WINTER 2012: ENG*265*1: ESSAY 4 (FOCUSED PIECE)
MINIMIZING BADNESS: TEX AND THE PURSUIT OF QUALITY

JONATHAN RASCHER

9 MAR. 2012

Donald E. Knuth, 72-year-old professor emeritus of computer science at Stanford University, has not had an email address for over twelve years. "Email is a wonderful thing for people whose role in life is to be on top of things," he writes, "but not for me; my role is to be on the bottom of things." Knuth claimed his place "on the bottom" of computer science long ago. In 1962—before he renounced email, before email had been invented for him to renounce—the young computer scientist set out to write a book on compiler design. While the earliest computers were programmed exclusively in machine code (the indivisible instructions executed by a computer's central processing unit), almost all modern programs are written in a high-level programming language. These languages are more expressive than machine language, sparing programmers the almost unbearable tedium of writing machine code by hand. Yet computers cannot directly execute high-level code. Instead, an intercessory program called a *compiler* must translate human-friendly high-level code into computer-consumable machine instructions. Compilers are perhaps the most complex type of program, and the most fundamental; much early computer science research focused on improving their capability, correctness, or efficiency. A forerunner in his field, Knuth worked with excellent diligence and attention to the smallest details, developing compilers while completing his doctoral studies. "You've got to see the floor before you build the ceiling," he later explained. When Addison–Wesley requested that he write a compiler textbook, Knuth accepted without hesitation.

Four years later, that book remained unwritten. A man whose perfectionism matched his already celebrated skill, Knuth was not at all pleased by the computer science textbook industry of the 1960s. Ever on the bottom of things, Knuth set out to write a computer science reference worthy of introducing his compiler textbook. Computer science sits at the intersection of theory and practice, where mathematical abstractions find serendipitous application in engineering. This tends simultaneously to disturb the engineers, who view math only as a tool of their trade, and to upset the mathematicians, who value these abstractions for their own intrinsic beauty, frequently disassociating themselves from applied fields. Knuth desired to mend that gap, creating a unified, definitive survey of computer science. Acting as a sort of "reverse compiler," Knuth endeavored to rewrite low-level mathematical details in a form suitable for serious programmers and researchers alike. As he explained in an interview with fellow programmer and author Peter Seibel:

> I'm starting out with stuff that's in math journals that is written in jargon that I wouldn't expect very many programmers to ever learn, and I'm trying to dejargonize it to the point where *I* can at least understand it.... Computer science doesn't all boil down to a bunch of simple things.... If I didn't write the books it would be much harder for people to find stuff out. That's what turns me on.

Forty-eight years, countless computers, and one Internet later, Knuth's computer science reference—the seven volume *Art of Computer Programming*—remains incomplete. The first three volumes are currently in print, the fourth slated for future release. Knuth's meticulous writing process has required him to revise published volumes while also completing the rest of the series. In 1976, the second edition of Volume 2 awaited publication; however, galley proofs of that edition horrified Knuth. The first edition had been set in hot metal type on a Monotype machine, but his publisher had since transitioned to phototypesetting techniques. The fledgling phototypesetting system produced distorted text with inconsistent spacing, proving particularly troublesome for the mathematical notation of Knuth's *Art of Computer Programming*. "I had spent 15 years writing those books," he later reflected, "but if they were going to look awful I didn't want to write any more. How could I be proud of such a product?" Yet Knuth knew that hot metal typesetting was abandoned for good reason. Computers could already do the job faster and cheaper—why could they not do it *better*? Ever on the bottom of things, the long-suffering Knuth saw only one option.

Knuth took up work on the typesetting tool TEX[1] in 1977, thinking the task would take just a few months. A few months later, the program remained unfinished. Knuth's perfectionism got the

[1]Pronounced to rhyme with "speck," not "vex." The lowered letter E follows Knuth's personal typographical practice.

best of him once again; as he noted in his book *Digital Typography*, he had set "a new personal record for being too optimistic." Not until the 1989 release of TEX 3 did Knuth declare TEX to be frozen, with new versions focused only on fixing bugs. Above all else, the fastidious Knuth valued *quality*, wanting his typesetting program "to produce documents that were not just nice, but actually the best." According to Knuth, laying out plain text was not too difficult (mathematics was another story), but typeface design proved much harder. Wanting to spare others the problems he faced with his *Art of Computer Programming* reissue, he sought to "create systems that would be independent of changes in printing technology as much as possible." To this end, he developed a novel means of making digital type—a companion program called METAFONT[2]. A "meta-font," according to Knuth, is "*a schematic description of how to draw a family of fonts*, not simply the drawings themselves." Glyph shapes are defined using equations written in terms of designer-specified parameters. By varying these parameters, a single METAFONT character definition can generate glyphs of various styles, relieving type designers of the need to shape, e.g., roman and italic glyphs separately. Knuth used METAFONT to design the elegant Computer Modern typeface, now the first choice for mathematical books and articles. By codifying font design in a mathematically precise format—and by freezing his typesetter's fundamental feature set—Knuth secured TEX's place as an archival format. A TEX document retains its precise visual appearance each time it is compiled, be it today, a week from now, or in a hundred years.

METAFONT allows digital fonts to match or even outshine their handsome hot-metal forefathers, but stylish typography requires far more than faithful reproduction of pleasant letter forms. Knuth optimized TEX for one purpose: preparing mathematical material for publication. Traditional mathematical typesetting was a multi-step process—so laborious that typesetters called mathematical notation *penalty copy*, referring to the additional fees charged for such difficult work. For example, to set the probability density function for a normal distribution,

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(x-\mu)^2}{2\sigma^2}},$$

[2]As with TEX, the quirky typography of METAFONT reflects Knuth's own usage.

find effect
find effect

FIGURE 1. Text set with ligatures (left) and without (right)

a hot metal typesetter must compose the mathematical symbols one section at a time, measuring, aligning, realigning until the desired result is achieved. But a TeX user needs only to type the following code—infinitely more convenient, if not especially elegant:

```
\[ f(x) = \frac{1}{\sqrt{2 \pi \sigma^2}}
   e^{\frac{-(x - \mu)^2}{2 \sigma^2}} \]
```

TeX excels at mathematical spacing. Knuth agonized over the mathematical journals and books he found most beautiful, finally settling on a few basic spacing rules sufficient to set the majority of equations. And TeX includes other features common in professional typesetting, but infrequently implemented in consumer word processing packages. From its inception, TeX has supported ligatures, combinations of two or more characters into a single glyph to reduce the aesthetic unpleasantness of juxtapositions like f followed by i (Figure 1). Likewise, TeX has long supported kerning—the reduction of unsightly space between characters like V and A, which appear too distant unless their bounding boxes overlap slightly (Figure 2).

But TeX's primary worth lies neither in subtle typographical adjustments nor in superior mathematical spacing. TeX documents derive beauty chiefly from something that seems almost too simple to mention: line breaking. Conventional word processing packages implement line breaking using a so called *greedy algorithm*. Greedy line breaking places words until no room remains. At this point, if the word processor supports hyphenation, a word might be broken across two lines; otherwise, the line is complete. If full justification is enabled, the line's inter-word spacing is uniformly increased so the first and last words sit flush against the margins. Fast, concise, easy to implement—greedy line breaking yields terrible results[3]. The printed page looks best when the text-to-paper contrast ratio remains consistent throughout; whitespace should be similar from line

[3]An exercise for the interested reader: Reproduce the first paragraph of this essay in a typical word processor. Examine the result, counting the total number of hyphenated words while studying the contrast between text and paper.

VACANCY
VACANCY

FIGURE 2. The sequence "VA," kerned (left) and unkerned (right)

to line, inter-word spaces keeping close to their natural length. And hyphenation must be used sparingly. Excessive hyphenation interrupts the reader, the conscious mind put on hold while the eye scans left, searching frantically for the broken word's other half.

Acceptable line breaking must take not just a single line into account, but several lines at a time. When setting a paragraph, TEX considers all possible breakpoints at once; additionally, hyphenation points are included where appropriate, and hyphenation penalties are added to discourage TEX from breaking words unnecessarily. For each possible combination of line breaks and non–line breaks, TEX sums inter-word whitespace and hyphenation penalties, computing what Knuth named the *badness* of the resulting paragraph. Having examined all possible combinations of breakpoints, TEX selects the least undesirable combination, typesetting the paragraph with minimal badness. A naïve implementation of Knuth's line breaking algorithm would be exponential time (and thus intolerably slow for long paragraphs), but, under Knuth's direction, graduate student Michael Plass used a clever dynamic programming technique to ensure reasonable running times. Knuth achieved efficiency in software that he could not capture in writing.

His ten-year break from *The Art of Computer Programming* completed, Knuth froze TEX development in 1989. "Typography is not my life's work," he explained in a letter to type designer Hermann Zapf:

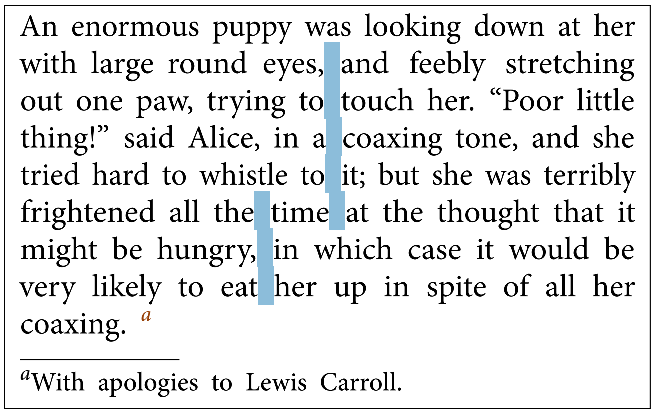
> I am primarily an educator, doing and guiding research in computer science and mathematics, and writing books that attempt to bring some unity into those subjects. Meanwhile, while solving a problem related to the publishing of such books, I seem to have stumbled onto some ideas of value in the printing industry, so I want to make sure that I have explained these ideas properly and gotten them off to a good start. If the ideas have merit, they should survive without my pushing for them; if not, they should die anyway.

Ever on the bottom of things, Knuth laid the foundation, framed the walls, wired the sockets. Should his construction prove sound, others could furnish the rooms, decorate the hearth, tend the garden.

TeX, METAFONT, the Computer Modern typeface—Knuth placed the source code for all his typesetting programs in the public domain. TeX's capabilities continue to exceed those of commercial typesetters; moreover, TeX is free to use, free to read, free to modify. Free to "hack," as computer scientists say. Such "hacks" have helped TeX remain relevant, allowing it to survive thirty years in a software scene where obsolescence often occurs after six months. Yet no software can attain perfection, even the product of such a perfectionist as Knuth. The mathematical spacing rules TeX implements do not always give optimal output. TeX's math spacing algorithm has recently been extended, but the improvements have not been implemented in TeX itself. Thus software may soon exist that sets math more beautifully than TeX, if only slightly.

Knuth wrote TeX to be almost completely programmable, allowing TeX users to create libraries of prepackaged code that render elegant tables, scalable graphics, and many things besides. But programming in TeX is much like visiting the DMV: Sufficient patience yields results, yet one can't help but wonder if there isn't an easier way. Knuth implemented programmability via a macro language, an arcane type of programming language suitable for 60s-era hardware, but tedious and inexpressive compared to modern alternatives. Knuth has always been forthright about TeX's limits. "Dogs can stand on their hind legs," he reflected, "and TeX can calculate prime numbers," but neither action comes naturally; both the clever dog and the clever TeX coder soon grow weary of their tricks.

Even the defining feature of TeX cannot altogether avoid criticism. Working to minimize the badness of typeset text, the Knuth–Plass line breaking algorithm successfully reduces inter-word spacing, avoiding unnecessary hyphenation and effecting uniform contrast across paragraphs. But TeX's line breaking algorithm fails to eliminate vertical or near vertical columns of consecutive spaces, called *rivers* by typographers. These rivers are visually jarring (Figure 3), misleading the eye, emphasizing the vertical over the horizontal, slowing readers, inhibiting comprehension. Attempts to add river avoidance to TeX have failed, the TeX line breaking algorithm too specialized, perfection still sacrificed at the altar of efficiency. Eliminating rivers requires an utterly new method of breaking lines—and that would require a substantial restructuring of TeX. The house that Knuth built still


An enormous puppy was looking down at her with large round eyes, and feebly stretching out one paw, trying to touch her. “Poor little thing!” said Alice, in a coaxing tone, and she tried hard to whistle to it; but she was terribly frightened all the time at the thought that it might be hungry, in which case it would be very likely to eat her up in spite of all her coaxing. [a]

[a]With apologies to Lewis Carroll.

Figure 3. Rivers of vertically flowing whitespace

stands, its foundation solid, but the ravages of time begin to show—shingles long lost to the wind, the once airtight doorjambs now drafty, the wiring no longer up-to-code.

Volume 4 of *The Art of Computer Programming* awaits publication; Knuth is 74. He expects to release Volume 5 in 2020; Knuth will then be 82. “After Volumes 1–5 are done, God willing,” he plans to publish Volumes 6 and 7, provided that topics they would discuss remain relevant. Maybe he will write these volumes—Knuth’s health is good for a man of his age—but it seems doubtful. Time inevitably ensnares the creator as well as the created. Driven by perfectionism, Knuth has walked from project to project. He never wrote his compiler textbook; he will not likely finish his definitive computer science reference; he almost certainly will not live to see TeX’s shortcomings remedied. TeX itself took ten years to build under the benevolent virtuoso’s direction; surely TeX’s replacement shall be even slower in coming.

Yet if Knuth has any major regrets, they are well hidden. In interviews and lectures, Knuth seems satisfied with his life’s work. Knuth possesses the essential trait of all true artists, a peace that passes all understanding, a satisfaction deriving from single-minded dedication to what philosopher Robert Pirsig calls Quality. In his reconditely lucid book *Zen and the Art of Motorcycle Maintenance*, Pirsig explains that Quality cannot be defined, yet all can know it through painstaking, perpetual self-comparison. Not merely an attribute of art, Quality *creates* art, unifying the objective and the subjective, giving beauty and vitality to the most mundane things—even to a piece of software, even to a line breaking algorithm. In Knuth's obsession, he finds release. "I wake up in the morning with sentences of a literate program," he once stated. "Before breakfast—I'm sure poets must feel this—I have to go to the computer and write this paragraph and then I can be happy." Quality—that ethereal entity which makes authors write and auteurs film, poets play and builders erect—effected every aspect of Knuth's work, down to the mildest, meanest, most diminutive detail.

By society's standards, Knuth is a failure, an idealist to be pitied as much as admired. So grandiose are his goals that a hundred score of men could not achieve them. Knuth remains focused as the world races around him, computer science evolving far faster than he can document it. His perfectionism frequently renders his work outdated as well as unfinished, requiring revisions which further impede completion. Yet by the standard of *Quality*, Knuth has done all that anyone can do. Neither man nor machine can completely eliminate badness; the problem is too computationally expensive—intractable, as computer scientists say. But badness can be minimized—in print by clever programs, in life by clever people. Quality works to limit badness, rendering some small part of human existence indisputably better. And it begins with people like Knuth, people whose care for and dedication to their work knows only the inevitable bound of a human lifetime. "Care and Quality," as Pirsig asserts, "are internal and external aspects of the same thing. A person who sees Quality and feels it as he works is a person who cares. A person who cares about what he sees and does is a person who's bound to have some characteristics of Quality."

Since 1989, TEX's version number has been converging to π. Each release adds another significant digit; version 3.1415926 came out in 2008. The last change to TEX, effective upon Knuth's death, will set the version number to its final value: precisely π, transcendent as TEX itself. From that time on, "all 'bugs' will be permanent 'features.'" Inevitably imperfect, TEX nonetheless approximates perfection, striving never to eliminate badness, always to minimize it. Ever on the bottom of things, Knuth must eventually sink to the lowest bottom of all, falling into the sleep from which none can wake while this world stands. But Knuth's documents shall remain—enduring treatises, mathematical books and papers, mellifluous prose beautifully set. Quality breeds quality; good type gives good text. And TEX too will linger, immutable yet evolving, its elegance admired, its innovations studied, its flaws fixed by future software—for Knuth's work is a work of Quality, and Quality never can perish.

References

[1] "Donald Ervin Knuth." *MacTutor History of Mathematics*. Sep. 2009. Web. 2 Mar. 2012. <http://www.gap-system.org/~history/Biographies/Knuth.html>.

[2] Eijkhout, Victor. *TEX by Topic: A TEXnician's Reference*. Raleigh: Lulu, 2008. Print.

[3] Knuth, Donald E. "The Art of Computer Programming." *Stanford Computer Science*. n.d. Web. 4 Mar. 2012. <http://www-cs-staff.stanford.edu/~uno/taocp.html>.

[4] Knuth, Donald E. *Digital Typography*. Stanford: CSLI, 199. Print.

[5] Oetiker, Tobias, Hubert Partl, Irene Hyna, and Elisabeth Schlegl. *The Not So Short Introduction to LATEX 2ε: Or LATEX 2ε in 157 Minutes*. Version 5.01. 6 Apr. 2011. Web. <http://tobi.oetiker.ch/lshort/lshort.pdf>.

[6] Pirsig, Robert M. *Zen and the Art of Motorcycle Maintenance: An Inquiry into Values*. New York: HarperTorch, 1999. Print.

[7] Seibel, Peter. *Coders at Work: Reflections on the Craft of Programming*. New York: Apress, 2009. Print.

[8] Wilson, Peter. *A Few Notes on Book Design*. Normandy Park: Herries P, 2009. Web. 26 Feb. 2012. <http://mirror.ctan.org/info/memdesign/memdesign.pdf>.