

The electric typesetter: The origins of computing in typography

John Labovitz

Standard history places the fusing of typography with the computer circa 1985, with the revolution of desktop publishing. Until then, as the story went, typography and printing had marched slowly and tediously forward from Gutenberg's first page of metal type. Typesetting became mechanical, then photochemical, and printing moved on from the hard relief of letterpress into the softer kiss of the offset press. But the process of production was essentially manual, performed by a team of specialists — typographers designed the layout, typesetters set the type, and layout artists assembled the final elements. This all changed upon the heroic arrival of the Macintosh desktop computer and PageMaker page-layout software, with reinforcements from the Laser-Writer desktop laser printer and the Linotronic imagesetting machine. Carrying a gleaming, digitized manifesto of graphical user interfaces and device-independent PostScript, the visionaries of desktop publishing scoffed at our dark past, damning those sad days of bluelines and bromides, the foul smells of molten wax and lead, the inconvenience of doing things by hand.

The story was compelling. The personal computer was cast in the role it has played so often since: a magical machine that could replace all previous toil with glorious creativity. By using a computer, everything could change for the better, overnight. But in reality, the revolution was more of an evolution, resting on a foundation established over many centuries in which computing was already part of the typographic scene.

Typography has always implied computation. Typography works with symbols, codes that represent the alphabets of our languages, and standardized methods (justification, pagination) that build formal aesthetic structures (lines, columns, pages). Even a printer's composing stick implements an algorithm: the sum of the widths of all the assembled letters of metal type, plus the sum of all the spaces, must equal the final measure of the line, and vice versa. By thinking expansively about typography as a computational act, the purpose of this essay is to consider the convergence of changes within mathematics, computing, and typesetting that made desktop publishing possible. In short, to look at the numbers behind the letters.

Before digital computers, generations of mechanical computers evolved over millennia. Their increasing precision and power coincided with the desire for automatic typesetting systems, and merged to create single-purpose mechanical typesetting computers like the Linotype and Monotype. The rise of digital computers brought not only speed and efficiency to calculations, but expanded coding systems, which made treating textual information as easy as numeric data. At the same time, phototypesetting machines evolved out of the older mechanical systems. As each type of device left behind its fixed-purpose, mechanical roots, the plasticity of the programmable world, and the advantages of fast digital communications, led to symbiotic systems that took over more and more functions of traditional typography.

Meanwhile, the emerging field of computer graphics invented new formal models into which typography could be integrated more holistically. Display screens and mice supplanted keyboards and punch cards, and a quickly advancing rate of computing power allowed the human-computer

interface to be metaphorical. By sufficiently simulating the old ways of typographical production, but in a sleek new package, desktop publishing was able to push out the old leaders and allow a much wider audience to teach themselves elements of publication that were once a mysterious art.

Why compute?

Today, typography is as intertwined with computers as anything else in our hyperconnected modern lives. For most people who work with type, imagining typography without computers implies a full retreat to hand-set metal type, as if computerized typesetting was born fully formed in the late 20th century. But the computerization of typography was only part of several much larger and far lengthier industrial and social processes.

The first and likely largest influence was automation — building machines to do work once done by humans. In the 19th century, typography was only one of many crafts and practices made subject to the god of automation. By minimizing the work done by hand, businesses could save time and make more money.

The second influence, coming in the midst of the 20th century, was the information revolution. Gutenberg’s innovations in communication technology were exponentially carried forward in 1948 by Bell Telephone Laboratories, who announced the invention of the transistor, a small electronic semiconductor, and “A Mathematical Theory of Communication,” that would make our current electronics possible.¹ Today’s Internet cloud only continues the focus on data creation that started decades ago. As Eric Schmidt, CEO of Google, has stated, “Every two days now we create as much information as we did from the dawn of civilization up until 2003.”²

With new models of generating, storing, revising, and transforming data, typography changed from a mode of handcrafting to a mode of processing. Whether for printed phone books and newspaper classified ads, or the Amazon global superstore, typography now serves the god of data. Given the speed and bulk of data production today, the underlying technology of typography has changed significantly. The external appearance of changing technology is obvious — say, mechanical to optical to digital — yet each of these changes necessarily effects and requires many other technological changes. To represent all the world’s languages (whether natural, machine, or Emoji) flung out from today’s firehose of networked data, we use the ever-evolving Unicode universal encoding system. To properly shape characters into appropriate font glyphs, we use the OpenType font format with its internal programmable rule engine. To format text for the screen, we use a variety of algorithms that perform all the traditional tasks of a human typesetter — line-filling and justification, hyphenation, spacing, and pagination. And because so much of our information is sourced from the network, there is rarely a definitive text. A web

¹ James Gleick, *The Information: A History, A Theory, A Flood* (London: Fourth Estate, 2011) pp. 3–4.

² Eric Schmidt, panel presentation at Technomy conference (Lake Tahoe, CA, 2010), <http://www.historyofinformation.com/expanded.php?id=3807>

browser's layout component must autonomously composite dozens or hundreds of pieces of data, in a multitudes of formats, in real-time, with no previous knowledge of the text.

The nature of computing

Before we head down the trail of history, we should define what computing means. Computation, of course, means performing calculations on numbers. But how do those numbers arrive to be computed? What happens after they are computed? And how do we represent and process ideas other than numbers, such as the glyphs of a font?

There have been computers that were very unlike the ones we now have on our desks, or carry in our pockets. Each technology has an *architecture* — its basic makeup, working process, and characteristics. A modern digital computer is electronic (using transistors in silicon), represents data as binary (base-2) numbers, stores data temporarily in RAM or permanently on a hard drive, displays data to a screen grid of lighted pixels, and is connected (via the Internet) to billions of other similar computers. A mechanical computer, on the other hand, is perhaps made of steel and brass, uses interlocking gears, pulleys, shafts, and cams, displays data using printed wheels and cylinders, and exchanges data with other devices using punched paper tape. Both are computers, but each has a unique architecture.

In order to communicate as humans, we represent our ideas as shared symbols or codes. For example, in English, the word “one” acts as a shared code that represents the numeric quantity of one. A digital computer, which does not operate in natural language, might represent that quantity using the binary code “0001.” When the computer is asked to display the quantity, it might convert the binary value to a font glyph with the code “1,” which then is drawn onto a screen using coordinates referring to a grid of pixels. All these codes exist in different *coding systems*, converted by many layers and levels of encoding, decoding, and transcoding of symbols. In computing, “data” is never really what it seems to be — it is always a reference to something else, whether coming or going.

In order to reference, dereference, or transform these codes, a computer must have storage — a workspace of sorts, where its operations take place. This can be temporary (the intermediate results of a calculation) or permanent (saving the text of a book for generations).

Once a computer has data, it can work with it. While a layman's explanation usually refers to a computer as a number-crunching machine, this is more of a relic and influence of the particular time when electronic computers were first introduced to the masses. Those early computers were indeed used for calculations (ballistic and otherwise), but at their deepest level, they operated by transforming values represented by codes.

Generally this transformation is known as *processing*. Processing can be very simple (adding two numbers together) or complex (rendering the pixels needed to display a glyph). Either way, processing is described by *algorithms*, which are a bit like recipes: descriptions of ingredients (data) needed to accomplish a task, and the general steps taken to do that transformation. For example, a simple algorithm to justify a line of text is:

1. Calculate the sum of the width of all the words in the line.
2. Subtract the sum from the desired line measure to calculate the extra space needed to justify the line.
3. Divide the extra space by the number of words minus one.
4. Add the result as extra space between each word.

The *implementation* of algorithms is always dependent on and directly linked to the technology's architecture. On a mechanical computer, the justification algorithm might be implemented using metal type as the input, interlocked gears for the calculation, and a metal-casting device that creates variable spaces. On a digital computer, the algorithm might be implemented using coded characters as input (along with a table of character widths), data structures that represent words and lines, a formula to calculate the word space, and a virtual pen that draws the glyphs to the screen. The algorithm stays the same, but the implementation is vastly different.

It is important to note that just as architecture affects the inward-looking design such as implementation of algorithms, it also affects the outward-looking qualities: how tools are used, and how things are built with those tools. Typography from the letterpress era often looks remarkably different than that of the desktop publishing era, which looks different still than that of the web era. Part of that is fashion and trends, but a possibly larger influence is the inherent nature (materials, models, constraints) of each technology.

The dawn of mechanical computing

Although digital computers did not appear until the 1940s, the use of machines to perform computations likely started two millennia before. Common knowledge of computing posits anything before the 1940s as simply theoretical or experimental. However, computing is not a specific technology, but a process or method, as the following examples show.

Perhaps the earliest is the abacus, which first appeared around 200 BC. By manipulating beads along rods, one could perform simple counting or complex arithmetic. Results could be stored for a short while, until the user “cleared” the device’s memory by shaking it. Since the method of using the abacus was standardized, it could also be used for communication (if you knew how to decode the results). Although it lacked automation, the abacus still served as a powerful device that took computation out of the human mind and allowed for abstract thought.

Around the time of the abacus, the Antikythera Mechanism³ was created to predict astronomical positions and eclipses. By setting positions with gears, and turning a crank to advance the calculation, its user could track the cycles of the solar system.

The field of mechanical computation lay dormant for quite some time, but was revived in the 1600s with the development of several calculation machines, starting with Wilhelm Schikard in

³ Antikythera Mechanism Research Project, <http://www.antikythera-mechanism.gr>

1623, Blaise Pascal in 1642, and Gottfried Wilhelm Leibniz in 1680.⁴ Mathematicians of the time often used tables of pre-calculated numbers, and the mechanical calculators were intended to compute, verify, or expand those tables. As Leibniz said,

[I]t is unworthy of excellent men to lose hours like slaves in the labor of calculation which could safely be relegated to anyone else if machines were used.⁵

The labor of calculation was as much a problem of computation as it was of printing. From around 1822 into the 1860s, Charles Babbage invented a series of mechanical computers. Some were built to solve specific equations, and others acted as general-purpose computers, using principles of storage and programming, not unlike today's computers. Because Babbage was concerned with producing accurate tables of numbers, he devised not only ways of calculating those numbers, but methods of reproducing them in print. His Difference Engine had mechanical wheels on which were mounted digits of metal type:

The next step was to devise means for printing the tables to be completed by this machine. My first plan was to make it put together moveable types. I proposed to make metal boxes, each containing 3,000 types of one of the ten digits. These types were to be made to pass out one by one from the bottom of their boxes, when required by the computing part of the machine.

[...]

Another plan for printing the tables, was to place the ordinary printing type round the edges of wheels. Then, as each successive number was produced by the arithmetical part, the type-wheels would move down upon a plate of soft composition, upon which the tabular number would be impressed. This mould was made of a mixture of plaster-of-Paris with other materials, so as to become hard in the course of a few hours.⁶

Consequently Babbage's Difference Engine could be looked at as a specialized computerized typesetter, foreshadowing the ideas of typesetting directly from dynamic data.

Automatic typesetting

During much of the 19th century, there was a great rush to create automatic typesetting systems that would free human typesetters from the drudgery of selecting, setting, and redistributing type. The race started with William Church's first experimental machines in 1822, and progressed in both complexity and success over the next hundred years or so.⁷ Many dozens of typesetting ma-

⁴ Herman H. Goldstine, *The Computer: from Pascal to von Neumann* (Princeton University Press, 1972), pp. 6–9.

⁵ David Eugene Smith, *A Source Book in Mathematics* (New York: McGraw-Hill, 1929), p. 181.

⁶ Charles Babbage, *Passages from the Life of a Philosopher* (London: 1865; reprinted 1968), pp. 44–46.

⁷ Richard E. Huss, *Dr. Church's "Hoax"* (Lancaster: Graphic Crafts, Inc., 1976).

chines were designed and patented, some built, a few marketed, a few bankrupted, but in the end, the winners were the Linotype, the Monotype, and to a lesser degree, the Ludlow Typograph.⁸

The automatic typesetters were incredibly complex devices, built of thousands of pieces of precision-machined iron, steel, and brass. They were the epitome of mechanical engineering of the period, arguably even to our present day. However, most of these machines primarily dealt with the problems of type selection, assemblage, and distribution. Only a few tackled the problem of justifying lines of type, and hence were more akin to clockworks than to computers. In 1887, Tolbert Lanston, a lawyer and inventor in Washington, DC, patented a machine he called “The Embossing Type-Maker.”⁹ This appears to be the first mechanical typesetter to use principles of computation.

The Embossing Type-Maker wasn’t Lanston’s first foray into mechanical computing. In 1894, while working as a clerk in the US Pension Office for Herman Hollerith, Lanston had patented a statistics-oriented, keyboard-driven adding machine.¹⁰ Later, Lanston shifted his interest from calculators to typesetting machines, perhaps due to his brother, Walter Lanston, being a printer.¹¹ Lanston continued with his typographical pursuits, going on to invent the Monotype system in 1890.

How is it that computation was integrated with typography? It seems to be related to the curiously viral nature of computing. As Goldstine says,

[T]he development of radical new machines always comes about because some inspired person sees how to adapt a new technology to computers and thus to make a major advance in the state of the art. To be precise, it is usually the convergence of two very different concepts. One is the technology, but the other is the recognition of the importance and necessity for an advance.¹²

The necessity of recognition is not so different from the combination of knowledge about woodcarving and metalworking, among other skills, that converged to give rise to the use of moveable type within printing presses centuries earlier.

As an exercise, looking at a Monotype Composition Caster through the lens of digital computing illustrates several prescient ideas that continue into the present day. The Monotype keyboard

⁸ Keith Houston, “Out of Sorts: typesetting meets the Industrial Revolution” in *The Book* (London: W. W. Norton and Co., 2016), pp. 128–152.

⁹ Richard E. Huss, *The Development of Printers’ Mechanical Typesetting Methods: 1822–1925* (Charlottesville: The University Press of Virginia, 1973), p. 127.

¹⁰ Tolbert Lanston, *Adding-Machine* (US Patent No. 622,157, filed May 25, 1894, and issued March 28, 1899).

¹¹ Richard L. Hopkins, *Tolbert Lanston and the Monotype: The Origin of Digital Typesetting* (Tampa: University of Tampa Press, 2012), p. 18.

¹² Herman H. Goldstine, *The Computer: from Pascal to von Neumann* (Princeton University Press, 1972), p. 11.



was, not surprisingly, used to enter text. As an operator keyed in copy, the key codes were punched onto the paper tape, along with spacing information. Rich Hopkins describes it well:

In addition to capturing keystrokes, the Monotype keyboard was a precise calculator. Each time a key was pressed, the keyboard registered the number of units involved with that character. When a space was tapped, a minimum width of 4 units was recorded along with special space codes. Approaching the end of the line, the operator was alerted by a bell to find a stopping point. Thereupon, the operator was informed of how many units were needed to fill the line. Simultaneously, a table was presented to the operator providing specific information for setting to justifying wedges [on the caster] which, in turn, would increase the width of word spaces precisely to spread the line to its desired width.¹³

¹³ Richard L. Hopkins, *Tolbert Lanston and the Monotype: The Origin of Digital Typesetting* (Tampa: University of Tampa Press, 2012), p. 13.


In other words, the Monotype employed the basic concepts of computing in mechanical form: coding (both keys and spacing information), processing (calculation), and storage (paper tape).

The birth of digital computers

Meanwhile, computing and data processing were evolving quickly. Starting in 1880, the data from the US Census had been translated to a coded form using punched paper cards. Tabulating machines could then pick through cards based on certain criteria, calculate totals, and produce reports of those numbers printed directly from the machines.

Gradually, mechanical computing gave way to analog electrical computing, and then to the digital electronic computer. The first successful general-purpose electronic computer was ENIAC, built in 1946. From this point, the computer industry exploded, with IBM and other manufacturers rapidly producing new machines. The ENIAC and other early computers were programmed by hooking up wires. It could take days to reconfigure the machine for another task. Because input was constrained to switches and dials, computing tasks tended to be value-oriented — generally, numbers.

Once the tedious “hard-wired” method was replaced by the ability to read and write data and programs to magnetic tapes, punched-paper tapes, and punch cards, the computers became more general-purpose. Keyboards and printers replaced switches and lights. Programming became far more flexible, faster, and easier. New concepts like operating systems, files, data formats, and programming languages were introduced. It was becoming clear that computers were not just about pure calculation. The previous focus on numbers evolved into general data processing, including text.



```
3;      ZEROJ =EOM14G; J3C14NZ;
      , , , , , ,
EXIT ;

225;    SETAY-2; SETAYDOJ JS224; (CLEAR BELOW PROG);
      , , , , , , , , , , , , , , , ,
DUP; SET AY1; Y2; +; JS224; (CLEAR ABOVE PROG);
      , , , , , , , , , , , , , , , ,
226;    DUP; SET36; -; SETAYO; -; =C1; (SET XS);
      , , , , , , , , , , , , , , , ,
DUP; SET 1; +; =M15; =MOM15;
      , , , , , , , , , , , , , , , ,
ZEROJNOT;=VIP295;ZEROJ=VY2P295; (INIT BUFFERING);
      , , , , , , , , , , , , , , , ,
Y-1; Y-2; SHLD-24; =E2; =E3;
      , , , , , , , , , , , , , , , ,
VOP299; JSP293;
      , , , , , , , , , , , , , , , ,
EO; SET B20; AND; J227=Z; (IF LOAD=AND=GO);
      , , , , , , , , , , , , , , , ,
JSP126; (READ PROG);
      , , , , , , , , , , , , , , , ,
```


The world as text

There is a good reason that text was adopted into computing so early. The invention of the electric telegraph had happened two centuries earlier, and what is the telegraph but a system to encode and decode text?¹⁴ In the intervening time, Morse and many others had developed sophisticated methods of efficiently coding, storing, and transmitting text. Most of these methods involved various combinations of electrical pulses — the dots and dashes were easily adapted to the binary 0 and 1 used by the digital computers.¹⁵

Once text arrived on the computing scene, it became the dominant method of interacting with a computer for nearly fifty years. Using keyboards that punched paper tape or cards, programmers and data entry operators fed the computers text: natural language, programming languages, system commands, even columns of numbers. Printers spit out listings and reports inelegantly, in monospaced capital letters.

As computers were still quite limited in processing capacity, data storage, and input/output, software was often built as special-purpose tools that accomplished a particular task, but worked together with other tools that could be executed at a later time.

For example, a text-formatting program might read a stack of punch cards containing textual data and formatting commands, process the data and commands, and save its output on additional cards or tape. Then another tool would take the formatted text and send it to a printer or phototypesetter. Large-scale, comprehensive, interactive software packages did not appear in mainstream computing until the 1980s. This method of handling data is generally referred to as “batch mode” or “batch processing.”

Text processing, including typesetting, was an interest in computing as early as the 1950s. Computer scientists of the time proposed methods of automating various processing of text, such as lexical analysis, language translation, revising manuscripts, sorting, indexing, and automatic hyphenation.¹⁶

Phototypesetters take the stage

Although phototypesetting had been in development since the late 19th century, the first machine on the market appeared to be a bizarre modification of the hot-metal Linotype. In Intertype’s Fotsetter, the usual brass matrices used for casting metal type were replaced with similar matrices

¹⁴ Tom Standage, *The Victorian Internet* (New York: Walker and Company, 1998).

¹⁵ Charles E. Mackenzie, *Coding Character Sets: History and Development* (Addison-Wesley, 1980).

¹⁶ W.P. Jasper, editor, *Advances in Computer Typesetting: Proceedings of the 1966 International Computer Typesetting Conference* (The Institute of Printing, 1967).

containing tiny pieces of film with the respective character in the given font, and the caster was replaced with a camera assembly which exposed each character onto a roll of film.¹⁷

The next generation of photo-mechanical typesetters started from scratch. They read codes from paper tape generated on a separate keyboard system. A entire font was stored on a photographic negative in a particular order. According to the data read from the tape, a mechanism acting somewhat like a photographic enlarger projected the particular glyph, at the desired size, onto a roll of film.

The Automatic Composing Machine

In 1954, Georges Bafour, Andre Blanchard, and Francois Raymond patented the first typesetting system that used electronic computing as its underlying mechanism. They called it the unlovely name of “Automatic Composing Machine,” but it was generally known as the BBR system.¹⁸ Just as the Monotype and Linotype had solved the first order of typesetting drudgery, namely the selecting and redistribution of type and of automatic justification, the BBR system intended to solve the next major problems on the list: automatic line-breaking and hyphenation. The system was intended either as a direct-input keyboarding station, or as a peripheral that took its input from a general-purpose computer via paper tape. In either case, it generated new paper tape that could then be fed to a phototypesetter or linecaster.

The BBR system also threw in a marvelous solution to the problem of revising and corrections. An operator could feed the system a tape of original text, and further tapes of corrections. All the tapes would be automatically merged into a single corrected version, and used as input. The BBR system was a specialized electronic computer, purpose-built for this task. It was not programmable, and had no software. Different circuits performed different parts of the typesetting task. If you were looking at the machine, you could have literally pointed to the circuit board that performed the hyphenation.

The BBR system apparently only existed as a prototype, but it sparked a great deal of interest in the possibility of having computers act as an intermediate agent between keyboarder and phototypesetter or caster.

Command and control

Although early phototypesetters appeared around the same time as the first digital computers, they were initially treated as unrelated technologies. Most early marketing material for phototypesetting systems show a Monotype-like configuration with a separate keyboard unit that punched paper tape, which, like a Monotype caster, would then be directly mounted on the phototypesetting unit. The tape could be stored for later reuse (to produce boilerplate text such as

¹⁷ Frank Romano et al., *History of the Phototypesetting Era* (San Luis Obispo: Graphics Communications Institute, Cal Poly, 2014).

¹⁸ G. P. Bafour et al., *Automatic Composing Machine* (US Patent No. 2,762,485, filed March 21, 1955, and issued September 11, 1956).

contracts, syndicated articles, or advertisements, for example), or easily mailed across the country instead of shipping heavy lead galleys or forms, but the model remained one of a human operator using a turnkey system.

However, it didn't take long to realize that computers excelled not only at pure computation, but also as controllers for mechanical devices. A programmer who knew the codes that governed a given typesetting machine would be able to write software to generate those codes. Initially, the bridge was built through punched-paper tape, a medium common to both computers and phototypesetters (as well as the popular teletype technology). There was even a nod to backwards-compatibility — the Monotype Paper Tape Conversion Unit was a device that read standard computer tapes and punched new tapes that could be fed into the now-elderly Monotype Composition Caster. Later, electrical-mechanical interfaces were created, making phototypesetters directly-connected peripherals, and formally bringing them into the new digital ecosystem.

During the 1950s and into the 1960s, large newspapers began to install computer-assisted typesetting systems from IBM and other manufacturers to handle composition of repeating or data-driven content like stock prices, classified ads, and directories.¹⁹ The *Los Angeles Times* and the *Mirror* as well as the *Post-Times* of West Palm Beach, Florida, installed RCA 301 computers, with others quickly following suit.²⁰

Evolution & experimentation

Starting in the 1940s, the US government encouraged the development of quasi-public research institutes, inviting collaboration between major universities like MIT, UC Berkeley, Carnegie-Mellon, and Stanford, corporations such as Xerox and AT&T, and government agencies including the Department of Defense.

The institutes attracted the kind of people who were curious about technology and what it could do. They acquired computers, and as part of their mission was to publish, they also acquired modern phototypesetters for utilitarian reasons.

The culture of the research institutes encouraged thinking well outside the box. Rather than obey the old saw, “If it ain't broke, don't fix it,” the researchers dove head-first into new ways of thinking about ideas and technology — even technology that seemed to be stable. As an informal slogan at Xerox PARC stated, “The best way to predict the future is to invent it.”²¹

¹⁹ Mary Elizabeth Stevens and John L. Little, *Automatic Typographic-Quality Typesetting Techniques: A State-of-the-Art Review* (National Bureau of Standards Monograph 99) (Washington, D.C.: National Bureau of Standards, 1967), p. 65.

²⁰ W. E. Blundell, “Printing Revolution: Research Push Brings Speedier Typesetting, Other Major Advances” (The Wall Street Journal, 10 December 1964).

²¹ Michael Hiltzik, *Dealers of Lightning: Xerox PARC and the Dawn of the Computer Age* (HarperBusiness, 1999), p. 122.

The printing business, especially at its peak in the 1960s, was highly regimented, whether by union or tradition. The trade schools required students to specialize in a particular sub-field: were you a typesetter, perhaps on a Linotype or Monotype if in a hot-metal/letterpress shop, or in a phototypesetter in an offset shop? Were you a prepress technician, making halftones and plates on a camera? Or were you a printer, spending most of your hours in front of a press?

The researchers, even the rare typographically savvy ones, were not going to play the game of master and apprentice, nor of specialization. Working outside the union system, researchers were able to take a long step back and look at the field as a whole. They were free — even encouraged — to break established rules of typography and design. This resulted in radically different thinking, and hence different solutions.

The corporations who produced typesetting equipment generally dictated how that equipment was to be used. This was accepted and standard in the printing industry, but became a conflict within the research institutes. The corporations' proprietary products were no longer being used the way they had intended. When researchers politely asked for information about how equipment worked, the corporations tended to dig in and refuse. The hackers retaliated the way they knew best: to disassemble and reverse-engineer the equipment, and claim it as their own.

A good example, though somewhat late in the game, was the experience of a group of programmers at Bell Labs, AT&T's research institute. Joe Condon, Brian Kernighan, and Ken Thompson were no ordinary coders: these were experienced computer scientists, and the inventors of the UNIX operating system, which now underlays much of modern computing.

In 1979, Bell Labs' Computing Science Research Center had purchased a Linotron 202, a phototypesetter sold by Mergenthaler, for its research on document preparation techniques. Like many phototypesetters of the time, the 202 had a built-in minicomputer. System and formatting software were stored on a paper tape (a storage format nearly obsolete by 1979) which had to be loaded into the phototypesetter each time it was turned on. Font data was stored on 8" floppy disks — then a hot new technology. Finally, a digital interface connected to other computers which would provide the actual text to be typeset. This was a strange hybrid beast.

The programmers had previously written their own typesetting software ("troff," mentioned later in this article), and wanted to use that software, in addition to fonts of their own design, to drive the 202. However, they immediately started running into problems. To start, the machine was incredible unreliable, taking the better part of a month of repairs to get to a basic working state. Documentation was either incomplete or incorrect. Then, as the programmers started to ask Mergenthaler for further documentation and technical assistance, it seemed the company did not want a rogue group of computer scientists trying to make the machine do things it was never intended to do.

Our main hesitation [...] was the secrecy surrounding the representation of characters, since from the beginning we intended to create our own. Repeated discussions with Mergenthaler representatives, both in the United States and in their head office in England,

made it clear that they would not divulge the principles of operation, not even under a trade secret or license agreement.²²

And so the programmers did what programmers are wont to do: they reverse-engineered the machine, not in malice or anger, but in curiosity and even a bit of joy. The programmers simply took what they had and figured out how it all worked, and how it could be made better. The programmers tweaked the mechanisms to be more reliable; they loaded the paper tapes on their own computers and decoded the programs there, uncovering the basic architecture of the built-in computer; they teased out the secret font formats from the floppy disks; and in the end, they completely replaced the built-in software with their own. Mergenthaler still owned the rights to the hardware, but the Bell Labs programmers had staged a coup and taken over the software.

This kind of reverse-engineering/re-engineering developed into a distinct culture of *hacking*. We now read this word as nefarious or criminal, but in the mid-1960s at MIT and other institutions, hacking was a respected sport, culture, and practice.²³ However, the lack of both formal training and respect for the traditional craft meant that few researchers really understood the nuances of typographic design. Little work from this era would be seen today as beautiful or lasting. Priorities were clearly different: experimentation and boundary-crossing were more valued than typographic finesse, and iterative development won out over final product.

Much of the technology we use today came out of those institutes, including the laser printer, image processing, the graphical/desktop interface, and networking. Unix, the operating system developed at Bell Labs, powers every Mac, iPad, iPhone, Android phone, and most of the servers on the internet.

Typesetting as software

With text being the dominant medium of digital computing, it was only natural that typesetting became an interest among computer scientists. As early as the 1950s, ideas and experiments in the field of computerized typesetting began to be formalized into specific software programs.

The earliest known text-formatting software, TJ-2, was created by MIT-trained computer scientist Peter Samson in 1963. Its design and architecture set the stage for text-formatting and typesetting programs for the next several decades. According to the the only extant description of the TJ-2 program,

TJ-2 accepts English text from typewriter or reader, and reproduces it at any line length via typewriter and/or punch [tape]. So much as possible both left and right margins are aligned in the output. To accomplish this the program doubles some of the spaces in the

²² Joe Condon, Brian Kernighan, and Ken Thompson, *Experience with the Mergenthaler Linotron 202 Phototypesetter: or, How We Spent Our Summer Vacation* (Bell Laboratories, 1980).

²³ Steven Levy, *Hackers: Heroes of the Computer Revolution* (New York: Anchor Press/Doubleday, 1984).

output line, and may hyphenate words, getting hyphenation data from its dictionary or from the operator via the display.²⁴

TJ-2 read lines of text as its input. Each line was collected and formatted to make justified paragraphs (of monospaced type). However, if a line started with a special control code (“overbar,” in the lingo of the PDP-1 system, its host computer), the program interpreted it as a command. There were only a few commands, including a primitive line-centering mode, some simplistic indentation, and a command that left a specific amount of vertical space for a figure (e.g., illustration) to be inserted later.

Like text coding itself, the concept of a “control code” was likely borrowed from the field of telegraphy.²⁵ Much as the “shift” key on modern keyboards signals a modal change into an uppercase character set, control codes allowed both content and control signals (commands) to be mixed together in a single stream of data. Although it seems a minor technical detail, the idea of text vs. control codes has a direct analogue with traditional editorial notation. For centuries, editors and typographers have used standardized symbols to “mark up” an author’s manuscript, indicating for the compositor what elements should be, say, italicized or boldfaced, or spaced or aligned in a certain way. Similarly, a “markup” format is interpreted by a software program to indicate specific styling.

Note that TJ-2 did not interface with a phototypesetter — or any typesetter. Its output was destined to be printed on what was essentially an automatic typewriter outfitted only with monospaced fonts. While this seems a limitation, perhaps it was a necessary constraint at the time. But the TJ-2 went on to inspire (directly or indirectly) a long branch of typesetting software beginning with RUNOFF (“A Right-Justifying Type Out Program”) in 1964, a program that used “control words scattered in the text [to] provide detailed control over the format” of text.²⁶ RUNOFF substituted TJ-2’s proprietary “overbar” control code with a simple period, and expanded the set of commands to produce line and page breaks and folios (page numbers). Then, in a fairly confusing list of technical begats over a decade or so, RUNOFF (capitals) led to “runoff” (lowercase), then to “rf,” “roff,” “nroff,” and “troff” — all software programs that utilized the same basic idea as TJ-2.

As the culminating product, the “troff” software, produced by AT&T’s Bell Labs research division, was a vast improvement over its ancestors. It incorporated an expanded command set, supported traditional proportional fonts, and interfaced with auxiliary software for typesetting bibliographies and references, tables, mathematical equations, diagrams, and more. More importantly, it interfaced directly with the Graphic Systems CAT phototypesetter, which revolutionized offset

²⁴ Peter Samson, *TJ-2: Type Justifying Program* (MIT, 1963) (transcribed at <http://www.dpbsmith.com/tj2.html>).

²⁵ Charles E. Mackenzie, *Coding Character Sets*, p. 17.

²⁶ Larry Barnes, *RUNOFF: A Program for the Preparation of Documents (CC-244/MAC-M-193)* (Washington, DC: Office of the Secretary of Defense Advanced Research Projects Agency, 1964). [<http://web.mit.edu/Saltzer/www/publications/CC-244.html>].

printing and made it easier for companies to bring typesetting and design in-house.²⁷ Later versions supported other typesetters.

Experiments and prospects

Michael Barnett, who ran the Cooperative Computing Laboratory at MIT, published *Computer Typesetting: Experiments and Prospects* in 1965,²⁸ documenting his experimentation with computerized typesetting.

Using a modest IBM mainframe, a Photon phototypesetter, and a Flexowriter keyboard, he had started his experiments with the typesetting of the Lewis Carroll poem shown here.

²⁷ Clark E. Coffee “Graphion Museum: Old Phototypesetter Tales,” [<http://haagens.com/oldtype.tpl.html#2G>].

²⁸ Michael P. Barnett, *Computer Typesetting: Experiments and Prospects* (Cambridge: The MIT Press, 1965).

[indn77d12ls24st1,,36cnxs1]
EXCERPT FROM ALICE IN WONDERLAND
[nl1sl8]
December 6, 1961
[sp4st2,10,36st3,11,36st4,12,36st5,13,36st6,14,36st8,16,36st9,17,36st10,18,36ls14d11xs2r1]
[sc19sc19]Fury said to
[xs3]a mouse, That
[nlxs6]he met
[xs7]in the
[xs8]house,
[xs9][sc19]Let us
[xs7]both go
[xs6ls12]to law:
[xs5d19]I[d11] will
[xs3]prosecute
[xs2d19]you.
[nlxs3d11] Come, I'll
[nlxs4]take no
[nlxs5]denial:
[nlxs7]We must
[nlxs8ls11]have a
[nlxs9]trial[sc47]
[nlxs10] For
[xs7]really
[xs6]this
[nl] morning
[nlxs8ls10] I've
[xs6]nothing
[xs5]to do.'
[xs4]Said the
[xs2]mouse to
[xs1] the cur,
[nlxs3][sc19]Such a
[nlxs4ls9]trial,
[xs3]dear sir,
[xs2] With no
[xs2]jury or
[xs1ls8]judge,
[nlxs2]would be

“Fury said to
a mouse, That
he met
in the
house,
‘Let us
both go
to law:
I will
prosecute
you.
Come, I’ll
take no
denial:
We must
have a
trial;
For
really
this
morning
I’ve
nothing
to do.’
Said the
mouse to
the cur,
‘Such a
trial,
dear sir,’
With no
jury or
judge,
would be
wasting
our breath.
‘I’ll be
judge,
I’ll be
jury.’
Said
cunning
old Fury:
‘I’ll try
the whole
cause.

Seeing the potential of this technological marriage, Barnett and his associates designed and built a set of software tools called PC6. One of the components was a text formatting language that could be either typed on the Flexowriter, or generated from other programs. The language was rich enough to create many types of printed documents, but simple enough that people who weren't typographers could use it. Making typesetting easier seemed to be one of Barnett's passions.

Although modest in both form and tone, Barnett’s book was highly influential, likely kickstarting the commercial development of computerized phototypesetting.

Generalized coding

By the 1970s, computer-assisted typesetting was well established. Early attempts at mixed textual/control had proved successful, and so many manufacturers of phototypesetters had adopted the general idea. However, there was a growing frustration that every system used proprietary methods for inputting, formatting, and storing text, as well as controlling phototypesetters. A document formatted for one typesetting system couldn’t easily be moved to another system. And yet technology marched onwards, with new systems appearing frequently. There seemed no way to convince the industry to standardize on a common code, and so a different approach was needed.

In 1967, William Tunnicliffe, engineer and chairman of the Printing Industries of America, proposed a way out of the quagmire: instead of the “specific markup” of proprietary typesetter codes, document producers should instead use “generic markup.”²⁹ That is, the markup codes should indicate its structure and semantics instead of its appearance. For example, instead of “Times Roman 12/15 Bold, flush left,” the markup language should say “Header 3.” The implication was that software would interpret this generic markup, and translate it into specific markup when necessary.

Around the same time as Tunnicliffe’s proposal, Stanley Rice was writing about “editorial structures” — essentially looking at documents as data structures, something that could perhaps be programmed. As he put it:

Editorial structures may be simple or complex, and flexible or inflexible. They can be simple and flexible (chapter openings), simple and inflexible (footnotes), complex and flexible (tables), and complex and inflexible (mathematical displays). Most structures have well recognized traditional forms, such as those found in *A Manual of Style* (University of Chicago) or the U.S. Government Printing Office *Style Manual*, but sometimes they are invented by authors to express some relationship of text elements that is novel or difficult...³⁰

Based on Tunnicliffe’s proposal and Rice’s concepts, Charles Goldfarb at IBM created Generalized Markup Language (GML), a standardized method of annotating a manuscript with information about its structure and semantics instead of its appearance. GML led eventually to SGML, which led to XML and HTML — the basis of our modern Internet, the coding language that lies behind everything we see and read when we visit a website.³¹

²⁹ Charles F. Goldfarb, *The SGML Handbook* (Oxford: Oxford University Press, 1991), p. 567.

³⁰ Stanley Rice, *Book Design: Systematic Aspects* (New York: R. R. Bowker, 1978).

³¹ Charles F. Goldfarb, “Design Considerations for Integrated Text Processing Systems” (Technical Report G320-2094) (IBM Cambridge Scientific Center, 1973) [<http://www.sgmlsource.com/history/G320-2094/G320-2094.htm>].

Scribe and Page Description Languages

Building upon the idea of a standardized language, also called generalized markup, Brian Reid at Carnegie-Mellon University invented the Scribe software in 1980.³² Although it used a similar text/control methodology as TJ-2 et al., Scribe's "commands" specified a non-specific style identifier instead of a proprietary command. A database of "formatting environments" (analogous to the style sheets of modern page layout applications) specified the particular font styles, sizes, spacing, and other attributes to be applied. Without changing the source document, Scribe was able to support a variety of output devices, from simple line printers to phototypesetters. Scribe was influential in many later systems, including directly inspiring the web's Cascading Style Sheets (CSS), which are used to make uniform the type and appearance of websites across each of their individual pages.

All the previously mentioned typesetting systems had an interesting if unexpected feature: in order to support a variety of output devices (screens, phototypesetters, laser printers), each system usually invented a "device-independent" format. The idea was that the "front-end" (Scribe, troff) would do the heavy lifting of command processing, line/paragraph shaping, and pagination, but save their results in a format that could then be repurposed for various output devices. However, while device-independent, these formats were proprietary, and incompatible with each other.

Researchers at Xerox PARC looked more formally at the output problem, and came up with the concept of a format that described equally all the elements of a page, including type, graphics, and images. Their page description language — first called Press, then InterPress — attempted to rationalize and integrate the usually specialized tasks of page composition.³³ Essentially, they tried to make all the page elements exist happily in the same universe, and obey the same rules.

A Cartesian coordinate system (where the X and Y axes head off in positive and negative directions from a zero origin point) defined the basic space. By defining a common geometric model, all page elements obeyed the same rules of placement and sizing. By making the geometry malleable (using affine transformations), any element could be easily rotated or distorted. By attaching color information and halftone screening to elements, a trip to the stat camera and stripping-in of the film was avoided. And by representing type as sequences of curved lines, typography could play the same geometric games.

John Warnock and Chuck Geschke worked at PARC on InterPress, but after finding Xerox unwilling to commercialize the project, started their own company, Adobe Systems. Out of the best ideas of InterPress, they built a new page description language called PostScript. PostScript was not only a way to describe the elements of printed pages, but a full programming language in itself, accessible by any software developer willing to dive into its somewhat perplexing syntax. (Early PostScript demos included a typesetting application implemented directly in the language.) In 1984, Adobe convinced Linotype to build PostScript into the Linotronic 300, a new

³² Brian K. Reid and Janet H. Walker, *Scribe Introductory User's Manual* (Unilogic, 1980).

³³ Maurice M. de Ruiter, *Advances in Computer Graphics III* (Springer Science & Business Media, 1988), p. 305.

laser-driven imagesetter. And in one of the luckiest strikes in the history of computing, Adobe collaborated with Apple to include PostScript into the new LaserWriter desktop laser printer.

Desktop publishing

By 1985 the Mac had been out for a year. It was well-received, with only a few old grumps who dismissed it as a toy. Apple was praised for their innovation — and rightly so, for they successfully copied Xerox’s Star desktop system at a price that was relatively affordable for many people, and far more open to software developers with even newer ideas. In that year, in the wake of the success of the Mac, Apple announced the LaserWriter. Laser printers already existed in the market, but with the collaboration between Apple and Adobe, it was the first time a powerful page description language like PostScript was declared to be the one and only way to interface with the printer.

Meanwhile, in Texas, Aldus was releasing PageMaker, one of the first desktop publishing applications, and the first to support PostScript. PageMaker, too, had been inspired by Xerox’s Star system, though smartly realized that its users were more interested in replacing their X-Acto knives and waxers than in new theories of office automation. Rather than expose the completely new and strange model of PostScript, Aldus instead chose to model PageMaker on the incumbent. Instead of a bright future of programmatic layouts, PageMaker emulated the dominant workflow of static text galleys, stripped-in images, and blue gridlines.

The pieces were all in place; the quadruple-play of Macintosh, PageMaker, LaserWriter, and Linotronic was powerful. Designers now could quickly make up pages visually, proof them on their desktop laser printer, and send the very same file to a service bureau who would imageset film on a Linotronic, knowing it would appear identically.

It was a serious threat to the existing industry of traditional phototypesetting, killing it off within a few years. But it was also a fulfillment of the prophecy set forward even a few decades earlier in the work of media theorist Marshall McLuhan and Quentin Fiore in works like *The Medium is the Massage: An Inventory of Effect* (1967). As Fiore wrote in a visual essay entitled “The Future of the Book”:

The changing style of life in the age of computer technology is forcing people to alter their perceptual styles to even the most commonplace things of their environment...such a world alters the most commonplace things, like the publishing of books. Publishing companies are now being acquired by the major electronic giants such as Xerox and RCA. Publishing houses who once published books are not going to publish information, but it will be instantaneous information.³⁴

Like the invention of moveable type centuries earlier, it was a perceptual revolution because desktop publishing was not a revolution in speed alone, taking precedent from the work and in-

³⁴ qtd. in Jeffrey T. Schnapp and Adam Michaels, *The Electric Information Book: McLuhan/Agel/Fiore and the Experimental Paperback* (New York: Princeton Architectural Press, 2012), p. 17.

novation of many before it. Above all, it made possible a new media experience: the experience of making your own media more easily than ever before that combined both bit and ink.

John Labovitz is a photographer, typographer, and computer programmer. He has been interested in computerized typesetting since he was a teenager in the 1980s playing with the shiny new DTP technology. He was fortunate enough to witness the transition of printing at the respective beginnings of both the desktop publishing and Internet eras, but has since escaped the clutches of the industry. John lives in West Virginia, where he runs an artist residency center called North Mountain. He is still fascinated by type, and continues to examine the nature of typesetting through software. He can be contacted at johnl@johnlabovitz.com.