The Origins of PostScript

John E. Warnock
Adobe Systems

Introduced by Adobe Systems in 1984, PostScript was a unique device-independent approach to describing the appearance of a printed page, for the first time allowing pages to be printed on a range of devices of different resolutions. From 1984 to 1987, the use of PostScript by printer manufacturers grew to the point that it became the de facto standard.

PostScript is a graphic programming language introduced by Adobe Systems in 1984. At its introduction, PostScript described pages in a device-independent manner so that a page could be printed on a wide range of devices of different resolutions. Other solutions at that time were markup or other declarative descriptions. From 1984 to 1987, the use of PostScript by printer manufacturers grew to the point that it became the de facto standard. This article describes the 11-year sequence of events leading to the announcement of the language.

MY BACKGROUND

Before telling the history of the development of PostScript, let me cover a bit of my own personal history.

I was born in 1940 in Salt Lake City, Utah, and I grew up in Utah. I received my BS in mathematics and philosophy in 1961, my MS in mathematics in 1964, and my PhD in electrical engineering in 1969, all from the University of Utah. From 1966 to 1969, I worked under David C. Evans and Ivan Sutherland on an Advanced Research Project Agency (ARPA) grant on "Man/Machine Communication."

Over the course of my career, I have worked for IBM, the University of Utah, Computime Canada, Computer Sciences Corp., Evans & Sutherland, Xerox Palo Alto Research Center (PARC), and Adobe Systems.

POSTSCRIPT MINUS 10: EVANS & SUTHERLAND AND JOY

The PostScript history really starts in 1974. In 1972–1973, I worked for Evans & Sutherland (E&S) as part of a group that was contracted by Ames Research to help make a supercomputer called the Illiac IV work for weather forecasting applications. Then, in 1974, Dave Evans asked me to hire four engineers for a new project that he wanted me to manage; I hired John Gaffney, Paul (PJ) Zima, Christine Barton, and David Lewis. We worked in a small office in Mountain View, California, to solve what appeared to be an impossible problem.

Published by the IEEE Computer Society
1058-6180/18/$33.00 ©2018 IEEE

Our employer, E&S, an early pioneer in advanced computer graphics systems that had been formed out of the University of Utah's famous graphics ARPA contract, had taken on a three-year subcontract with Philco-Ford to build a five-projector graphics system to simulate a ship steaming into New York Harbor and through its surroundings. This graphics system was to be installed on the mock-up of a ship's bridge and was to be used to train the ship's pilots how to navigate into and out of the New York Harbor. The project, called CAORF (Computer Aided Operations Research Facility), was being built for the US Maritime Academy.

Our group in the Mt. View office was responsible for modeling the harbor from the Verrazano-Narrows Bridge, down around Staten Island, into Newark Bay, and into the Port of Newark. This model needed to include the Verrazano-Narrows Bridge, Staten Island, Brooklyn, Constable Hook, Bayonne Bridge, and the Newark shoreline. This included the topography of the area and about 4,000 structures, such as buildings, oil storage tanks, and houses. We also had to construct the model to work for the expected four types of ships that might be in the harbor.

Evans & Sutherland was an early pioneer in advanced computer graphics systems that had been formed out of the University of Utah's famous graphics ARPA contract.

In addition, the Mt. View office was responsible for writing all the software needed to manage the 3D database and drive the image generators. In those days, not enough computer memory was available to store the images that were generated (frame buffers), so the image generators had to create all images from the database in real time.

The Salt Lake City office of E&S was responsible for designing the image generation hardware and the interfaces to the five Eidophor projectors. The special-purpose imaging systems were to be driven by a PDP-11/45 computer. All the electronics were to be contained in six racks (each the size of the PDP-11).

Two years had already elapsed since the start of the project, and essentially nothing had been accomplished except for designing and building a large digitizing tablet that was interfaced to the PDP-11. This tablet was designed and built by Ivan Sutherland. The bottom line was that we had to design, build, and install all the database components, software, and hardware and get the entire imaging system working in one year.

Why was this so hard?

First off, in those days a PDP-11 had a maximum of 32,000 16-bit words of memory and was incredibly slow by today's standards. It only had DEC tapes, a keyboard, a text display, and our digitizing tablet as I/O devices, and the programming was all done in assembly language (the machine's native language).

Second, other than topographic maps of the New York Harbor, no source material existed about the appearance of building, tanks, bridges, or anything else in the harbor.

Lastly, because no specifications of the electronics were available (they were not designed yet), we had no concept of the final format or requirements of the database.

So, this is what we did. John Gaffney traveled to New York with multiple cameras. The E&S project management team arranged for a small boat to take John through the Verrazano-Narrows and along the complete path of the harbor. He and a helper took pictures every few seconds from multiple cameras to record all points of view. He took about 35 rolls of 36 exposure film.

Next, all five of us in the Mt. View office spent about a week going through the images, noting the major landmarks, finding them on the topographic maps, and recording the type of landmark—for instance, three-story brownstone building, fuel tank, water tower, industrial building, or house. We would also note each landmark's color. The topographic maps showed the building's footprint, orientation, and elevation.

The question then became: How do we build a 3D database?

We imagined drawing lots of 2D menus, taping them to the digitizing tablet, taping down the topography map, and then writing a lot of programs (in assembly language) to produce some kind of database that was not yet specified.

At this point, however, John Gaffney had a brilliant idea: imagine a machine that had registers and storage, but where the operators of the machine were arbitrarily powerful. For example, operators would convert strings to numbers and numbers to strings, write strings to files, read values from the digitizer, perform mathematical functions, execute operations based on conditions, execute loops, and do anything else that higher-level languages do. But this machine would interpret the programs (text) and execute them with no compile step. Also, these programs could be associated with menus on the digitizing tablet. The user would type in the program, and the interpretive program would execute it. Arguments to subroutines would be passed through registers. This was my first exposure to the possibilities of an interpreted computer language. It was simple, relatively easy to implement, and we called it Joy.

The strategy needed to build the database became clear: rather than directly building computer structures that were the final database, we would have the Joy programs write intermediate text files that contained all the information needed to construct the data structures, no matter what they were ultimately defined to be.

For example, a water tank would need its latitude, longitude, altitude, diameter, height, and color. The digitizing program would collect this information when the user touched menu entries (color, height, width), location on the topographic map, and a menu entry specifying the type of object (water tank). It would write this information as a text line in a text file. When we discovered different types of objects, we would write a little Joy program to collect the information and output the text files.

This general methodology was used to build the entire New York Harbor database. The text files that represented the database could be edited manually with a text editor and could be processed by the final program that would generate the database in its final form. The miraculous result of using these concepts along with the 24/7 work of the Salt Lake team was that we were able to deliver the working simulator in the one-year timeframe.

The simulator was used for both research and training about 2,500 hours per year from 1975 until at least 1983 (http://www.dtic.mil/dtic/tr/fulltext/u2/a135601.pdf). Figure 1 shows two images from the CAORF simulation.

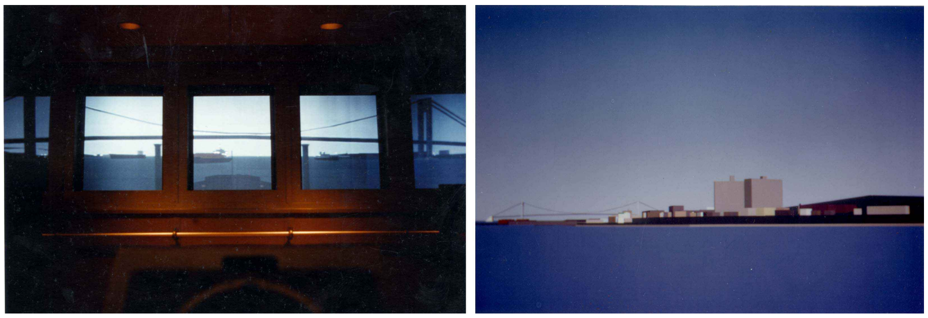
Figure 1. Two views from the CAORF (Computer Aided Operations Research Facility) simulation. Evans & Sutherland built CAORF for the US Maritime Academy to train ship pilots how to navigate into and out of the New York Harbor.

POSTSCRIPT MINUS 2: THE DESIGN SYSTEM AT E&S

The Joy implementation was sort of a toy, but it worked so well that John Gaffney, PJ Zima, and I decided to build a complete interpreted language around a more sophisticated machine architecture. The motivation for this new language was that it would make it much easier to build complex graphical databases. In particular, an upcoming project at E&S involved building a database for the space shuttle simulator for NASA. Also, E&S had other customers that had complex graphical databases and visualization problems.

We were motivated by these requirements to build graphical databases that were procedural rather than declarative. We called this effort the *Design System*.

The architecture suggested by John Gaffney was to be based on a fictitious stack machine (at that time we had no knowledge of a similar approach taken by the Forth language). The data types included integers, real numbers, strings, arrays, names, operators, procedures, dictionaries, and files. All of these are fairly common except for dictionaries. As the name implies, a dictionary is a collection of names that are associated with values, where the values can be anything. Our design system program was just a text file, where the syntax allowed for words, numbers, and strings. There were also constructors for procedures, dictionaries, and arrays.

The machine had three kinds of stacks: operand, execution, and dictionary. The execution stack contained operators and procedures. The dictionary stack provided name lookup context (scope). The operand stack provided a source command's operands. This implementation was developed on a PDP-11 between 1975 and 1978 and contained almost everything that would be eventually needed except for graphic and imaging operators. Instead of graphic operators, we interfaced to the E&S graphic displays.

This entire effort was funded by E&S and a number of consulting projects we accepted along the way.

In early 1978, Dave Evans asked me to move back to Utah to join him at the main E&S facility. My wife and I concluded that our home was in California.

POSTSCRIPT MINUS 1: XEROX PARC AND JAM

Because I had received my education at the University of Utah while working on the ARPA project, I knew a number of people who had left Utah to work at Xerox PARC in Palo Alto, California. The head of the Computer Science Lab, Robert Taylor, had worked briefly at Utah as part of the ARPA contract. Alan Kay, William Newman, Bob Flegal, and Patrick Boudelaire all came from Utah to work at PARC.

Because of these prior relationships, I left E&S in 1978 and joined Xerox PARC under Chuck Geschke in a newly established Imaging Sciences Laboratory.

Xerox PARC was and is still famous for the innovative advancements in computer science it made in the 1970s. I arrived at PARC into a truly amazing environment. Each researcher had his or her own computer, in most cases an Alto.[1] This computer was designed and built at Xerox. It had a bitmapped display the size of a printed page. Each bit on the display was either black or white. The machine had about 65 Kbytes of memory, a 2.5-Mbyte hard drive (the size of a large pizza box), a keyboard, and a mouse. More importantly, each machine was connected to a network with file servers and laser printers. Remember that this environment was around within Xerox almost nine years before the IBM PC was announced.

When I arrived at PARC, the researchers were working on two new machines: Dolphin and Dorado. The new machines were not restricted to the Alto's bitmapped display; instead, they had color displays of various resolutions. I was hired to work on device-independent graphics. Most of the graphics at PARC were focused on the manipulation of bitmaps (arrays of black and white dots). The new machines being designed had the ability to drive both grayscale and color displays of various resolutions, and the bitmap technologies used on the Alto did not scale. Graphic representations were needed that were abstract enough to be rendered on any kind of display or printer. A model with this flexibility is what we called *device independent*.

At PARC, the programming environment of choice was a strongly typed language called Mesa (not an acronym). (A strongly typed language detects type errors during compilation, not at run time.[2]) Other programming environments like SmallTalk and LISP were also used. Mesa was well suited to work on systems software as well as on any other software that demanded extreme reliability and stability.

Early on, I missed the interpretive programming environment we had at E&S because I felt it would provide a great experimental environment in which to do research. I embarked on reproducing the E&S environment at PARC, implementing this environment in Mesa. Because implementing the interpretive system was a large project, I enlisted the help of Martin Newell and Doug Wyatt. We called this language JaM (John and Martin). As part of this project, we had to define an "imaging model" and the associated graphics operations.

The graphics required at Xerox were very different from the 3D modeling that I had done at E&S. The model had to handle anything you might see on any printed page: photos (black & white or color), lines, curves, filled shapes, text (in any typeface), geometric patterns, and stencils (clipping regions) blocking portions of any of the above.

Doug and I submitted a paper to Siggraph outlining a proposed imaging model.[3] Then, Martin, Doug, and I implemented that model into JaM. The model's geometric components included straight lines and curves. To represent almost any shape one would encounter on a printed page, we chose third-order Bézier curves.

Curves and lines were combined to make graphical shapes. These shapes could be filled with images or solid colors. They could also be outlined with any color. Theoretically, the imaging model was practically complete, and the only part of the model that didn't work in a straightforward way was text. If all displays were super high resolution, then everything would have been fine, and we could have defined each letter with curves and lines and rendered them onto a raster display. But the display resolution was relatively low (72 spots per inch), and the printer resolution was about 300 spots per inch. On these devices, straightforward algorithms to fill shapes with pixels applied to small text rendered from curves and lines did not work, and the output looked horrible, as the top portion of Figure 2 illustrates.

At PARC in the 1970s, and for the Alto and laser printers, the text problem was solved by manually creating bitmaps of each character, size, and font and then storing these bitmaps in a file. This was okay, except you could not transform these characters (scale or rotate them). This was at odds with and contrary to any generalized device-independent imaging model.

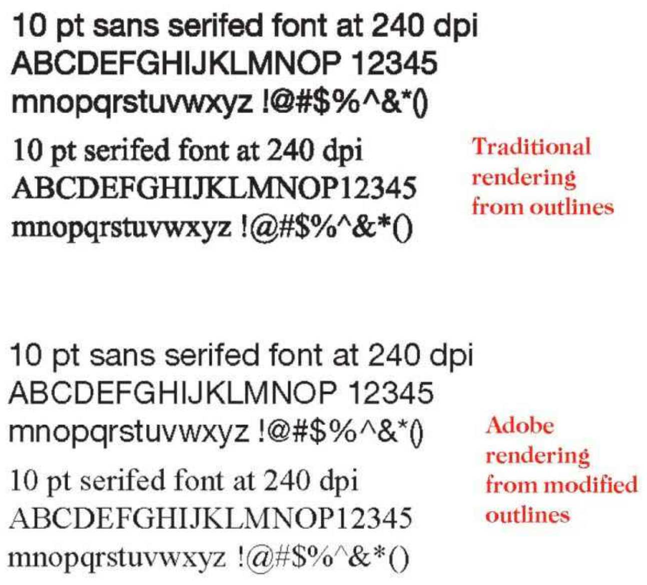
10 pt sans serifed font at 240 dpi
ABCDEFGHIJKLMNOP 12345
mnopqrstuvwxyz !@#$%^&*()
10 pt serifed font at 240 dpi
ABCDEFGHIJKLMNOP12345
mnopqrstuvwxyz !@#$%^&*()
Traditional rendering from outlines
10 pt sans serifed font at 240 dpi
ABCDEFGHIJKLMNOP 12345
mnopqrstuvwxyz !@#$%^&*()
Adobe rendering from modified outlines
10 pt serifed font at 240 dpi
ABCDEFGHIJKLMNOP12345
mnopqrstuvwxyz !@#$%^&*()

Figure 2. Two renderings of characters. In the 1970s, due to low device resolution, straightforward algorithms to fill shapes with pixels applied to small text rendered from curves and lines did not work. (Courtesy of the author)

INTERPRESS

In 1979, five senior scientists (Butler Lampson, Bob Sproull, Chuck Geschke, Brian Reid, and I) and one project manager from El Segundo (Jerry Mendelson) embarked on designing a new printer protocol for Xerox. The motivation for this effort was that Xerox was trying to commercialize many of the developments that had occurred at PARC and make them available to corporate America. The machines and systems they were building was called the Star System.

The Interpress team had extensive experience in this area.[4] Bob Sproull wrote the existing printer driver at PARC called Press. Butler Lampson was probably the most senior systems architect at PARC. Brian Reid was the author of a document creation system called Scribe. Chuck Geschke was an architect of the Mesa programming language. Jerry Mendelson had system printer experience. And I had considerable experience with graphics.

Over a two-year period, this group only met together three or four times. Most interactions took place over email. Actually, I think this was good because each member had to think carefully about suggestions and responses to ideas that were introduced. Thus, little emotion interrupted this process. Because of the team's experience base, several features of the printing protocol were thought to be essential and required for any solution to be acceptable. These requirements included page independence and extensive bitmap handling functions to provide font capabilities.

These overarching requirements made the design process difficult, and the solutions were disconnected and awkward. Interpress was becoming a mash-up of a declarative format and a JaM-like language. I was skeptical that any user would be able understand it. The biggest and fatal problem with Interpress was that it was not device independent (due to the font strategy). I felt that this problem would doom its future success. At the end of the process, it was my personal belief that Interpress could not be practically implemented,[5] but most of the rest of the team did not agree.

The initial idea behind Adobe was to leverage our experience at Xerox PARC and build a document creation system that businesses would use.

In spite of our misgivings, Chuck and I were able to sell this potential solution to the powers that be at Xerox. And Xerox adopted the Interpress solution as a standard, but only under the constraint that it would never be disclosed to the public until all Xerox printers were driven by Interpress. This constraint was a breaking point for Chuck and me. We were both convinced that this solution would never be successful.

Up until that point, my employment at PARC was the best job I had ever had. We were living in a scientist's dream sandbox, surrounded by and working with some of the most creative, talented computer scientists in the country. We were also given extraordinary freedom to create.

But Chuck and I spoke in his office one day, and we agreed that some of our best work would never see the light of day and that we should consider leaving Xerox to form a new company. I flew to Salt Lake City to meet with Dave Evans (my thesis advisor and previous employer at E&S). We discussed what Chuck and I had in mind, the building of computerized document creation systems, and he introduced us to Bill Hambrecht, a well-known and highly regarded venture capitalist with Hambrecht & Quist in Silicon Valley. He was intrigued by the idea of a company that could engage the printing and publishing sector. Because of his own business, he personally did not like working with the financial printers then available. Hambrecht & Quist committed $2 million over two years to fund our new company.

To avoid intellectual property issues, our new company licensed the design system from E&S. We needed no implementations because we would be implementing on different machines.

On 2 December 1982, Chuck and I founded Adobe Systems. After many false starts, we picked the name Adobe because it was the name of the creek down the street from our house. (Figure 3 shows one of my early photos during this time.)

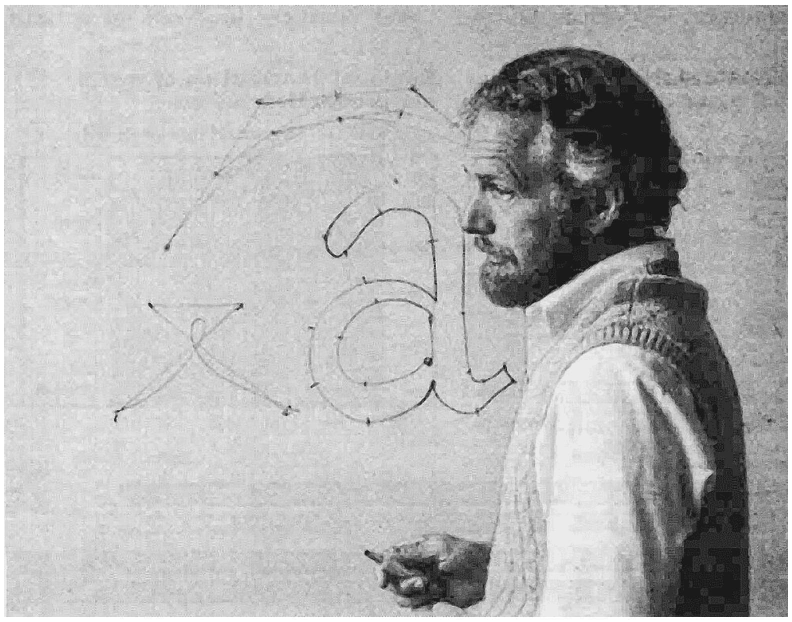
Figure 3. John Warnock. This photo taken in 1984 just before PostScript was announced. (Courtesy of the author)

The initial idea behind the company was to leverage our experience at Xerox PARC and build a document creation system that businesses would use. At PARC, where we both had worked, we had used the Alto computer and its successors (Dolphin and Dorado), Ethernet networks, and Xerox laser printers as an effective document creation system, and we were interested in building a configuration outside of Xerox that would replicate its functionality. At that time, the personal computer was not yet widely available, Apple computers were toys, and the IBM PC was just being introduced, but workstations from Sun and Apollo were becoming established in the market. We anticipated using Sun Microsystems workstations and existing laser printers and then building additional hardware and software, as necessary, to make the whole system functional. We even contemplated building a high-resolution machine to replace traditional typesetters.

At Adobe we took an entirely different approach than we had taken in the Interpress design at Xerox PARC. We decided to gamble by basing our solution on JaM, a familiar interpreted programming language that I had championed over the years. This interpreted language would evolve into PostScript with help and guidance from extremely talented software engineers (Chuck Geschke, Doug Brotz, Bill Paxton, and Ed Taft).

After writing an ambitious business plan, we talked with many vendors, potential clients, and potential partners. Then, in May 1983, Steve Jobs called us because he had heard about our efforts and asked to meet with us. At that time, Apple was completing the final aspects of the Macintosh, which was to be partnered with a dot-matrix printer. Steve had also been meeting with Canon about their new low-cost laser printer. I think Steve realized that a dot-matrix printer was not going to succeed in an office environment, so he needed a laser printer to go along with the Mac.

Steve was taken by our progress and experience in driving laser printers, and he was interested in a partnership with Adobe. One of the advantages that PostScript had for Apple was the language interface. Apple had developed applications for the Mac and had a graphic interface called QuickDraw. Because PostScript was a programming language, it was fairly straightforward to write a QuickDraw-to-PostScipt converter in PostScript. This converter would reside in the printer's memory, making the interface to the Mac fairly easy. This property of PostScript would be used by many customers and applications.

For our work, we had borrowed a laser printer from Digital Equipment Corporation and had been having discussions with Sam Fuller, who was the head of research at DEC. Sam was also interested and supportive of our work with laser printers.

Because of the increasing demand for a new way to drive printers and our discussions with potential customers, we decided to change the Adobe business plan and focus on providing software that would provide the interface between the emerging personal computers and the newly introduced laser printers.

POSTSCRIPT AND SOLVING THE FONT PROBLEM

Even though our initial focus was on full publishing systems, printing and PostScript were always a major component of our various business plans. In that regard, there was one remaining very technical hurdle to overcome. As we developed PostScript, we were committed to unifying the imaging model so that fonts and text could be manipulated like any other graphics components. That would mean that one could scale, rotate, and transform all text and graphics in a unified, device and resolution independent way. But laser printers at PARC were 300 dots per inch, the newly introduced Canon printer was 300 dots per inch, and image setters from Linotype were 1,200 dots per inch. Documents in PostScript had to image flawlessly on all these devices and had to anticipate the color displays and color printers of the future. We strongly believed that solving these problems was the key to Adobe's future success.

By June 1983, PostScript had been developed to the point where we could start experimenting with outlined fonts (letters defined by mathematical curves). In the past, the font problem was framed as a scan-conversion problem. That means, given a letter outlined by a set of curves, you found an algorithm that would "turn on" the pixels inside the curve that would produce an acceptable bitmapped character for the device's resolution. This problem had been attacked by researchers at PARC and Donald Knuth at Stanford, with little or no success.

I came up with a different way to frame the problem: instead of figuring out what bits to turn on, I suggested that we modify the outline of the characters based on the target resolution so that a standard scan-conversion process would yield great bitmaps. After working months on this problem with Doug Brotz and Bill Paxton, we finally came up with a robust solution that passed the scrutiny of graphic artists, publishing professionals, printer manufacturers, and type designers (see the bottom portion of Figure 2).

We decided not to patent the font technology because of the possibility that someone would figure out a way to work around the patent.

At Adobe, we decided not to patent this approach because of the possibility that someone would figure out a way to work around the patent. Instead, we kept the approach a trade secret until 2012, when I gave a Goldstine Lecture at the American Philosophical Society.[6]

The basic idea behind the solution was to distort the outlines of each letter so that the vertical and horizontal boundaries would align with the pixel rows and columns. We also modified the curves so that the letters were less fat. (This reduced the apparent weight of the letters.) These changes to font rendering produced great results and made PostScript truly device independent.

POSTSCRIPT ADOPTION

Solving the font problem allowed PostScript to be widely adopted throughout the computer industry.

We started to call on customers, printer manufacturers, industry gurus, and potential partners to show them what we had done. The reaction was pretty much disbelief that laser printers could produce such varied and high-quality output. One of the industry leaders, Jonathan Seybold, told us this technology would change the entire world of printing and publishing.

We signed contracts with Apple, Linotype, DEC, Wang, QMS, and Compugraphic to produce a variety of laser printers and typesetters. After we signed with IBM in 1986, Hewlett-Packard decided to license PostScript for its printers. At that point, we essentially had established the standard for the industry.

We did not know at the time that PostScript would became the foundation for Adobe's success. Together with a following derivative technology called PDF, it would play a major role in transforming all aspects of printing and publishing over the whole world.

(Both John Gaffney and PJ Zima, who are now deceased, worked with me on the CAORF project in 1974 and finished their careers working at Adobe.)

The rest of this story is the history of Adobe and the desktop publishing industry, which will be the subject of a subsequent special issue article in the *Annals* next year.

REFERENCES

1. C.P. Thacker et al., *Alto: A Personal Computer*, CSL-79-11, Xerox PARC, August 1979.
2. C.M. Geschke, J.H. Morris Jr., and E.H. Satterthwaite, "Early Experience with Mesa," *Comm. ACM*, vol. 20, no. 8, 1977, pp. 540–553.
3. J. Warnock and D.K. Wyatt, "A Device Independent Graphics Imaging Model for Use with Raster Devices," *ACM Siggraph Computer Graphics*, vol. 16, no. 3, 1982, pp. 313–319.
4. R.F. Sproull and B.K. Reid, "Introduction to Interpress," *Xerox System Integration Guide*, XSIG 038306, Xerox PARC, June 1983; http://www.bitsavers.org/pdf/xerox/xns/standards/XSIG_038306_Introduction_to_Interpress_Jun1983.pdf.
5. B. Reid, "PostScript and Interpress: A Comparison,"; https://groups.google.com/forum/#!msg/fa.laser-lovers/H3us4h8S3Kk/-vGRDirzDV0J.
6. J.E. Warnock, "Simple Ideas That Changed Printing and Publishing," *Proc. Am. Philosophical Soc., vol. 156, no. 4*, 2012, pp. 363–378.

ABOUT THE AUTHOR

John E. Warnock cofounded Adobe Systems with Charles Geschke in 1982. He was successively president, CEO, and board cochairman at Adobe from its founding until 2017. With Geschke, he developed the PostScript page description language. He has also held positions at Evans & Sutherland, Computer Sciences Corp., IBM, and the University of Utah. He has a PhD in electrical engineering from the University of Utah. Contact him at warnock@adobe.com.