NEW TRENDS

Computer Typesetting or Electronic Publishing? New trends in scientific publication

Philip Taylor

Abstract

W ciągu ostatnich 15 lat skład tekstu wspomagany komputerowo całkowicie wyeliminował stosowanie tradycyjnych technik drukarskich. Obecnie stoimy przed jeszcze bardziej radykalną rewolucją technologiczną, jaką stanowi pojawienie się wydawnictw elektronicznych (electronic publishing – w skrócie e.p.).

W przeciwieństwie do składu komputerowego i tradycyjnego, publikacje elektroniczne w ogóle nie wykorzystują papieru, nośnikem informacji staje się ekran komputera.

Potencjalne korzyści płynące z zastosowania tej nowej metody publikacji wydają się oczywiste: znaczna redukcja kosztów i niemal natychmiastowa dystrybucja do zainteresowanych odbiorców. Nie można jednak pominąć i wad, takich jak: możliwość bezprawnego kopiowania, łatwość plagiatów i nielegalnej redystrybucji.

Obie wspomniane metody publikacji różnią się w sposób zasadniczy: o ile systemy składu tekstu z góry narzucają postać końcową strony, to systemy publikacji elektronicznych stwarzają jej bardziej niezależną reprezentację, w której ostateczna forma zależy raczej od programu prezentującego dokument (tzw. viewera, czy po polsku – przeglądarki).

W niniejszej pracy omówione są najnowsze osiągnięcia zarówno w dziedzinie składu komputerowego jak i publikacji elektronicznych. Wskazuje się również na różnice tych dwóch metod rozpowszechniania.

A Brief History of TEX

Until about fifteen years ago, typesetting was virtually ignored by the vast majority of mathematicians, scientists, and scholars in general: manuscripts were prepared using a typewriter, the more esoteric symbols (which meant almost all symbols for mathematicians) were laboriously inserted by hand, and the whole was then simply dispatched to the publisher. Some time later galleys would be returned, emendations noted in the margin, and once again the whole would be sent to the publisher. A similar but shorter cycle was probably repeated for the page proofs, and finally the author's intentions appeared in final form in the finished book. At no point did the author and the typesetter communicate directly, and indeed the former was almost certainly virtually unaware of the latter's existence.

The typesetter, however, was only too aware of the author: mathematical copy is traditionally referred to as 'penalty copy' in the printing trade, since it is notoriously difficult to set correctly. In the time that his colleague could set ten pages of straight text, the mathematical typesetter was barely able to accomplish a single page, and even when set he knew that there was every possibility that it would have to be re-set more than once, since mathematicians are only too keen to invent new symbols of their own when no existing symbol seems entirely appropriate. And since the typesetter would never have encountered such a symbol before, he would (quite reasonably) assume that it was simply a badly drawn version of a symbol with which he *was* familiar, and substitute the latter...

Needless to say, some of the more aware authors began experimenting with computer technology as soon as it became generally accessible, and for a while the academic world seemed convinced that if it were possible to get just a couple more symbols onto the daisy-wheel of a Diablo printer, all would become possible: there were even specialist companies who would re-mould a daisy-wheel, replacing an apparently unwanted glyph with one which its owner deemed indispensable. Of course, the approach was doomed to failure: one can no more set mathematics with a fixed set of 144 glyphs than can one with a set of 128, and despite the best efforts of all concerned, the daisy-wheel printer was soon consigned to the scrap bin.

In parallel with this, the dot-matrix printer manufacturers first began to have a significant impact. With a 7×5 dot matrix, there are potentially

$$\sum_{i=0}^{35} \binom{35}{i} = 2^{35}$$

different characters (a very large number indeed!), but unfortunately a number of these are virtually indistinguishable: a single dot at co-ordinates $(4, 3)$ looks astonishingly like another single dot at at co-ordinates $(4, 4)$ to even the most astute reader (I believe that there are 33 034 338 305 *distinct* characters, as opposed to a total of 34 359 738 368 characters, where a character is regarded as *distinct* if it's not simply the result of sliding another character horizontally, vertically, or both: this figure is based on an analysis by Dr Warren Dicks of the Autonomous University of Barcelona). Furthermore, the print quality of a 7×5 dot matrix printer is so appallingly bad that no attempt should *ever* be made to set a book using one – unfortunately this well-meant advice was seldom heeded at the time.

Of course, in order to exploit these technological revolutions, suitable software had to be written, and the Unix world in particular decided to standardise on ROFF and its derivatives: NROFF, TROFF and finally DITROFF all made their mark. Unfortunately none of the ROFF derivatives ever directly supported the typesetting of mathematics, and so adjunct programs such as EQN and TBL had to be used to add mathematical functionality. There were also commercial systems, used to set publications such as the *Transactions of the American Mathematical Society*, but these were both expensive and arcane, using a rather non-mnemonic syntax to represent the possible mathematical constructions.

Fortunately (as is absolutely clear in retrospect), at least one eminent mathematician believed that something better not only could, but *should*, be created; and being not only a mathematician but a computer scientist, he decided to create it. His name was Knuth, and his creation was TEX.

Yet had it not been for a happy co-incidence, TEX might never have been born. At the time, Knuth was working on his *opus magnum*, a seven-book series entitled *The Art of Computer Programming*, and by 1977 the popularity of the early volumes of this series had proved so great that Volume 2 had already run to a second edition. Unfortunately the timing of this was such that whilst the first edition had been set using traditional hot-lead technology, the second edition was produced using one of the first phototypesetters [an aside to readers: throughout this paper I use the term *typesetter* to mean both the *person* performing the task of setting type, and the *equipment* used to achieve that end: I hope that it is always clear from the context which of these two meanings is to be inferred, since there is no other word which could easily and felicitously be substituted for either of these usages]. And whilst the new phototypesetter was more than capable *in theory* of achieving results as good as, if not better than, the traditional hot lead device used previously, the results in practice left a great deal to be desired. Knuth, as mathematician and computer scientist, was convinced that the fault lay not in the technology but in the software used to drive it, and he decided that rather than see his life's work appear in second-rate format, he would devote a short portion of his professional life to developing a suite of software which *would* exploit the full potential of the phototypesetter. Little did he know when he took this brave decision that it was to take not the anticipated one year but at least ten, although he most certainly had a demonstrably working version within his anticipated time-frame.

The first published reference to TEX is probably *Mathematical Typography*, published as report *STAN-CS-78-648* by the Computer Science Department of Stanford University; in the bibliography to this, Knuth gives the definitive reference as being *Tau Epsilon Chi, a system for technical text* which was at the time "in preparation" and is now sadly out of print. For those interested in the subject, the former paper makes fascinating reading, and the bibliography alone makes it a more than worthwhile acquisition; it was reproduced in the *Bulletin of the American Mathematical Society*, in which form it should still be available.

TEX was both typical and atypical of programs of its era: it was typical in that it was completely script-oriented, pre-dating as it did any widely-used graphical user interface; it was atypical in that it was a completely programmable *macro* programming language, in which there were no reserved words, and in which even individual characters could change their semantics on the fly. Thus a TEX document consisted both of the text to be typeset and the commands to accomplish that typesetting, and only TEX itself could unambiguously determine whether any particular element of the document was to be interpreted as 'program' or 'data'.

Despite being created primarily in order to accomplish one particular end – the typesetting of Volume 2 of *The Art of Computer Programming* – TEX rapidly took on a life of its own, and soon became the *de facto* standard for typesetting within much of Stanford University. Before long its fame had spread, and by 1980 the TEX Users Group had sprung into existence, with members of the Steering Committee drawn from far beyond the restricted domain of Stanford faculty. The American Mathematical Society were represented on that Committee, and liaison between the AMS and Knuth was very close: Knuth assigned the TEX logo to the AMS who then applied for trademark protection to prevent it being used to describe any unauthorised modification of TEX – unfortunately this application was rejected because of a prior registration of TEX (sic) by Honeywell, but despite this lack of formal registration, Knuth's high profile and high standing ensure that the TEX logo (or its non-typeset equivalent, TeX) is universally recognised and respected.

Within a couple of years, it became clear that the initial implementation of TEX left something to be desired, both in terms of functionality and in terms of portability, and Knuth set out to redress both by re-implementing TEX from scratch. This time he decided to eschew SAIL ('Stanford Artificial Intelligence Language') as the language of implementation, and instead to adopt the far more widely available programming language Pascal. To further increase its portability, he adopted only a strict subset of Pascal, encompassing only those features which he was confident could be found (or easily emulated) on all Pascal implementations; but he also decided to take this opportunity to render the program in a form which he termed 'literate': that is, he wanted people to be able to *read* the source of TEX in the same way that they might read a book, and to therefore be able to benefit by being exposed to a major piece of software engineering presented in a highly literate manner. Once again Knuth decided that there were no adequate tools available for this, and once again he digressed from the main project by breaking off to design and implement the concept of a WEB program, together with its two adjunct programs TANGLE and WEAVE.

A WEB program consists of a highly stylised dialect of Pascal, interspersed by lengthy comments describing the purpose and function of every element and module of the program (I suspect that Knuth would deny this, and say that a WEB program consists of a highly elaborate description of the workings of the program, interspersed by occasional fragments of Pascal which implement that functionality: and I suspect that he would almost certainly be right!). By permitting the elements of a Pascal program to be presented in arbitrary order (as opposed to the strict order of presentation required by the Pascal standard), WEB allows the programmer the opportunity to present the elements of a program in a natural and logical order, as opposed to the artificial order imposed by the Pascal design criterion of 'efficient compilability': it is then the task of TANGLE to paste together these fragments in the order required by Pascal, and the task of WEAVE to bring together both the program fragments and the comment fragments into a form which can immediately be typeset by TEX.

Thus for the first time TEX became self-referential: in order to be able to produce the Pascal code from the WEB source, one needed a working version of TANGLE; to be able to produce a literate listing of the WEB source, one needed a working copy of WEAVE; but both TANGLE and WEAVE are themselves written in WEB, so to produce a working TANGLE one needs a working TANGLE, and so *ad infinitum*. Of course 'bootstrapping' (as the technique is generally termed) is well understood in the Computer Science world, and it was estimated that the task of 'hand compiling' TANGLE from the WEB source was well within the competence of 'the average implementor': however, I remember only too clearly the trauma through which a colleague went when he attempted this bootstrapping for himself...

During the re-implementation, Knuth re-wrote almost the complete TEX program: he had learned much about its limitations during the first couple of years of use, and by 1982 a completely re-written TEX had emerged. This version of TEX (often referred to as TEX 82, to differentiate it from the earlier version which analogously became known as TEX 78) was rapidly ported to a wide range of machines, and is quite possibly the most widely available program in the world today, being available on every class of system from the smallest PC to the largest super-computer. Its almost universal acceptance as *the* standard package for computer typesetting is almost certainly the result of a large set of very positive attributes: the source of the program, and the vast majority of implementations, are available either free of charge or at a modest cost which covers no more than the media on which they are supplied; the program is virtually bug-free, a claim which Knuth backed up until very recently by offering a cheque for every bug found, the value of the cheque doubling each year since the scheme's inception (he still offers a cheque, but the value no longer doubles, since he estimated that before too long it might exceed the total Federal reserves...); the program is highly stable (there were virtually no major changes during the period 1982–90, and similarly there have been virtually no changes at all since 1990, nor will there be at any point in the future); and there are an enormous number of users throughout the world, most of whom are only too keen to pass on their expertise to any who need it, so any real problems resulting from a lack of experience with TEX can be rapidly resolved by a message to any one of a number of TEX-related mailing lists and news groups (even those without network access are not cut off, as the TUG (TEX Users Group) office offers telephone support from 03:00 in the morning until late in the evening – a service which is *not* restricted to members of TUG).

So, during the 1980's, TEX emerged as *the* standard package for computer typesetting: it was available on almost every conceivable system, device drivers were written for everything from dot-matrix printers to 2400 dpi phototypesetters (but *not* daisy wheel printers!), and an ever-increasing number of publications appeared which were either typeset using TEX, or were about TEX, or both. Many scientific journals adopted it (or one of its derivatives such as LATEX, which may be thought of as a somewhat restrictive but more user-friendly 'front end' to TEX) as the standard format in which papers were to be prepared. Since an author could very easily proof a paper using a local implementation of TEX, and since TEX was *guaranteed* to produce identical results no matter on which system it was run, the number of iterations between author and publisher was reduced to the bare minimum, and all benefitted. And since TEX has been designed by a mathematician, and since a part of its *raison d'être* had been to allow mathematics to be typeset almost as easily as running text, its take-up by the mathematical community was if anything even faster than its take-up by the scientific and academic communities in general.

To give a simple example of why TEX is ideally suited to the typesetting of mathematics, consider the following set of equations:

$$\begin{aligned}\biggl(\int_{-\infty}^\infty e^{-x^2}\,dx\biggr)^2 &=\int_{-\infty}^\infty\int_{-\infty}^\infty e^{-(x^2+y^2)}\,dx\,dy \\ &=\int_0^{2\pi}\int_0^\infty e^{-r^2}r\,dr\,d\theta \\ &=\int_0^{2\pi}\biggl(-\frac{e^{-r^2}}{2}\bigg|_{r=0}^{r=\infty}\biggr)\,d\theta \\ &=\pi. \qquad (11)\end{aligned}$$

A mathematician writing this by hand would almost certainly start with the left-most element of the first line, proceed from left to right, and alternate between baseline, subscript and superscript elements as logic dictated; a pure WYSIWYG ('What you see is what you get') word processor, on the other hand, would require the typist to analyse each row of the equations into horizontal strata (thus the top stratum might contain only ∞, 2, ∞ and ∞, for example) and to enter these stratum by stratum; since, in general, WYSIWYG systems do not automatically displace preceding or following lines of text horizontally when an intervening line is shortened or lengthened, the correction of such equations is tedious and error-prone in the extreme. More recent, WYSIWYG-like, systems require a different approach in which the author has to enter the formula in the order dictated by its parse-tree; needless to say, this approach too demands more of the author than should reasonably be expected.

TEX allows the mathematician to enter the formulæ in the most natural manner, starting at the left and finishing at the right; alignment is automatically maintained if insertions or deletions are made, and even the horizontal alignment of the four primary = signs is performed automatically, virtually regardless of the length of individual left- or right elements. To clarify this, here is the exact TEX source which was used to set the table:

```
$$
\eqalignno
  {\biggl(
      \int_{-\infty}^\infty e^{-x^2}\,dx
   \biggr)^2
  &=\int_{-\infty}^\infty
    \int_{-\infty}^\infty
         e^{-(x^2+y^2)}\,dx\,dy \cr
  &=\int_0^{2\pi}\int_0^\infty
         e^{-r^2}r\,dr\,d\theta \cr
```

```
  &=\int_0^{2\pi}
    \biggl(
      -{e^{-r^2}\over 2}
          \bigg \vert_{r=0}^{r=\infty}\,
    \biggr)
    \,d\theta \cr
  &=\pi.&(11)\cr
}
$$
```

It is worth noting that TEX completely ignores any spaces in mathematical text, since the rules for typesetting mathematics are complex, and cannot be expected to be understood by mere mathematicians! Thus the layout of the equations above is simply for the convenience of the author, and is completely ignored by TEX, which is far more concerned by special characters such as dollars, backslashes, braces, underscores, carets and ampersands. And whilst each of these characters has a distinct meaning to TEX (a dollar symbol, for example, both introduces and terminates a stretch of mathematical text), that meaning may at any time be overridden, and either assigned to a different character or, if not needed, turned off completely. So, for example, if some particular computer lacked a backslash key, it would be trivial to assign the semantics of backslash to some other key (say, yen, if a Japanese keyboard were to be used).

Furthermore, it can be seen that TEX is highly mnemonic in its choice of control sequences ('commands', preceded by a backslash); to pick out just a few examples, `\int` represents an integral sign, `\infty` an infinity, `\exp` the exp operator (representing the exponential e) and so on. Compound subscripts and superscripts are presented in logical order, rather than in order of their appearance vertically on the page; and facilities are provided for the author to give TEX hints about the logical structure of the expression, so that (for example), `\,` is used to set off differentials such as $d\theta$ from the preceding text by a little extra white space, thereby improving both the appearance and the legibility of the expression.

Thus the attraction of TEX for mathematicians is clear: a highly logical markup language, capable of being entered from any keyboard; access to a very wide range of mathematical symbols; professional standards of layout; widespread acceptability by journals; and the ability to proof on anything from a dot-matrix printer to a 600 dpi laser printer. Add to this the now universal ability to preview the document on the computer screen (something the early advocates of TEX could only dream of), and it is hard to explain why any mathematician with access to a computer would *not* typeset his papers using TEX!

However, use of TEX is restricted neither to mathematicians nor to North Americans, and at the TEX User Group conference in 1989, an influential and voluble group of European TEX users ganged up on Knuth and succeeded in convincing him that, despite his assertion on the previous day of the conference that the development of TEX was finished, there were features missing from the current implementation which made TEX entirely useless to the majority of the world, since whilst it behaved perfectly in unaccented languages, it was grossly deficient for typesetting any language which made more than occasional use of diacritics. And Knuth, recognising the validity of this argument, agreed that something had to be done.

The result of all this was TEX 3: TEX 82 became known simply as TEX 2, and TEX 3 became the One True TEX. In practice, this just didn't happen: those who had no need for the extended diacritic support offered by TEX 3 simply continued to use TEX 2, and for quite a while TEX macro writers had to write very defensive code which first checked the environment before making any assumptions about (for example) the number of distinct characters with which TEX could internally deal (this was 128 prior to TEX 3, and 256 thereafter). With the release of TEX 3, Knuth made it *absolutely* clear that this really did represent the end of the TEX evolutionary line: he had better things to do with his time, and TEX was now frozen (modulo any essential bug fixes, which he undertook to continue to make if and only if it could be shewn that their fixing was essential). Furthermore he made it equally plain that TEX could *not* be further evolved by anyone else: he wished to leave for his children, and for his children's children, and for all perpetuity, TEX as his creation, and not as his-creation-as-modified-by-someone-else.

In general, the TEX world took this in good part: Knuth is enormously highly respected by those who use TEX, and there were very few who advocated ignoring his wishes and who were prepared to suggest modifying TEX. But there were also a quite significant number of TeX users, among them the present author, who felt that if TeX did *not* evolve, then it would simply die. Not because of any fundamental deficiencies in TeX – it is generally accepted that there are very few – but because the world had moved on since 1978, and whilst a script-driven language might have been state-of-the-art then, it most certainly was not state-of-the-art now. Furthermore, despite increasing the number of distinct internal characters from 128 to 256, Knuth had done little if anything to enhance TeX to deal with Asian languages, in which the number of distinct characters may be measured in thousands if not in tens of thousands. And finally, there were those who felt that there were *some* areas in which a very significant increase in functionality could be gained (particularly from the perspective of the macro programmer, who is also known as a 'format writer' when the suite of macros provides a complete functional system in its own right) with relatively little investment in terms of modifying TeX.

The implementation of these ideas probably represents the leading edge of TeX technology today: companies such as Blue Sky have produced instantaneous/incremental TeX interpreters, which are capable of displaying the effects of a change to the source code of a TeX document in real time; Advent Publishing have produced 3B2, which allows both a graphical and a textual specification of a layout, automatically updating one to reflect changes in the other; John Plaice and Yannis Haralambous have implemented a 64-bit version of TeX which uses Unicode internally; and the group with which I am most closely associated (the NTS group, where NTS stands for 'New Typesetting System') have produced a completely compatible successor to TeX, called e-TeX, which adds functionality without compromising compatibility (the NTS group also wish to re-implement TeX from scratch, using a modern rapid-prototyping language such as Prolog or CLOS, the idea being to allow rapid experimentation with alternative typesetting algorithms or paradigms). Whether or not any of these ideas will catch on remains to be seen, although among Apple Macintosh *aficionados* Classic Textures (the Blue Sky product referred to above) is already highly thought of. One fundamental question is that of stability: since one of the great strengths of TeX is its stability, how will the world feel about systems which encompass TeX but which are specifically intended to remain evolutionary and responsive, rather than fossilised and unyielding? Only time will tell.

What is perhaps worth noting is that all of these projects have ensured that Knuth's wishes are honoured not only in the letter but in the spirit: none seeks to call itself TeX (indeed, that of John Plaice and Yannis Haralambous is called Omega, which could never be confused with TeX), yet all acknowledge the debt which they owe to Knuth and to TeX: without them, none of these other projects would ever have seen the light of day.

Parallel Developments

Of course, while TeX was evolving, the rest of the world did not stand still: computer science continued to develop, and computer networking moved from the laboratory to the military and the Universities and ultimately to the whole world. Line-oriented editors fell by the wayside, and were replaced by full-screen editors (except in the rather time-warped world of MS/DOS, which continued to offer only EDLIN until comparatively recently). Script-oriented markup languages such as the ROFF family referred to earlier were challenged by increasingly sophisticated word-processors, and WYSIWYG ('What You See is What You Get'), GUI ('Graphical User Interface'), and WIMP ('Windows, Icons, Menus and Pull-down lists') became the order of the day.

At about the same time that Knuth was starting work on TeX, John Warnock and Martin Newell re-implemented an earlier language ('the Design System') as 'JaM' ('John and Martin'!) whilst working at Xerox PARC, and from this cloistered beginning ultimately emerged both the Interpress (Xerox printing protocol) and PostScript languages. Whilst Interpress remained relatively unfamiliar, Adobe PostScript took the computing world by storm: for the first time there was a *de facto* page description language, which allowed complex pages to be described algorithmically (and thus very efficiently). Although Hewlett Packard's Printer Control Language (PCL) continued (and continues) to be both widely supported and widely emulated, PostScript rapidly established itself as *the* standard for high-level printers (by which I mean laser printers and better), and fairly quickly printer manufacturers sought to provide either PostScript interpreters or PostScript emulators for their high-end products. Unfortunately (for the emulator writers) PostScript is a complex language, and many of the earlier emulations were deficient in one or more respects; Adobe, of course, as designers of the language had far fewer problems in this respect, although even they released improved versions of their interpreter as time went by.

For a long while parts of PostScript remained a closely guarded secret: the mysterious `eexec` operator was undocumented, and whilst the PostScript manual gave information on the format of so-called 'Type 3' fonts, the equally mysterious 'Type 1' fonts remained undocumented. Of course, reverse engineering is a well-understood tool, and finally the barriers were broken: descriptions of `eexec` started to appear in the press, and ultimately Adobe themselves relented and gave full documentation of both `eexec` and their Type 1 fonts.

Before long, Type 1 fonts established themselves as as much a standard for fonts as PostScript was a standard for page-description languages; companies such as Corel started to release Type 1 fonts of their own, closely modelled on industry-standard fonts but sufficiently different (at least in name) to avoid accusations of font piracy (although this latter problem continues to worry top font designers such as Hermann Zapf to this day). All the major font foundries started to offer their fonts in Type 1 format, and many gave a commitment to have *all* of their fonts in Type 1 format within the foreseeable future. The so-called 'font magic' which enabled early Adobe fonts to render well even on relatively low resolution devices such as 300 dpi laser printers was renamed 'font hinting', and this too was eventually documented by Adobe. New features continued to be added to the PostScript language, and in 1990, Adobe announced a completely new version of the PostScript language, 'PostScript Level 2'. This new version unified all previous additions to the language, and added many new features as well, such as the ability to have compact (binary) representations of a PostScript document as well as the earlier (ASCII) representation; new colour models were introduced, and support was added for composite fonts.

PostScript was originally conceived as an embedded language for printers, but before long it became clear that a version of PostScript which could drive a computer screen would be extremely useful. Adobe created their own version of this called 'Display PostScript', but in the meantime L. Peter Deutsch had started work on a PostScript interpreter of his own, called 'Ghostscript', and fundamental to its functionality was the ability to drive the screen of any computer on which it was used (it also contained drivers for a wide-range of non-PostScript printers, as well as pseudo-drivers for some of the more popular graphics interchange formats). During 1995 Peter finally announced Ghostscript version 3, which provided almost a complete Level 2 emulation, and whilst the official Adobe interpreter remained a licensed (and relatively expensive) product, Ghostscript was and remains free of charge to those who do not use it for profit-making purposes; a very significant debt of gratitude is owed by the computer world to L. Peter Deutsch, both to his skill in writing Ghostscript and to his generosity in making it so freely available, and also to the many individuals who have donated their own drivers and/or enhancements to the Ghostscript project (PS-View, from Bogusław Jackowski and Piotr Pianowski warrants special mention).

From the ARPAnet to the Web

A few years before Knuth started work on TeX, the American military as personified by [D]ARPA (the [Defence] Advanced Research Projects Agency), had initiated a pilot project to link computers over very wide distances; whilst local computer links were not uncommon, links across thousands of miles were unheard of, but [D]ARPA realised the potential military importance of such links and therefore initiated a whole series of research projects aimed at making this a reality. Whilst these projects initially started in isolation, as soon as the pilot network was available the project gained a momentum – indeed, a very existence – of its own, and the whole development strategy henceforth was established by discussion *across*, as well as *about*, the network. This network, known as the ARPAnet for obvious reasons, evolved a mechanism for distributed discussion and voting known as the 'Request for Comments' ('RFC'), and any new idea for anything from a protocol to a picnic was likely to find itself the subject of an RFC. From these RFCs emerged some of the most important *de* *facto* standards on which we still to this day: TCP ('Transmission Control Protocol'), IP ('Internet Protocol'), SMTP ('Simple Mail Transfer Protocol') and so on were all enshrined in the published versions of the RFCs, and each was allocated a unique number: electronic mail, for example, was addressed by and specified in, RFC 822.

Although the American military launched the networking initiative, it was the American universities which were actually the primary contributors to its success, and once the network was well established it ceased to be 'the ARPAnet' and became instead 'the Internet', the name by which it is still known today. Strictly speaking, the Internet is not a network *per se*, but a network-of-networks; however, the distinction is of little significance, and most now regard the Internet simply as *the* international computer network. From its military origins, where permission-to-connect almost required a personal interview with a five-star general, the Internet has now become the network to which even the most humble private citizen may aspire to gain access: Internet service providers have sprung up across much of the Western world, and connecting to the Internet today requires little more formality than a letter (and a fairly modest cheque!) to an Internet service provider, together with the purchase of a equally modest personal computer and a modem: at the time of writing, there are Internet connections from something like 150 countries throughout the world (the number of actual Internet nodes is *far* harder to gauge, but it is already estimated to lie between five and ten *million*).

Initially the protocols used, and services provided, on the Internet were very primitive: FTP ('File Transfer Protocol'), TELNET (remote terminal access), PING (check if a remote node is alive), and SMTP were probably the most common, with FINGER (check if a remote user is logged in) coming not far behind. But whilst the end-user protocols were fairly simple, the underlying mechanisms were not, and the DNS ('Domain Name Service') provided a quite sophisticated mechanism for a distributed node lookup protocol. As more experience was gained, the range of protocols and services increased, and things such as Usenet News (a distributed bulletin board) and NFS ('Network File System', providing remote access to a complete file system) were added. Then the information explosion really took off, and tools for information retrieval and display began to proliferate: GOPHER and WAIS ('Go for', and 'Wide Area Information Service, respectively) were early candidates, shortly followed by WWW (the 'World Wide Web', now usually shortened to 'the Web'). It should be emphasised that there is *no* connection between WEB programs and the World Wide Web; within this document at least, the former is consistently shewn in upper case, whilst the latter is consistently shewn in mixed case.

With the advent of the Web came one major breakthrough: whereas previously each protocol had specified its own unique method of identifying a remote resource, WWW brought with it the concept of the URL (the 'Universal Resource Locator'), so that from within a single program (the 'browser'), almost *any* Internet resource could be specified. For example, a remote FTP resource would commence `ftp://`, a remote GOPHER resource would be `gopher://`, and the Web's native resource, HTTP ('HyperText Transfer Protocol') would commence `http://` (aware readers may appreciate that this is a slight over-simplification, but the deviations from reality are essentially very small).

With the Web and URLs came unified browsers: tools such as MOSAIC which allowed access to a wide range of Internet resources from a single graphical front end. Even if a resource had no unique URL, it was still possible to associate with it an adjunct renderer which would display it correctly: thus, for example, although there is no unique URL for an MPEG file ('Motion Picture Expert Group': a compact standard for encoding and storing full-motion video), a correctly configured browser such as MOSAIC could identify an MPEG resource from its file type (the portion of the file name which follows the period), and on down-loading such a resource would then spawn off an instantiation of the appropriate renderer, so that down-loading and viewing were essentially indivisible entities.

Native-mode documents for access over the Web are coded in a language called HTML ('HyperText Markup Language'): this is a direct derivative of an earlier (but still current) specification for a generalised markup language called SGML ('Standard Generalised Markup Language'), and a conformant HTML document is also normally a conformant SGML document, although as is often the case the converse does not necessarily obtain. Both HTML and (typical but not all) SGML documents are characterised by the frequent occurrence of tags which are enclosed in angle-brackets: they therefore resemble the 'metalinguistic variables' of a much earlier standard – the BNF (or Backus Naur/Normal Form) of the original Algol-60 report – although they do not perform the same function. In an HTML document, each tag specifies the *nature* of the entity to which it refers: whilst this can be augmented by a specification of how the entity should appear, in the purest form only the nature of the entity is specified, and it is left to the browser to determine how the entity should appear. This represents a very significant philosophical breakthrough: no longer need a document be formatted by its author, the reader then requiring the technology to resolve that format; instead, using HTML, a document is simply tagged using high-level content-oriented markup, and the reader may then display that document using whatever technology is available. For example, most Internet systems are capable of running a browser called LYNX: this is a purely textual browser, and so it makes no attempt to represent subtleties of the document; it simply takes advantage of whatever text-mode functionality is available to it (for example, emboldening or underlining) to display the document to the best of its ability. Images which would normally require a graphics mode browser to resolve are simply displayed as the word IMAGE. On more sophisticated systems, graphics mode browsers such as MOSAIC (previously referred to), or the now ubiquitous NETSCAPE, can be used: these will exploit the graphics capability of their platforms to the full, and are capable of displaying full-colour, and even motion-picture-insertions, either using inherent functionality or through the medium of adjunct software which has been used to customise the browser.

But an HTML document is far more than just a passive entity: elements of it can be designated as 'hot spots', and if a hot spot is selected (using the mouse on a graphical system, or the tab and/or cursor keys on a line-mode system), a further document may be downloaded and displayed entirely automatically: the document containing the hot spot and the document referred to by the hot spot do not need to originate from, or be stored at, the same site: a document stored at (say) the University of Western Ontario can reference, through a hot spot, another document stored at (say) the University of Queensland. Furthermore, although the discussion so far has been concerned with 'documents', hot spots can in fact be linked to *any* Internet resource, provided only that the resource is specifiable via a URL. Thus a document which was fetched using HTTP can reference another document that can be fetched using only GOPHER; that document could specify a third document which is accessible only via FTP: that could contain a reference to a Usenet Newsgroup; and so on.

Yet even this does not represent the limits of a Web document: such documents can also be *forms*, with fields which must be completed by the reader; when the form is completed, a further hot spot can transfer it to a remote site, where it will be interpreted and acted upon. In this way, the original ARCHIE protocol (ARCHIE is an Internet tool for locating files available via anonymous FTP) has been extended from its traditional usage in which it is launched from a command line invocation specifying the file of interest and some constraints on the manner of search; with the HTML version (a.k.a. 'Archiplex'), the Archie user invokes his preferred Web browser to fetch an Archie form from a convenient server; he then completes the form, and uses a hot spot to return it to the Archie server; the latter then locates the file of interest, and returns a list of places at which it can be found, where the list of places is possibly constrained by options selected on the form by the user (for example, he may say that he's only interested in copies of the file that can be found within his own domain). The list of hits is then displayed by the browser, and once again using the mouse, tab key and/or cursor keys, the user selects one instance of the file of interest; the file has associated with it a hot spot, so the instant he selects the file from the list, a request is issued to retrieve the file; assuming that there are no hiccups, the file is fetched entirely automatically and displayed on the originating screen. If the file is not displayable for some reason (perhaps it is an executable image, or something else for which the concept of 'display' is ill-defined), the browser will inform the user and ask if he wishes to save it to a local disk.

Whilst previous introductions such as GOPHER and WAIS had a relatively modest impact on overall use of the Internet, which in general continued to be used mainly by academics and hackers, the introduction of HTML and the concept of the Web brought about a revolution in Internet usage: commercial companies clamoured to get on-line, governments put up their own Web pages, and every man and his dog suddenly appeared to be beset by the need to create a unique and highly individualistic 'home page' (Web documents are often regarded as being divided into pages, by analogy with a paper document, and a 'home page' is a (usually brief) document giving information about the individual who owns it; many institutions provide facilities whereby each user can create his or her own home page without formal approval). The reason for this sudden change in usage patterns is not hard to explain: whereas the more traditional Internet tools such as FTP required a modicum of expertise before they could be successfully used, the various Web browsers were intentionally designed to be 'user friendly' from the very outset, and this user friendliness together with the ability to seamlessly down-load and display documents in an astonishing variety of formats without any expertise whatsoever resulted in an unprecedented rate of take-up and an almost universal acceptance. There can be little doubt that the current near-exponential rise in Internet registrations and usage results almost entirely from the concept and ease of use of the Web.

The Web and Publishing: Unlikely Bedfellows?

Whilst it might initially seem that the two themes of this paper represent quite distinct branches of the evolutionary tree, it did not take long for those involved in publishing to realise the untapped potential of the Web: even prior to the establishment of the Web there had been some experimental use of the Internet for electronic publishing. In particular, the so-called e-journal *EJournal* (subtitled Electronic Journal for Humanists) was a direct electronic analogue of a more traditional journal, containing scholarly essays as well as shorter "letters to the editor". However, *EJournal* uses simply ASCII text as the communications medium, whilst the Web potentially allows even greater richness of medium than any traditionally produced journal, since unlike a paper journal a Web journal could contain not only text and static graphics but full motion video and sound as well.

In July 1994 the American Mathematical Society launched a project entitled "New Media", chaired by Frank Quinn of Virginia Tech., to investigate the possibility of developing a multimedium, interactive, hypertextual version of TeX: the brief of the sub-committee established to investigate this was to "co[-]ordinate the development of a technical authoring tool which will integrate text, graphics video, non[-]linear documents, hypertext links, and interactive computation. [The] tool should share the characteristics of the TeX typesetting system which have made it so remarkably useful: open file structures, open and portable source code, a stable standard core, and an uncompromising commitment to the highest quality. [It] is expected to be an extension of TeX.".

The rationale behind this proposal is also interesting: "Educational communities need interactive texts. Technical communities need hypertext and non[-]linear document types to tie together complex or cumulative efforts. Users of computation need better ways to document and illustrate programs. All these capabilities are available now in primitive forms, and authors are pushing ahead. Some are writing interactive texts using computer mathematics programs. Others are experimenting with hypertext extensions of TeX, [WWW] documents, etc. Commercial publishers are experimenting with hypertext, CD ROM publication, and linked databases. In a few years we can expect powerful tools for constructing interactive multimedia documents. But they may be 'accessible' in the same sense that typesetting was accessible before TeX: publishers will use expensive proprietary systems with closed file formats, and authors will use a multitude of free or inexpensive systems which require professional resetting to get professional results. Our experience with TeX shows that this fragmentation is undesirable and unnecessary. The HyperMath Project is being organized to avoid it.".

The introductory document for the project then went on to explain: "The HyperMath Project is primarily a framework to co[-]ordinate work already in progress. Several groups have already incorporated simple hypertext links into versions of TeX. The NTS (New Typesetting System) group is exploring improvements to traditional paper-and-ink typesetting. Most implementations of TeX have methods for incorporating graphic material, and there are publicly available packages which do this. The 'Interactive Mathematics Text' project and many groups in the calculus reform movement are using Mathematica, Maple, MatLab, and other programs to write interactive texts. These and similar initiatives can be brought together in the development of a general tool. But the opportunity is limited. As development proceeds, the costs of switching to a common standard increase, and the benefits become less obvious. We should not let this opportunity pass. The Project will sponsor working groups and conferences. The working groups will develop standards and goals, and work on prototypes. Communications among working groups will be maintained to ensure coherence and uniformity. And contacts will be developed between developers and end-users to ensure that real needs are being addressed. Working groups are planned in the following areas: traditional text; non[-]linear documents (including hypertext); inclusions (graphics, video, and sound); interactivity; and users. The first HyperMath conference is planned for the San Francisco area in conjunction with the combined math society meeting early in January, 1995, contingent on funding. The 'New Media' subcommittee of the Publications Policy Committee of the American Mathematical Society will serve as the advisory board for the HyperMath Project."

This was heady stuff: sadly by September of the same year it had been abandoned as being "too ambitious", and replaced by a more incremental approach, now entitled "non-traditional forms of publication". Whereas the earlier project had been predicated on the development and adoption of enhanced TEX, the new project proposed that "the AMS should adopt the Adobe portable document format 2.0 as the standard (output) format for electronic publication of documents". It then went on to explain that "This does not mean giving up TEX, nor does it solve all TEX problems. It is a proposed replacement for DVI as output, not the use of TEX source in authoring." What did this mean?

The first thing to realise is that by now all three threads of this paper have finally come together: TEX, Adobe, and the Web. Whilst Adobe had been very successful in developing POSTSCRIPT as a page-description language, and marketing embedded POSTSCRIPT interpreters for incorporation in laser printers and the like, it had been somewhat less successful in ensuring that Display POSTSCRIPT became established as another *de facto* standard. Indeed, with the advent of Ghostscript, a significant future for Display POSTSCRIPT was by no means certain, and the proliferation of Web-based browsers (MOSAIC, NETSCAPE and the like) which could slave Ghostscript was a further challenge to Adobe's position in the marketplace. Unlike Adobe's POSTSCRIPT interpreters and Display POSTSCRIPT systems, Web browsers were (and remain) freely available: that is, they are literally available *free of charge*, even when they are as sophisticated as NETSCAPE (which is developed and supplied by a commercial organisation). Whilst Adobe could maintain its niche as a supplier of POSTSCRIPT interpreters, it was becoming clear this was a limited, and possibly even diminishing, market: if Web-based publication rather than paper-based publication ever became the norm, the rôle of POSTSCRIPT printers and image setters might be seriously challenged as more and more documents were read from a computer screen rather than from paper. It was therefore no great surprise when Adobe finally announced (there had been clues previously, such as their work on so-called 'multiple master fonts' and Carousel) their alternative to a Web browser as a universal document rendering engine: Adobe Acrobat. Just as with the Web browsers, Adobe Acrobat is available free of charge (indeed, they send complete CD ROMs containing a full multi-lingual installation kit at the slightest provocation). And, rather like NETSCAPE, who seek to recover the costs of developing their browser by selling their Web server, Adobe will endeavour to recover the cost of the development and production of their Acrobat reader by selling the technology which is required to produce an Acrobat document in the first place.

And what is an Acrobat document? The very same thing that the AMS are investigating as a possible standard for their mathematical publications: something written in Adobe Portable Document Format (PDF). And although in theory one can develop applications of one's own which will write PDF, in practice many will elect simply to purchase Adobe Acrobat (which acts as a pseudo-printer driver for MicroSoft Windows, Apple Macintosh or Unix systems), or Adobe Acrobat Pro[fessional] (which also includes Adobe's "Distiller" to convert POSTSCRIPT documents into PDF documents), or Adobe Acrobat Capture (which uses the TWAIN protocol for scanners to generate PDF documents directly from a scanner). Thus despite their apparent generosity in giving away Acrobat free of charge, Adobe are (of course) really seeking to increase their market share by encouraging the purchase of other Adobe products.

HTML or PDF?

With HTML and PDF emerging as *the* two portable hypertext exchange standards, organisations (and, to a lesser extent, individuals) are going to be forced to make a choice. It may well be that for some applications the choice will be clear-cut, but for others there may seem little to choose between the two. It is therefore worth exploring the basic differences between HTML and PDF, in order to better allow an informed choice to be made.

HTML, being SGML, is essentially a very high level, content-oriented, markup language: its *forte* is the specification of the content of a document, and its weakness is the relatively little control that an HTML author has over the final appearance of the document. Because it is so high level, it is not possible using the current received wisdom of computer science to automatically generate HTML from an arbitrary document: if a word-processor, for example, is used to prepare a document, and if that document has been created *ex nihilo* without consideration for its logical structure, so that only the final appearance of the document has been considered, then it is almost certainly impossible to reverse-engineer the document to ascertain its logical structure: in these circumstances HTML would have little option but to represent it as an indivisible bit-map, thereby effectively wasting almost all of HTML's functionality. Despite this restriction, HTML has much to offer, for two main reasons: (1) the tools needed to generate it are already in the public domain, although the interface between those tools and pre-existing software such as word-processors is unlikely to be available (it is far more likely that word-processor packages will start to be shipped with HTML drivers, but their use may require a major re-think by the user concerning the the way in which a document is created); and (2) high-level markup is increasingly recognised as being *the* way in which to mark up a document: as experience of the use of typesetting systems such as TEX/LATEX is gained, it becomes ever more clear that low-level, form-oriented, markup is simply a dead-end and should rapidly be expunged from the practices of responsible authors.

PDF, on the other hand, consists essentially of a strict subset of POSTSCRIPT with the added functionality of hypertext: PDF documents can reference other PDF documents using hot spots, rather like HTML. According to the PDF blurb (this paper is written before my copy of Adobe Acrobat Pro has arrived, so what follows must be taken as speculative at the moment):

- Create electronic documents as easily as printing from existing applications with PDF Writer.
- Protect files with passwords; control access, printing, changing the document, adding and changing notes, copying text and graphics.
- Find exactly what is needed across multiple PDF files by searching on keywords, author, title, subject synonyms, etc.
- Re-use information easily by extracting, copying, reordering and replacing pages among PDF files – with bookmarks, links and notes preserved.
- Create custom views into information.
- Add value, set priorities and maintain a dynamic information network with links, bookmarks, notes and connections to external applications and documents.
- Take advantage of third-party plug-ins to add new features to Acrobat.
- Integrate Acrobat with desktop applications with Acrobat's support for OLE automation, Notes F/X, AppleEvents, and more.

Although perhaps it is too soon to compare HTML and PDF with any real accuracy, it would seem that at the moment they are intended for, and best suited for, rather different applications: HTML documents can either be created *ex nihilo* (for those who have no better way, simply cloning and modifying an existing HTML document is an excellent way to get started), or by using an HTML editor (of which there are already several in the public domain), or by using a package or packages (for example, a suitable word-processor) for which an HTML driver already exists. PDF documents may be created using one of the Adobe tools – Acrobat Writer, Distiller or Capture – depending on whether or not the source documents pre-exist. As HTML allows only a degree of control in the formatting and placement of entities, it is not really suitable for the presentation of anything other than simple mathematics, although HTML 3 demonstrates that the designers of HTML are aware of many of the limitations of the previous version, and are working towards a specification which may ultimately allow arbitrarily complex formulæ to be displayed. [A comment in the HTML 3 discussion document reads "Including support for equations and formulæ in HTML 3 adds relatively little complexity to a browser. The proposed format is strongly influenced by TEX."]. Of course, since HTML allows reference to be made to non-HTML documents, many of these difficulties can be overcome: an HTML browser such as NETSCAPE can be configured to invoke an external renderer if no internal renderer is suitable for the entity referenced, and in that way both DVI (from TEX) and POSTSCRIPT documents can be referenced from, and displayed from within, an HTML document. Since both DVI and POSTSCRIPT are equally suited to the accurate representation of mathematical material, there is no real reason why a mathematical document should not be displayed from within an HTML framework by an HTML browser configured with a suitable external renderer. PDF, on the other hand, has no need for external renderers, since its native mode of operation uses a strict subset of POSTSCRIPT; indeed, Adobe Acrobat is intended to be configurable *as* an external renderer for HTML browsers such as NETSCAPE! By using Adobe's 'multiple master' font technology, Acrobat can generate a reasonable substitute for any font specified in a PDF document, even if that font is not available within the system on which the document is being displayed. It is by no means unlikely that before very long a DVI-to-PDF driver will emerge, and in the true tradition of TEX it is also extremely likely that such a driver will be placed in the public domain; DVI-to-HTML is an unlikely eventuality, however, since by the time a TEX document has been converted into DVI, too much information has been lost to allow the high-level structure of the document to be re-created.

On the other hand, we can certainly envision a format being created for TEX which embeds `\specials` in the DVI file to convey information about the high-level structure of the source document: since the user interface would be completely unaffected by the presence of these specials, such a format could appear to the user exactly like any of the present formats or format variants which support appropriate high-level markup ($\mathcal{AMS}$-TEX, LATEX, $\mathcal{AMS}$-LATEX, LA$\mathcal{MS}$-TEX, etc.). Such specials could then be directly mapped into HTML constructs, and thus a TEX-to-HTML route is neither impossible nor unlikely; indeed, it is surprising that no such extended format has yet been announced (at least, to my knowledge). Finally it is worth remembering that HTML is essentially a *distributed* markup language; it is primarily intended for documents which need to reference other documents which may be anywhere on the Internet; PDF, on the other hand, is essentially Internet-unaware, and whilst it can transparently reference other documents that are visible through (say) NFS (or, using ALEX, anonymous FTP), it makes no assumptions that documents might be anywhere other than the local filestore or on a Microsoft-compatible network. [This last sentence is somewhat tentative: in the absence of the definitive PDF specification, it is somewhat difficult to accurately interpret the claim that Adobe Acrobat allows one to "Add value, set priorities and maintain a dynamic information network with links, bookmarks, notes and connections to external applications and documents', but I suspect that the 'dynamic information network' does not allow the transparent referencing of arbitrary Internet resources, although this may well come in time.]

Computer Typesetting or Electronic Publishing: Pros and Cons

Computer typeset material, particularly that typeset using TEX or a functionally equivalent system, represents the finest in typeset quality that can be easily accomplished today; where such computer typesetting software is unavailable, comparable results can only be accomplished by a skilled professional using either old-fashioned technology (e.g. hot lead) or a modern but proprietary system. This is not to say that the use of a system such as TEX *guarantees* professional quality results: there are far too many counter-examples in existence which demonstrate that in completely unskilled hands, TEX and comparable systems are capable of generating absolutely appalling results. None the less, in reasonably skilled hands, and/or using a format package which prevents the author from making design decisions, TEX is capable of generating results which meet the highest professional standards, particularly in the field of mathematics where TEX essentially performs as an 'expert system'. The disadvantage of such a system is that in its intermediate form (DVI), a TEX document is not fully portable: a DVI file contains references to, but no instances of, fonts; at the point where the DVI file is converted into its final form (usually paper, but on-screen preview is now also ubiquitous), the same fonts which were used to create the document must be available in order to render it correctly; in their absence, only a poor approximation of the intended document is possible. [It is worth noting that the creator and the viewer/printer of a TEX document need a common set of fonts, but each needs a quite different representation of those fonts: the creator needs only the *font metrics*, which specify the height, depth and width of each glyph, and kerning and ligaturing information for the glyph set; the viewer/printer of the document can normally get by without the metrics, but instead needs the actual glyph set, either as bitmaps or as outlines.]

Electronic publishing, on the other hand, and particularly e.p. accomplished through the medium of HTML, does not place any emphasis on the quality of the end product: indeed, HTML voluntarily cedes control over the appearance of the final document to the browser used to render it, although there are some placement options which allow the author a little control over the final appearance (and there are considerably more such options in HTML 3). Within an HTML document there is no font information *per se* (again, this is true only of current HTML: HTML 3 adds the concept of style sheets, which will "[...] eventually lead to smart layout under the author's control, with rich magazine style layouts for full screen viewing, switching to simpler layouts when the window is shrunk"); instead the document consists of a set of high-level markup tags, which are mapped by the browser to a particular font or font variant. Whereas a DVI file is a monolithic entity, and makes no reference to any external resources other than fonts, an HTML file is frequently little more than a container for other HTML files, and may make reference to an extremely wide range of resources (further HTML files, images, AFS files, Usenet newsgroups, e-mail addresses, FTP-accessible files, etc.) which may be anywhere on the network, and which may themselves contain further references and so *ad infinitum*.

PDF is essentially a reasonable compromise between the two: the creator of the document specifies its appearance, and the PDF reader then displays that document to the best of its ability: if the fonts needed to display it properly are embedded, or if they are resident on the target system, then the document will be displayed exactly as the author intended; if the fonts are not accessible, then Adobe's proprietary 'multiple master' technology will be used to interpolate a substitute for the missing font(s) which allows the original line-breaks, leading, etc., to be retained. A PDF document may reference further PDF documents, but these are assumed to be available on the local filestore; there is no apparent support for the automated fetching of remote Internet documents, although the absence of the PDF documentation at the time of writing makes analysis of this feature rather more of an informed guess that a definitive statement.

All three formats discussed allow searching to be conducted; within a DVI file there is no intrinsic support for indexing, but it would not be at all difficult for a DVI viewer to create a dynamic index to the document being viewed. Both HTML and PDF allow fully indexed documents to be referenced.

Publication in the Twenty-First Century

It is no longer possible to assume, as countless previous generations of scientists have done, that "publication" involves printing on sheets of paper which are ultimately distributed as a part of a journal or as a book: increasingly both economic and environmental pressures will dictate that only essential information be committed to paper, and anything even slightly ephemeral will be restricted to electronic distribution. At least two *de facto* standards have already emerged for electronic publication: HTML, which originates in the distributed and anarchic world of the Internet; and PDF, which originates in the commercial world. At the time of writing, HTML is the better established, and two freeware browsers are widely used (MOSAIC and NETSCAPE), with a third (ARENA), being developed specifically to support HTML 3; for PDF, there is only one reader currently available (Acrobat), and that too is classified as freeware. HTML devolves to the browser most of the decisions concerning the actual appearance of a document; PDF allows the author to make most of those decisions, but reserves the right to substitute interpolated fonts if the genuine article are not available at the point of rendering.

HTML is essentially a distributed protocol, and will allow bibliographies to reference cited texts no matter where they are in the world (so long as they are on-line), thereby adding truly incalculable value to the bibliography of a document; PDF, it would appear, is essentially a local protocol; whilst bibliographies could still cite full-text sources, those sources would need to be available to the system on which the bibliography is being read.

Many issues remain to be resolved before the world can truly move to electronic publishing as the mainstream form: Internet access in every home, office, library, vehicle, and restaurant will be just a start. There remains the very contentious issue of copyright: whilst there are usually economic costs associated with the photo-copying of a printed document, the costs of copying an electronic document are virtually nil, and therefore the enforcement of copyright for electronic publications is a major concern. It is highly likely that some form of encryption and licensing will emerge to prevent the unauthorised copying and/or re-distribution of electronic texts. From the psychological and physiological point of view, displays will need to become significantly better (in many senses: weight, resolution, glare, portability, etc.) before the electronic book completely replaces the printed equivalent: few of us, going on holiday today, would choose to take a notebook computer with a CD ROM containing the complete works of Shakespeare in preference to a couple of (disposable) paperbacks... Although originally developed as front ends for the generation of printed material, typesetting systems such as TEX will almost certainly have a major rôle to play as front ends for electronic publishing, since (for example) the linear representation of mathematical formulæ is equally convenient and applicable whether one's mathematics are eventually to appear on paper or on a computer screen. Within ten years, HTML and PDF will appear *passé*: new standards emerge faster than most of us can keep up, and today's technology is tomorrow's door-prop. But the future of the book (or even the newspaper) as the normal means of communication is surely as doomed as that of the petrol-driven car as the normal means of conveyance; the first to guess exactly what form the replacement will take may become as rich as today's newspaper magnates and publishing house principals; or perhaps the converse will occur, and the Internet will finally cause the collapse of the publishing empires, as academics and authors suddenly realise that they are no longer beholden to the few. Self-publishing may become the norm, or peer review may take on an entirely different form; perhaps a two-tier hierarchy of electronic publishing will emerge, with unrefereed papers being available via each academic's home page whilst those that have survived the refereeing process will be available from prestigious and highly accredited archives. What is certain is that almost all of the readers of this paper will find out for themselves what the future holds, at least as regards computer typesetting and electronic publishing: the future is just around the corner, and approaching at an ever increasing speed.

Addendum

Within the last forty-eight hours, I have learned of two new facts which significantly impinge on the material above: NETSCAPE have licensed the use of PDF technology from Adobe, which will allow them to incorporate a PDF renderer within their HTML browser, and Michel Goossens & Sebastian Rahtz have demonstrated the feasibility of using Adobe's 'multiple master' fonts from with TEX; further details of the latter, including very useful information on multiple master fonts, are given in the *Baskerville* issue cited in the Bibliography.

Acknowledgements.

I would like to thank Professor Adam Jakubowski and Jerzy Ludwichowski for making it possible for me to present this paper, to Elżbieta Kuczyńska and Bogumiła Rykaczewska-Wiorogórska (University of Warszawa) for kindly providing two alternative translations of the abstract into Polish, and to Professors Adam Jakubowski and Andrzej Jonscher for providing reverse translations into English to enable me to check the accuracy of the initial translation. I would like to thank Dr Warren Dicks of the Autonomous University of Barcelona for his analysis of the problem of the number of visually distinguishable configurations of an $m \times n$ matrix, as used to establish the number of distinct characters which can be generated by a 7×5 dot matrix printer. I would like to thank Dr Frank Quinn of Virginia Tech. and the American Mathematical Society for granting me permission to reproduce extracts from their documents on "The New Media" and "Non-traditional forms of publication", and finally I would like to thank Barbara Beeton of that same Society for allowing herself to be persuaded to review the paper before publication and for her many helpful comments; needless to say, any errors which remain are solely my responsibility.

Bibliography

I have not given formal references in the text, since I feel that they are inappropriate in a paper of this nature; however, the following short list of publications may be of interest to those who wish to pursue further the topics discussed here.

Mathematical Typography by Donald E. Knuth, *Bulletin of the American Mathematical Society* (new series) 1 (March 1979), 337–372. [Reprinted as part 1 of *TeX and METAFONT: New Directions in Typesetting* (Providence, R.I: American Mathematical Society, and Bedford, Mass: Digital Press, 1979).]

Tau Epsilon Chi, a system for technical text by Donald E. Knuth, Stanford Computer Science Report 675 (Stanford, California, September 1978), 198 pp. [Reprinted as part 2 of *TeX and METAFONT*, the book cited above.]

The WEB *system of structured documentation* by Donald E. Knuth, Stanford Computer Science Report 980 (Stanford, California, September 1983), 206 pp.

Literate programming by Donald E. Knuth, *The Computer Journal* 27 (1984), 97–111.

Using Adobe Type 1 Multiple Master fonts with TeX by Michel Goossens and Sebastian Rahtz, *Baskerville* Vol. 5, No. 3 (UK TeX Users' Group, June 1995, ISSN 1354-5930), 4–8.

HTTP: A Protocol for Networked Information `http://www.w3.org/WWW/Protocols/HTTP/HTTP2.html`

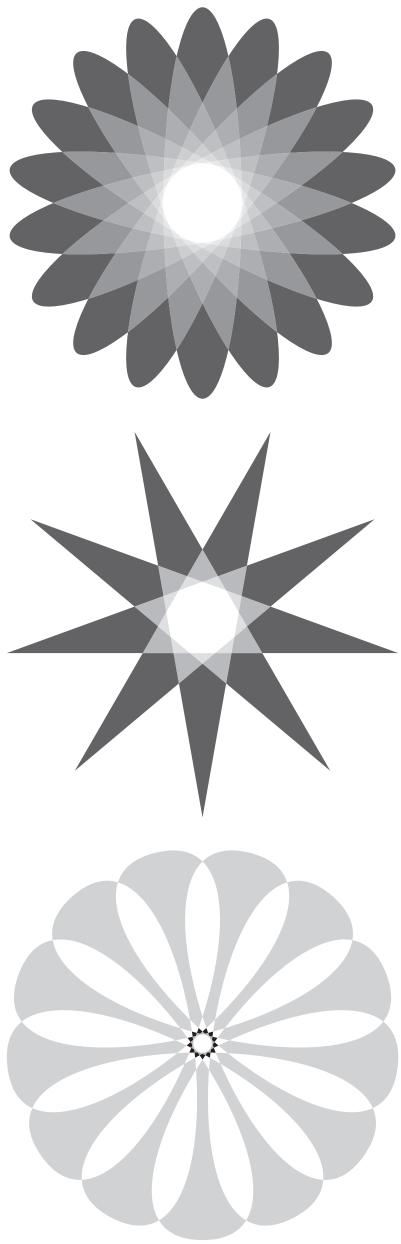
A Quick Review of HTML 3.0 `http://www.w3.org/hypertext/WWW/Arena/tour/start.html`

HyperText Markup Language Specification Version 3.0 `http://www.hpl.hp.co.uk/people/dsr/html3/CoverPage/html`

Document Type Definition for the HyperText Markup Language (HTML DTD) `http://www.w3.org/hypertext/WWW/MarkUp/html3-dtd.txt`

Adobe Acrobat `http://www.adobe.com/Acrobat/Acrobat0.html`

◇ Philip Taylor
`P.Taylor@Vms.Rhbnc.Ac.Uk`



Te „kwiatko-gwiazdki" to przykłady grafiki METAFONT-owej wygenerowane za pomocą pakietu makr MF-PS (Bogusław Jackowski et al.). Zwróćcie uwagę na obecność figur jednokreślnych.

Pakiet zawierający powyższe przykłady jest dostępny w archiwum GUST-u:

`ftp.pg.gda.pl:`
`/pub/TeX/GUST/contrib/MF-PS`