably - if they didn't, they would be 'dialects' of Lisp - and this makes the distinction seem harder than it really is.

Lisp is significant to surveying what fonts are because it highlights the fundemental softness of the distinction between programs and data. When a bitstream is interpreted by a computer processor to perform an action, such as turn on a light, it is a program. When a bitstream is processed by a program, it is data.

A broad theme in the history of computing is that features are presented as "new" but have actually been implemented many years ago and languished in obscurity. All other programming languages are based on the idea of abstracting machine language into more human-expressable language, and these have slowly gained the powerful features of Lisp. Graham suggests that the language which shares the most of Lisp's features, while maintaining a familiar syntax and not actually becoming a variant of Lisp, is Python.

[Python Origins Story Goes Here]

3 Sourcecode

The human-readable expression of a program in any programming language is called 'source code.' For a computer processor to follow the instructions expressed in source code, it must be converted or 'compiled' into machine language.

3.1 What is compilation?

The process of compilation means the verbose source code becomes a single terse and compact lump of code, called 'object code,' or colloquially 'a binary.'

The process of compiling source code into object code can take a long time, especially if the process takes several passes to generate very efficient machine code. It also neccessarily obfuscates the way the program works.

This is practically convenient for users because the compiled program can be used immediately and runs quickly, but the obfuscation puts them at a social disadvantage; they cannot understand how the program works, so when the program inevitable exhibits unexpected behavior - 'bugs' - they are inhibited from understanding the cause of the problem and suggesting improvements to fix it. They also cannot go futher and fixing problems for themselves privately and extend and adapt the program to their specific needs.

[If you are a proprietary developer, it is also advantageous to distribute only object code to users so the way your program works is obscure; hiding any spying of the users activities, for example in Adobe Photoshop and Windows XP; other proprietary control benefits?]

Nevertheless, it is straightforward to observe the job that any program does, and independently write a new program that does a very similar job. But

interchanging data between two different programs can be very complex, because data can be treated in a way akin to compilation.

3.2 What is an encoding?

Raw data bitstreams are also not meaningful to people, and so schemes for corresponding numbers to alphabets were developed, called 'charachter encodings.'

The Morse Code, invented by Samuel Morse and Alfred Vail in 1844, is an early example of a simple 'encoding scheme,' as it encodes the English alphabet with long or short telegraph key strokes - as bits.

These enabled human languages to be processed by computers, both for programming languages and for text data, and as processor and memory technology became fast and large and cheap enough to handle ever more detailed information, encodings were developed for all kinds of information, not just alphabets.

Today the meaning of 'encoding' has shifted, like the term 'software,' and now refers only to encodings of alphanumeric symbols. Encoding schemes for all the other, more complex, kinds of human information are now refered to as 'file formats.'

3.3 What are file formats?

Consider a file format for tables of numbers, encoded as plain text numerals and separated into columns by commas and into rows by line endings. This format is easy for anyone to understand, so anyone can write programs to process file in that format.

Alternatively the same table can be encoded in a binary data file format that is optimised for the needs of a specific and complex program such as a spreadsheet program.

This obfuscates the source data, just as object code obfuscates the correponding source code.

For other programs to read and change that information easily - that may be totally different kinds of programs, for example that can generate chart graphics or read the data very quickly - the format specification must be published, either by the developers directly, or by the public studying the way the program object code works - 'reverse engineering' it - and publishing an unofficial specification. There must also not be legal restrictions on writing such interoperable software.

3.4 What are standards?

When a file format is widely used it becomes a *de facto* standard; a 'true' standard is one with a publically documented specification and no legal encumberances.

File formats are now a part of everyday conversation in parts of the world with ubiquitous computers - and even a political issue given the long term implications of data stored in *de facto* standards controlled by a single company that may in the future go bankrupt. (ODF vs MOOX)

3.5 What is a typeface?

A typeface is a set of shapes that represent the letters of a written language that are harmoniously related; the shapes look like they belong together.

The etymology of the word "typeface" comes from the printing technology of medieval Europe, where a "type" was a small long metal cuboid with the shape of a letter in relief on one face of the cuboid. These types were arranged into bound blocks that were printed. The shapes of the letters initially mimiced the calligraphy of the time and were created through a process known as "punchcutting." (Smeijers 1996)

Calligraphy is hand writing done in a stylised way to create letter shapes that are harmoniously related. As printing technology developed, the shapes of letters on type faces diverged from traditional calligraphic forms, although they retained the elemental forms dervied from the structural charachteristics of hand writing. (Noordzij 2005)

Printing technology developed using many different kinds of media to express letter shapes, and many did not involve anything like the metal types. So today a set of letter shapes intended to belong together, abstracted from the media in which they are expressed is a "typeface."

3.6 What is a typeface designer?

A typeface designer is someone who fits these shapes to a purpose, which can range from something as subliminal as subcommunicating feelings in text set in that typeface to something as rationally exacting as fitting 0.1

A professional typeface designer is someone who is recognised by others as skilled at fitting shapes to a purpose, and has been paid by others to do so.

Typeface designers are distinguished from 'typographers'; the people who set text in a typeface they choose - which is now everyone who uses computers.

3.7 What is a typeface family?

Any typeface occupies a position in a "design space." van Blokland's (2007) There are various parameters, scales or axes of form universal to all typefaces, that derive from the structural characteristic of certain usage scenarios: expanded and condensed typefaces vary in their horizontal proportions, being wider or narrower than the 'regular' typeface; caption and display typefaces are intended for use at small or large sizes where optical distortions of human vision and printing/display technologies alter the appearance of identical shapes rendered at different sizes; in Latinate scripts, italic shapes are used for expressing emphasis. Each axis of variation can be combined with the others.

Typeface design end-game is the selection of positions in the design space that are interpolated from specified (drawn) positions but are better suited to use cases than the drawn positions. ???CRYPTIC???

A typeface designer's selection of positions in the design space, when aggregated and distributed to typographers, is called a "typeface family."

Since a typeface is an abstracted set of shapes, it can be expressed in many kinds of media: metal, wood, potato and bitstreams. That is to say, a font is a typeface expressed in a particular medium, especially encoded as a bitstream.

3.8 What is font engineering?

Thus it is useful to distinguish 'typeface design' from 'font engineering,' often also referred to as 'font production.' The roles were initally both done by the same person, a punchcutter, but "by 1900 there were very few punchcutters left, and most of these ... were skillful executors of other people's designs." (Smeijers 1996, p. 72)

The 'litmus test' that the same typeface is used in two different media is not based on the shapes in the media; it is based on the similarity of appearance of text rendered with each medium as identically as possible.

For example, consider the way that ink "bleeds" and spreads out when tranferred from printing types onto paper. This is typically accounted for by typeface designers by adding "trapping" to the smallest areas of white space in their lettershapes, ???define glyphs above and insert here??? such as between the bowl and stem of a "d."

If the precise shapes of letters on metal types are copied exactely into another printing medium that has less or zero ink spread, blocks of text printed with that new medium will appear light define light and dark colour of textblock? and the letters themselves on closer inspection will appear "spindly" and odd.

Although this may sound obvious, the early digital font versions of "Bembo" suffered badly from exactely this problem, but the use of a trademark name associated with a high quality phototypesetting (???) font ensured it was popular for some time.

Historically, a font in typographic jargon was "a set of charachters of any one size and style: say, 8 point Baskerville Italic." (Smeijers 2003, p. 62)

But with Adobe's introduction of PostScript 'Type 1' fonts, computing jargon expanded to become "a file that contains a complete set of type of one style and face." [REFERENCE]

3.9 What is PostScript?

type 1fonts are the mostpopularfontformatofall time. they werepartof postscript digital imagesetting system.

postscript had 3 different things, images in bitmaps, graphics in vectors, type in fonts.

3.10 What is a bitmap?

...

Image like a compiled program - source is obfuscated into object to use, hard to change without original source

3.11 What is a vector?

postscript describes vector shapes with bezier curves, 2 points with a handle each to define the curve.

ILLUSTRATE THIS

footnote, there are other ways of describing curves mathematically as implemented by Peter Karow's Ikarus and Raph Levien's Spiro

Vector like interpreted program - source code is used directly, easy to change without original source

3.12 What is a font?

Postscript fonts were just special kinds of vectors; a file format containing vector data for letter outlines shapes, plus a table of numeric data for the interletter spacing (metrics and kerning- explain in what a typeface is above).

That is, postscript fonts are clearly data, not programs.

font outlines are numeric data that need programs to visualise to become meaningful to people. the differences between font source code outline data and font outline object code data are slight; the obfuscation is almost zero and you can edit them easily in any font editor, although there is some loss of information from the source code (guidelines, alternate and past versions, bitmap forms that are traced).

??All images, bitmap or vector, are like this. the RAW data in a camera is compressed and then lost, eg.??

3.13 How are fonts made?

3.14 Metafont; programs to generate font data; back to the future

A compuer professor at Standard, Donald Knuth, was writing "The Art of Computer Programming" and was so unahppy with the digital typesetting programs of the 1970s that he took several years out from writing the book to develop his own typesetting system, TeX.

Like Lisp, TeX pioneered some advanced concepts that have only recently made their way into mainstream font technology - if at all.

metafont is a programming language designed for expressing letterforms by don knuth in 1970s. its an algebraic programming language, which is quite a rare kind, and because it is a language instead of a graphic user interface, it is unusually powerful as it has a "skeleton/flesh" model of stroke paths and pen nibs travelling along the paths to create the basic shapes, and then eraser pens further changing the basic shapes. this is very powerful because is means the design space of a type design can be very efficiently staked out with interpolation between different strokes and by using different shaped nibs for strokes and erasures.

but since it was a programming language, not a directly visual design process, this was very unfamiliar with visual-minded designers and never became popular despite its power - like lisp. Knuth knew Zaph, a prolific and wealthy german calligrapher and type designer, and he designed a mathematical typeface (Siegel 1985) and they tried to express this in metafont, like computer modern, the initial font that knuth designed, but doing it in the unusual stroke-path way took too much effort and was abandoned; instead metafont code was used in the way typical of visual systems, describing simply the outlines of the specifically designed shapes.

3.15 visual font editors

In the same way that programming languages based on simply abstracting machine code slowly included the powerful features developed for Lisp decades earlier, the powerful features of Metafont took years to appear in font editors

based on simply describing the outlines of type forms, and were far simpler than what was possible with metafont. [story of Adobe MM]

but they are equivalent, because a metafont font isnt really a font, its a program that generates a font. like a font editor.

Fontforge is a free software font editor with many unique featurs, inspired by fontographer because fog was the *de facto* standard editor for years, until FOG company was sold off and development of fog ceased. FF went on to be a world class editor with many unique font engineering features; it was the first program to include spiro editing directly on canvas not written by raph, by turning raphs code into a library, "libspiro", and also supports debugging truetype hinting instructions thanks to the freetype library, and also supports the Apple Advanted Typography format with its 'automata state machines' (Haralambous 2007)

3.16 font editor source file formats

FF has a plain text source format, SFD. but its considered hard to install and hard to use (or at least ugly to look at, no antialiased canvas or delicately shaded menus and scrollbars like modern UIs) and is not widely popular at this time. the popular editoris now fontlab, and like FOG has a secret binary source format, VFB, which is the current de facto standard for font sources.

UFO, export font outlines in XML source; make other font editors - robofab guys made their own programs so they had software freedom and were not held hostage by fontlab, like superpolator which is the most advanced font outline interpolation program ever written but is available under proprietary terms and like frederik berlans' stroke-nib based font editor which allows the use of metafont's powerful calligraphic model to be used in a visual way and to export the outlines as UFO back into an outline editor for font object file production. like the python language containing the most powerful features of 70s lisp, the python font programs contain the most powerful features of 70s metafont.

revivals of type designs. revivals have 'source,' in non-digital letterforms; parts of another printing technology, or printed matter, or letter carving, etc. ALso raphs new software that takes a scanned text block, composites the letters to give an archetypical letter, and then he invented (and patented) a new kind of curve technology, Spiros, that makes drawing vector outlines around existing scanned in shapes that are mathematically smooth, very easy. (explain this much better as per my slides.)

original type designs. ...

3.17 How did fonts become programs?

3.18 TrueType hinting programs

Lisp, programs-data. XGridFit, XML data to construct Hinting programs with. Hinting programs are processed when TrueType/OpenType fonts are displayed on screen with a text renderer that like FreeType

Hinting is on the way out though; high res hardware like OLPC makes it totally obsolete, and FreeType, Mac OS X and Vista now ignore them and have autohinter programs that do the job better than manual hints.

3.19 Complex Script Logic

Unicode encoding, worlds languages. but worlds languages arent allsimple layout like latin (latin is complex if diacrtiics and ligatures are assembled; house's ed crazy)

complex script logic = programs as part of fonts.

3.20 AAT layout programs

3.21 OpenType layout tables

focusedon unicode; not as clever as AAT but better for designers.

3.22 SIL Graphite

for languages/scripts (in typeface, after calligraph must explain script? or not, and front notice that this aimed at and assumes knowledge of type design community, so hackers may not get it) not in unicode or with highly complex layouts.

Data preferred; fonts with programs inside rejected because? usability of tools? designers are not programmers? power of format not needed?

By making OT logic outside the font, they seem like pure data. but the tables are in fact obfuscated, like raw machine code, and programs to make fonts may have sophisticated programs in a more abstract programming language (even if the programs are constructed visually, like VOLT) that are compiled into OT tables when the font binary is compiled.

4 Conclusion

fonts are generally thought of as data, and this is a simplistic view that only sees the outlines (the only visible part, and the type design part), but is appropriate for 'display' type designs. but for 'text' type designs, the invisible font engineering part means those fonts are complex and combine outline data and programs written specifically for those outlines.

these programmatic parts must be free for the same reasons all programs must be free. Additionally, type designs for typesetting text instead of display use are functional works. (Stallman 2006)

———-move above———

but the vector 'source' has 'source', and that has source (scans or drawings). but it isnt *neccessary* to *have* the source scans or drwaings, or even the guides and so on [says gerard U] but it is neccessary to have the source for the 'logic,' the small scripts and large editor programs that generate the font binaries.

if you have source code, you may be technically able to copy and modify but are you free to do so, and is doing so susytainable business?

Fonts are a particular kind of bitstream that encode typeface designs. Are they programs or data?

As Lisp shows, there is not neccessarily a hard distinction, and being able to treat them interchangably can be powerful. ————

Unfortunately powerful font formats have been rejected, because like Lisp, expressing programs or typefaces in these unusual ways requires unusual ways of thinking: From Metafonts programming language for describing type designs

to generate pure outline data, to command line programs for interacting with state automata in AAT. Not just usability of tools but also way of thinking.

OpenType and VOLT, by making programming a visual activity, cracked it. MetaFont needs an interactive visualiser to make it usable; f berlaans kalliculator. We need a tool that complements FontFroge, like XGridFit, to generate OpenType layout programs visually like VOLT.

Longer term, we need visualiser for Graphite layout programs, because Grpahite is free software and developed in a community process, not an immutable standard that will not accept community improvements. [gww fea file email] and may change without notice and break existing programs. [robofog story above]. [is graphite developed in acommunity? need to push gww to do it and sil be more interested in his work :)]

in counterpunch the punchcutter role became divorced from type design (also in loc) but initially fonts reunited the roles. but as fonts become more programmatic, they are separated again.

References

Graham, P. (2001), The roots of lisp.
URL: *http://www.paulgraham.com/rootsoflisp.html*

Grier, D. A. (2005), *When Computers Were Human*, Princeton University Press.

Haralambous, Y. (2007), *Fonts & Encodings*, 1 edn, O'Reilly Media.

Noordzij, G. (2005), *The Stroke: Theory of writing*, Hypen Press.

Riordan, M. & Hoddeson, L. (1998), *Crystal Fire: The Invention of the Transistor and the Birth of the Information Age*, 2 edn, W. W. Norton & CompanyW. W. Norton & Company.

Siegel, D. R. (1985), 'The euler project at stanford'.

Smeijers, F. (1996), *Counterpunch: Making type in the sixteenth century, designing typefaces now*, 1 edn, Hypen Press.

Smeijers, F. (2003), *Type now*, 1 edn, Hypen Press.

Stallman, R. (1992), Why software should be free.
URL: *http://www.gnu.org/philosophy/shouldbefree.html*

Stallman, R. (2006), Copyright versus community. Transcription of a speech by Paolo Predonzani.
URL: *http://www.predonzani.com/stallman/stallman.html*

The Computer Language Company (2007), Computer desktop encyclopedia.
URL: *http://www.techweb.com/encyclopedia/defineterm.jhtml?term=software*

van Blokland, E. (2007), Introduction to the basic features, 2 axes , 4 masters. Promotional screencast for the program that introduces its core concepts.
URL: *http://superpolator.com/movies/SuperpolatorScreencast_intro_small.mov*