Computerization of the IUCr Editorial Office, Chester: a Progress Report

The Technical Editor's Office, International Union of Crystallography, 5 *Abbey Square, Chester CH*1 2*HU, England*

19 *October* 1990

Abstract

Recent acquisition of modern computer equipment has enabled the Editorial Office of the International Union of Crystallography to undertake various initiatives in electronic publishing and data analysis. This report is designed to inform Co-editors, members of the Executive Committee and other interested parties of the main areas of development currently addressed, in order to supply background information against which to debate future policies and directions of change. The main topics covered are the checking of structural data for structures submitted to *Acta Crystallographica* (and *JAC*) and the in-house composition of papers from author-supplied electronic files. These applications have been developed as part of the CIF project, which is nearing maturity and should allow for files conforming to this standard to be accepted as publication material within a few months. The prospects of future development are also discussed.

I. Introduction

Crystallography has long relied on the power of digital computers, and practitioners of the science have been accustomed to using the most advanced hardware and software in their work. Many crystallographers have readily adopted word processors and other software on personal computers, and are familiar with the use of network electronic mail to communicate with their colleagues. It must therefore seem natural that the Editorial Office of the IUCr should investigate the usefulness of computers in the production of crystallographic journals and books. Many initiatives in this area are now being undertaken, and there appears to be good grounds for believing that many advantages will follow from an enthusiastic development of computerized production procedures. However, the editorial staff of the Union are small in number and have little direct experience of the technologies and programming skills that may be needed. Despite this, good progress has been made in introducing various software systems that may radically change the way in which the journals *Acta Crystallographica* and *Journal of Applied Crystallography* are produced, and that will have substantial implications on the future handling of crystallographic data in journals and databases. It is important that we approach the tasks in hand systematically and with a proper sense of priorities.

The manner in which the efficient and productive implementation of new methods and techniques is managed will be of interest to various Committees and Commissions of the Union. Some of these bodies will have direct responsibility for overseeing these developments; others will wish to make constructive suggestions. This report has been prepared for the benefit of interested parties, partly as a review of what has already been achieved, partly as an indication of the options now available. It is offered purely for the sake of information, and any comments that might appear to reflect on matters of policy are intended only to reflect the relevant technical background, not to offer any judgements on the policies themselves.

This report is lengthy and somewhat detailed because it aims to discuss the major aspects of the computerization of the Editorial Office in enough depth to be of use to the Commissions and Committees who need to make decisions on future directions. It comprises a section reviewing the introduction of the new technology and describing the hardware capabilities of the office; a section on the procedures being developed for the centralized checking of structural data; and a discussion of text formatting issues. Some comments on possible future directions are also offered.

II. Historical background

Computer-based operations in Chester began in 1984 with the acquisition of a Systems Group 2900 series machine running under the OASIS operating system. This machine had a 512 kB memory partitioned into 64 kB segments allowing access by up to 8 users, and 34 MB of hard disk storage for a 'flat' (*i.e.* non-directory-based filesystem). A system of programs in the OASIS compilable BASIC language was commissioned to maintain records of manuscripts being processed, prepare and sort annual and cumulative indexes, store formulae of structures reported in *Acta* and search for duplicate submissions, and maintain a print spooler. These programs, which shall be collectively referred to as the journal housekeeping software, were reasonably effective. A typical screen display from one of these programs is reproduced as Figure 1. However, the operating system was temperamental and had a frequent tendency to hang, and the hardware was unreliable. Further, the operating system and i/o media formats were unusual, and FORTRAN (and C) were not supported, so that there was very limited scope for information interchange with other systems.

In 1988, a desktop publishing graphics workstation was acquired from Rank Xerox. This machine (the Documenter) has various utilities available under the Viewpoint operating system specifically for document formatting and composition. It has a very user-friendly interface, and the ability to handle moderately complex document layouts. It was used to produce *Fast Communications* in *Acta* throughout 1989 and for the first half of 1990, and is still used for various types of in-house documentation. Its main disadvantages are that it

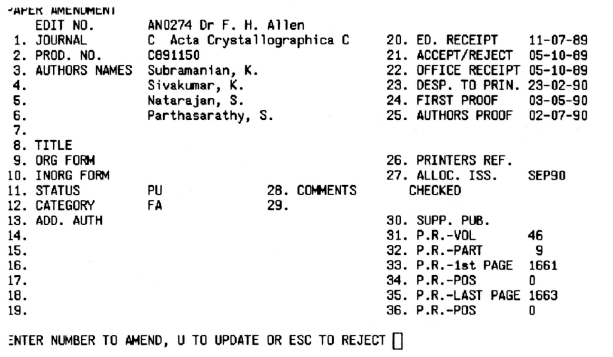

Fig. 1 Typical screen display for database entry as developed on Systems Group machine and subsequently transferred to 3B2/500.

is relatively slow (it is based on the 80186 chip) and output is limited to 300 dpi laser-printer copy, which we have used as camera-ready copy in the journals. The relative inadequacy of the quality of the final printed page is apparent. The Documenter also provides an IBM-PC emulator and can read DOS formatted 5 1/4" floppies, which gave us our first opportunity for import of files from other systems.

In October 1988, an 80386 DOS machine (Olivetti M380/C) was purchased with 640 kB memory and 80 MB hard disk. This can read and write high-density 5 1/4" and 3 1/2" diskettes and has a VGA colour screen. Thus we now have full compatibility with the DOS-based personal computer community. The machine has a BASIC interpreter and runs a variety of applications written in FORTRAN. Until the middle of 1990 its main function was to run the structural checking programs used in the pilot scheme to carry out checking in Chester. This machine was also used to carry out initial tests with the TEX programming package.

At the same time as the PC was delivered, an AT&T 3B2/500 minicomputer was installed to act as the main office computer. This machine has 4MB of memory, 147 MB hard disk, and can support multiple users (currently up to 18 serial ports may be used) under the UNIX operating system. Data transfer is possible *via* 5 1/4" floppy disk (formatted in a machine-specific way), 1/4" cartridge tape unit and a Hayes-protocol asynchronous dial-up modem rated at 2400 baud. There is also an Olivetti PG303 PostScript laser printer (which is functionally similar to an Apple Laserwriter). The operating system is standard AT&T System V.3 UNIX, and the languages supported are C, FORTRAN and UX-BASIC. This last is compatible with OASIS BASIC, and allowed the transfer of the journal housekeeping software from the Systems Group machine with remarkably few problems. The transfer of programs and data was complete by early 1989, when the old machine was shut down. The journal housekeeping programs were enhanced and complemented by procedures to format the annual indexes in the PostScript language, to maintain on-line indexes, generate form letters and statistical summaries, and various other utilities. Figure 2 illustrates one of the menus available to users for navigating through the filesystem and performing desired tasks. Most of these procedures were written in the UNIX shell language and using operating-system text management utilities (*sed*, *grep*, *cut*, *paste*, *awk*), because these were the only programming tools available to us at first. C and FORTRAN compilers were ordered, but the length of time required to ship these to us was scandalous (almost 18 months for the FORTRAN). It is primarily for this reason that structural checking was done on the PC. The 3B2 continues to give good service, its reliability is excellent, and it is a tool shared in common by the entire office staff.

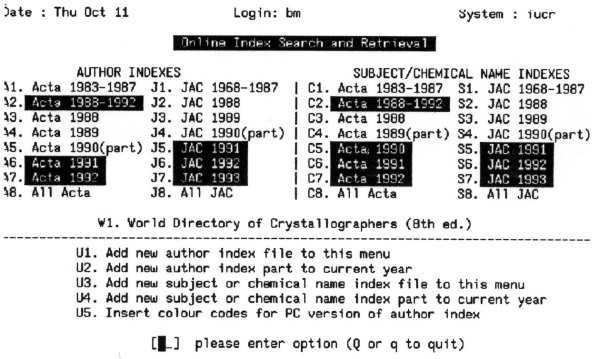

Fig. 2 User menu on the 3B2/500 giving access to online index files and to utility programs for manipulating these files.

In 1990, an Apple Macintosh IIcx was purchased to allow receipt of files from the Mac-based community of personal computer users, and as a general secretarial machine. This is used for word processing using Microsoft *WORD*. It has its own Laserwriter, another PostScript machine. Most recently, a cluster of four SUN computers was purchased, comprising a SPARCserver 330 with 8 MB memory, 667 MB hard disk, and three SPARCstation 1+ graphical workstations, each with 12 MB memory and 208 MB hard disk. These are networked on an Ethernet bus, and share a 150 MB 1/4" cartridge tape unit and a 1/2" 1600/6250 bpi tape deck. This last gives us full access to the academic community for data transfer by tape. We also have a synchronous modem and Kilostream link to Daresbury Laboratory which will allow us access to the UK JANET academic network in the near future (and thence *via* gateways to EARN, BITNET, USENET *etc etc*). These machines run under Sun OS 4.1 (an enhanced version of Berkeley 4.3 and System V UNIX) and are fast, employing RISC architecture. The graphics screens (1 colour, 2 monochrome) allow windowing operations under the SUNVIEW and OPENWINDOWS windowing systems. All machines have C and FORTRAN compilers, the latter of which is said to be highly compatible with VAX FORTRAN. These machines will be used to take the full burden of checking and electronic publishing. Already, *Fast Communications* are being set on a SUN workstation using the *Publisher* package, and the checking programs have been transferred to the SUN cluster and expanded.

At present, the 3B2 is linked to the PC, Mac and the SUN cluster by RS-232C connections and can transfer data using *Kermit* (da Cruz, 1987), *PC-Interface* or *uucp*, as appropriate. From an operational point of view, it would be useful to have a networking connection between the 3B2 and the SUN machines (ideally the NFS protocol which already links the SUN machines), but consideration of this has been deferred on grounds

of cost (the relevant hardware and software for the 3B2 would amount to something in the region of £6000). At present data transfer between the UNIX systems is satisfactorily achieved by the *uucp* protocol, so that the 3B2 can supply print spooling services to the SUN machines, for example.

III. Centralized checking

In a sense the entire editorial process has to do with checking—checking for clarity, consistency and style of presentation. It is a logical extension to check wherever possible the *accuracy* of the manuscript. For some considerable time, a number of checks have been performed that go beyond the usual editorial concerns of a publisher. These have included checks for resubmission by an author of a paper already rejected by one of the Co-editors, and searches for previous reporting of identical or similar structures in the literature. These are reviewed briefly below. We are now beginning to investigate the scientific accuracy of the reported structure, and our procedures for doing this will be presented at some length.

Duplicate author resubmission

A list of rejected papers is maintained and a simple test is run to see whether authors of newly submitted papers match any in that list. This is a simple check to guard against submission to one Co-editor of a paper rejected by another. The criterion for a 'hit' (that there should be a match of at least one author's name) is very conservative, but if this check is run regularly, it involves very little work to eliminate the large majority of cases which do not represent a resubmission.

Duplicate structure reporting

Structures reported in Acta

A database is maintained of formulae for structures published in *Acta*, and new compounds are tested against this. The formulae are represented in a simplified manner, by residue for organic compounds and by elements for inorganic. Similar as well as identical structures are found. The database is not complete, extending back only to about 1982. This check is run manually when each updated Co-editor's status list reaches us, and is fast.

Structures reported elsewhere (and in Acta *before 1982)*

Possible earlier reports are searched for in the Cambridge Structural Database and in the Inorganic Crystal Structure Database. These are currently accessed *via* the *CSSR* software on the crystallographic VAX at Daresbury Laboratories, on which we have a guest username. Connection to this machine is *via* dial-up modem, and early work was hampered by noisy telephone lines which restricted the data communications rate to 300 baud. Subsequent digitization of the telephone exchanges now allows us to operate at 2400 baud, which makes it feasible to widen the search criteria used and to examine graphical representations of structures on Tektronix-type screens. It is anticipated that the Cambridge Database will be accessed directly across the network using a graphics-based interface in the near future. This form of checking has been carried out routinely over the last two years.

Structure checking

The major innovation in checking procedures is the introduction of programs that will allow geometry and symmetry checking to be performed in Chester. A pilot scheme involving some half-dozen Co-editors has been operating since summer 1989. A modified and enlarged version of this scheme is now in place and undergoing further development. It is expected that centralized checking of structures will be offered to all Co-editors in the near future, as soon as staffing levels make this possible. The intention will be to provide an assessment of the structure to the Co-editor. Since the Co-editor retains the responsibility for deciding whether a structure should be accepted, it will probably be useful to give a detailed description of the procedure currently used.

Input of data

The information required by the checking programs we use is entered into a crystallographic information file (CIF) by secretarial and editorial staff, from the submitted typescript. Only the information required for checking is entered in this way; however, the motivation behind the use of a CIF is to allow exactly the same procedures to be run on a CIF supplied directly by an author. In this latter case, the subset of data in the CIF corresponding to the data we enter for checking can be extracted automatically by the same software as is used on our 'home-made' CIFs. For the sake of convenience, our checking CIFs are constructed on the 3B2 computer and e-mailed to a dedicated login on a SUN workstation; this will again parallel the treatment of author-generated CIFs submitted by e-mail.

The user interface

The dedicated login on the SUN machines has a customized execution environment and a graphical interface that are designed to provide a robust and easy-to-use procedure. Incoming CIFs are represented by crystal-like icons in a reception directory CIF_IN. A mouse is used to move a pointer over one of these icons. Double-clicking a mouse button initiates a procedure which creates a subdirectory for each structure recorded in the CIF, places in that subdirectory a series of input files containing the relevant data correctly formatted for each program used, and runs the programs with pre-defined default parameters. Figure 3 illustrates the display to the screen during this process.

If all runs successfully, the CIF is moved to a CIF_OUT directory for archiving or further processing. In the directory created for each structure, the output files are available for inspection, the input files may be edited and 'double-clicked' to rerun individual programs with different parameters, and additional programs are available from a pull-down menu (see Figure 4). Different icons are used both to differentiate the various categories of files and to associate with each a different application. Thus, selecting an input file (represented as a deck of punched cards!) runs the associated program; selecting an output file opens it in a screen-based editor; selecting a 'CD' binary file invokes the relevant binary file editor; selecting a PostScript file (indicated by the PostScript logo) opens a PostScript previewer and source editor; and so on. All i/o redirection and file assignments are handled automatically, and need not trouble the user.

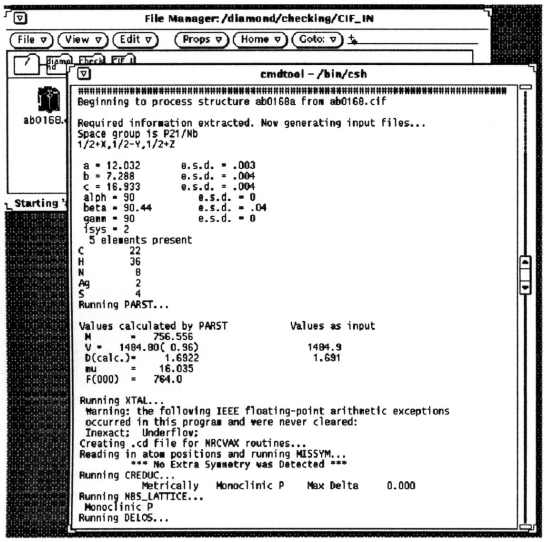

Fig. 3 The user has double-clicked his left mouse button while holding the cursor over the icon representing file ab0168.cif. A command tool appears on screen, to which comments are written as the file is processed. This provides a means of monitoring the checking in real time, and summaries of some of the checking results may be read as they are produced. Full results are stored in a set of files within directories constructed for each reported structure.

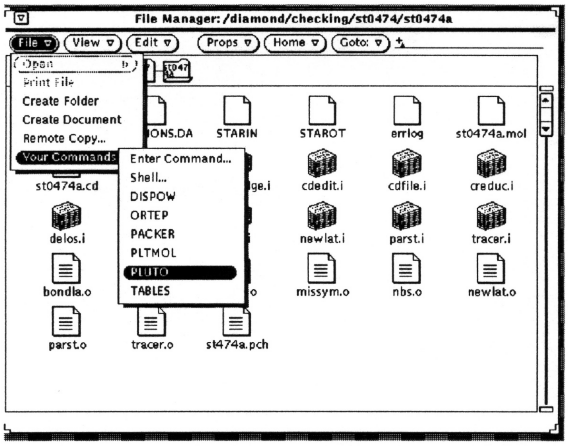

Fig. 4 The graphics program *PLUTO* has been selected from the *Your Commands* option of the File Manager's *File* menu. When the (right) mouse button is released, a full-screen Tektronix emulator will be created, in which the chosen program will start up. This particular application is a SUNVIEW tool, showing how the two windowing systems available on the workstation can coexist. Other checking programs can be started up by double-clicking on the icons (decks of punched cards) representing their input files.

The process of constructing input files and running programs with standard defaults could, of course, be done entirely automatically and in batch mode. This is likely to be done as the checking load increases. At present, however, it is felt to be useful to staff to monitor the entire procedure automatically.

Programs in use

Geometry checking

PARST The Parma structural package (Nardelli, 1983), developed for the checking of structures reported in *Crystal Structure Communications*, has been supplied to us through the kindness of Professor Nardelli. This program provides facilities for checking essentially every aspect of the geometrical relationships between atoms in an input coordinate list. By default, lists are produced of orthogonal coordinates, bond distances and angles and torsion angles, interatomic contacts, intermolecular contacts and possible hydrogen bonds. The reduced (Niggli) matrix is also output.

Options exist for calculating H-atom positions, least-squares weighted planes and straight lines, ring puckering coordinates and stereographic projections and comparison of subsets of atoms. None of these options is at present exercised on a regular basis. The principal axes of thermal ellipsoids may also be calculated.

One frequent fault in submitted structures is the presentation of a set of coordinates describing the asymmetric unit which contains atoms from separate molecules in the unit cell. *PARST* does not try to build a unique molecule in these circumstances, which can be a nuisance to us in trying to recognize the connected molecule. Nevertheless, most of the bond distances in such a case can be found in the list of intermolecular distances, which contains symmetry-generated atoms. However, in the case of a non-connected molecular unit, bond angles cannot be retrieved by the program. Despite this limitation, *PARST* is one of our most useful checking tools because of the very wide range of geometrical relations which it generates.

BONDLA (XTAL) The routine *BONDLA* within the *XTAL* crystallographic system (Hall & Stewart, 1990) also generates geometrical relationships, but is restricted to consideration of bond distances and angles, contact distances and torsion angles.* Its advantage over *PARST* is that it *does* seek and report the coordinates of sets of atoms comprising connected residues. This routine has only recently been implemented and is currently under evaluation. Another advantage it has over *PARST* is that atomic radii may be redefined in the input list; at present the default atomic radii are chosen for compatibility to be those compiled in to *PARST*.

DATCHK/UNIMOL The Cambridge Crystallographic Data Centre (CCDC) has recently supplied us with the source code for their checking programs (Allen et al., 1974). These are very comprehensive and powerful, and check every aspect of the data that will be entered into the Cambridge Structural Database. As such, they are more detailed than we require for our present purposes. However, one aspect of the CIF project is intended to be the ability of the IUCr to provide database maintainers with data of very high quality, and it is therefore likely that we shall develop techniques for applying most of the checks within this package to full CIFs submitted to us. At the moment, the major item of potential interest within this package is its ability to compare bond lengths automatically, as its major diagnostic of the accuracy of reported coordinates. If we are to take advantage of this, we shall need to invest more time in the secretarial input of bond distances, and some editorial time in inputting a description of the chemical connectivity of the structure. The bonus should be an

* Other geometrical calculations can be performed by additional routines in the *XTAL* program system, a copy of which has just reached us.

overall saving in editorial time. It is envisaged that the full implementation and development of this approach will be undertaken in collaboration with the CCDC.

DISPOW/DISANG (NRCVAX) The program package *NRCVAX* has been made available to us by the kindness of Dr E J Gabe and co-workers (Gabe, Le Page, Charland, Lee & White, 1989). It contains many useful routines for the handling of various aspects of crystal structure analysis. The modules *DISPOW* and *DISANG* generate bond distances and angles. Together with the unique molecule building routine *UNIMOL* within the *NRCVAX* package these can provide geometric details of a connected residue. This program is not run routinely (there is no particular reason for this; one must simply make an arbitrary choice when one has such a cornucopia of useful programs!) but is accessible through a pull-down menu.

MOLDRAW This is the only program we use that is not implemented on the SUN workstations, because it is written using PC-specific graphics routines (Ugliengo, Borzani & Viterbo, 1988). (The input file is, however, generated automatically on a SUN.) It is a molecular visualization program, and generates static or animated views of molecules, molecular fragments, unit cells and even extended crystal regions. The user can generate stereo views, stick, ball-and-stick or space filling models in arbitrary orientations. Distances and angles between specific atoms can be requested interactively. Although the program itself performs no analysis, editorial staff find the flexible visualization of cell contents extremely helpful in interpreting output from other checking programs. We are grateful to Professor Ugliengo and his collaborators for allowing us to use this program.

PLUTO/PLTMOL/PACKER (NRCVAX) As was mentioned in the comments to *MOLDRAW*, visualization of the molecule is often very helpful. These graphics routines supplied within the *NRCVAX* package may all be run on the UNIX machines to allow immediate visualization.

Space-group misassignments

Whereas the geometric checking programs contain information relevant to the space group chosen for the structure refinement, a number of programs are also used to test the lattice symmetry. Most of these consider the generation of a reduced cell and the consequent implied lattice. The different programs do not always converge to a common answer: this is sometimes due to the algorithm used or to the conventions for reporting a unit cell. More often, however, it has to do with the allowed tolerances on symmetry elements. One of our major learning tasks will be the recognition of the circumstances in which the default tolerances presented by the programs should be modified, and to what extent they may be changed. In the meantime, we run several of these programs and compare the output results.

NEWLAT This program (Mugnoli, 1985) assigns an empirical figure of merit to a set of new lattices generated metrically from a given initial unit cell. This has the advantage for us of listing several alternatives that are consistent with some specific experimental tolerance. Derived lattices are listed in decreasing order of figure of merit within each candidate crystal system.

CREDUC (NRCVAX) and LEPAGE These programs use the same algorithm for cell reduction. *LEPAGE* (Spek, 1988) is an interactive PC program, which we now use as a swift check of papers that have not otherwise been checked as part of the 'full service' on receipt.

*NIST*LATTICE* This program (Mighell, Hubbard & Stalick, 1981; Himes & Mighell, 1987) has recently been sent to us by Dr Mighell of the National Institute of Standards and Technology, and is now undergoing evaluation. It is part of the checking procedure which NIST run on entries for their Database, and contains provision for detailed indexing and analysis of powder patterns. For our specific requirements at present, its major contribution is the automatic identification of a lattice type from the components of the Niggli matrix. This has previously been done manually, but the programmatic approach handles tolerances automatically when e.s.d.'s of the unit-cell parameters are supplied.

DELOS This program (Burzlaff & Zimmermann, 1985), which was circulated to contemporary Co-editors some time ago, determines lattice parameters of the conventional standardized unit cell and evaluates the probable Bravais lattice following the method of Delaunay.

TRACER This is an old lattice transformation/cell reduction program which has been integrated into the driving software of some diffractometers (Lawton & Jacobson, 1965).

MISSYM (NRCVAX) This is a particularly powerful routine (Le Page, 1982,1988) which generates metric symmetry elements according to the method of *CREDUC*, and then applies the symmetry transformation implied in the space-group description to the atomic coordinates of the input atom list, to search for possible additional symmetries. The program will list additional symmetries or pseudosymmetries which can be referred back to the author for consideration. Generous tolerances are built in to the program and options are available for excluding specified atoms or atom types from the search list, so that the results are conservative in the sense that additional symmetries are likely to be indicated more often than they actually exist. However, the corollary is that few genuine cases should be overlooked. It may be that asking an author to account for any positive result thrown up by this program will improve the usefulness of the discussion in the paper, even when it does not result in a different space-group assignment.

IV. In-house typesetting

One of the immense benefits perceived in current developments in computer technology is the growing availability of programs that can in effect compose and typeset text (and sometimes graphics) to produce high-quality printed output (though at lower resolution than is obtainable from phototypesetting machinery). There are two aspects of this new technology that we should address. These are the development of a powerful and flexible system that we can use to produce printed output conforming to the quality standards of a respected academic journal; and the ability to handle text processed by authors using their own (perhaps arbitrary) packages. We shall look at the first of these topics—the

need to be able to generate printed output of the same quality as a commercial typesetter—by looking in detail at two applications (the text formatting software TEX and the output format PostScript) that we use in-house. A particular implementation of a powerful TEX-based formatting system will also be described.

TEX

The program chosen as the test-bed for the development of in-house composition is TEX (Knuth, 1984). This has three principal advantages:

1. It is a comprehensive typesetting package — it addresses concepts such as font kerning, interline leading, widow and orphan lines, and the specific requirements of mathematical setting. Many commercial programs suffer from the limitation that, though you can print lines of 10 pt type 12 pt apart, you may not be able to place them 11 pt apart. This is inconvenient if your journal has always been printed in 10/11 pt.
2. It is cheap, widely available, and produces identical results on a wide range of machines. This is particularly important in respect of accepting TEX files generated by authors, but is also useful in allowing us to process the same file on any of our machines.
3. It is macro-based, and can be run as a background job. This is of particular significance to the way in which we intend to process CIF documents. The *macros* referred to above are code words that represent procedures that may be defined by the user. These procedures may be simple font changes, they may substitute the word immediately following by some other phrase, or they may be full-fledged programming constructs which redefine the entire format of the document subject to conditional tests. Hence, a stream editor (at present) or some other program can be used to extract from a CIF the data items that will form the text of a paper, and append or prepend to each such item a macro name which will correctly format that item. The entire derived source file (containing a string of data items with embedded macro calls) may be processed automatically by TEX, to produce a first draft of a printed document without any human intervention.

Several of these points will be developed in more detail below. However, it is worth mentioning some of the disadvantages of TEX.

First, the manner of processing text as a stream of characters with embedded macros produces a source document that is more like a computer program than a manuscript. Indeed, it is probably better to refer to TEX as a programming language, rather than as a program. This makes it somewhat unfriendly to the user, who must learn the syntax as well as the vocabulary of a new language. Further, there is no direct visualization of the finished document as the user works on it — the source file must be periodically processed by TEX and the output displayed by an on-screen previewer on a high-resolution graphics screen, or printed out. This is in contrast to the 'what-you-see-is-what-you-get', or 'WYSIWYG', approach of many popular packages, where an on-screen representation of the printed page is manipulated directly with the help of a mouse. Therefore, although the first draft of CIF-derived documents may be produced effortlessly, subsequent editing (which is likely often to be extensive) will be more laborious.

Secondly, although TEX has many more capabilities than most word-processing or desktop publishing (DTP) programs, it still has its limitations. Its mode of operation is 'linear', in the sense that it assembles a notional column of text, which it then breaks at relevant points to compose pages. However, a page of print (especially one with two columns of print and tables or graphics of arbitrary width) is a two-dimensional entity, and TEX is unable to handle pages of arbitrary complexity.

Thirdly, it is intended to act as a text processor, and thus the inclusion of graphics is not necessarily a straightforward task. However, although there are some DTP packages which can easily handle *certain* types of graphical material, there is always a problem with incorporating *arbitrary* graphics and text.

These limitations can be alleviated to some degree by making use of a commercial text-formatting package called *The Publisher*, which combines the power of TEX with a user-friendly front end and screen previewing facilities. This software package will be discussed in greater detail below.

TEX—the program

It is useful to discuss some aspects of TEX that differentiate it from most other DTP applications. Its very inception is interesting. Donald Knuth, a professor of computational science at Stanford University, was engaged in writing a series of textbooks on programming, and became frustrated by the large number of errors in typeset galley proofs. He determined to write a program that would enable him to perform his own typesetting to the same high quality as encountered in commercial bookwork, and that would be capable of handling complex mathematical setting. TEX was the result. However, as befits the computer scientist, the program that was produced was designed to have widespread applications, and, indeed, to act as a formatting 'engine' in subsequent software. The program itself is therefore designed to parse and tokenize an incoming text stream, and apply to the various categories of tokens thus established any rules that might later be applied — it performs no *specific* formatting. The formatting rules are supplied by a file that overlays the base program. This file contains font definitions, initializations of variables and a large number of macros which define the style rules for formatting documents. Such a file is known as a 'format'.* TEX is distributed with a format known as 'plain', and it is, strictly speaking, TEX + plain (or 'plain TEX') that we use as our formatting program.

Plain TEX produces, as suggested by its title, a fairly bland document style, but further macros can be added to specify characteristics such as page size, heading fonts, degree of paragraph indentation and other style requirements. We are currently inviting authors to contribute papers in plain TEX form, because it is relatively easy for us to superimpose our typographical style on top of such a file.

* In some installations, this file (or a modified binary image) is read in at run time; in other cases it is incorporated into the executable binary.

One drawback of this approach is that every heading, say, has to be marked up with specific style commands. A more powerful approach is that of 'structured documentation', where every structural element in a document is marked up or tagged with a code that indicates its logical function, rather than its typography. The typography is assigned once and for all in some style file or other initialization procedure. Another 'format', called LaTeX (Lamport, 1986), is usually distributed with TeX, and allows for a TeX manuscript to be marked up with such structural codes.

We do not currently use LaTeX for the following reasons:

1. The typographical conventions as established in the distributed program are wholly inappropriate for *Acta* papers and other documents we wish to produce. Mechanisms exist for changing these conventions, but it is complicated and time-consuming to do this thoroughly. In any case, the changes we will need to make (in TeX-specific terms) will only become clear after we have undertaken more experimentation with the simpler plain TeX format.
2. Although widespread, LaTeX is not universally available (or used) at TeX sites. Plain TeX is a common denominator which we should always support, even if we choose also to accept LaTeX submissions.

Nevertheless, it is possible that eventual adoption of LaTeX or a format file written in-house and conforming to LaTeX-type conventions will prove fruitful.

Three full articles written in TeX were published in 1990 in *Acta*, Section A (Diamond, 1990a,b; Thomas, 1990). These were supplied by authors with considerable experience of TeX. A considerable amount of editorial time was invested in procedures to format these papers in the journal house style, and the results were judged to be satisfactory.

Some details of the program

We turn now to a brief account of the way in which TeX works. Conceptually, the program assembles a vertical list of rectangular boxes. These boxes may be considered as lines of type, with each typographical character, accent or symbol itself represented as a rectangular box.* The boxes are 'glued' together—their distance from each other is specified as a certain amount plus or minus some tolerance, so the 'glue' is quite flexible. When the length of the vertical list being accumulated exceeds the target length of a page, the vertical list is cut and the contents of a full page are 'shipped out', or written to a file known as the DVI file. The criteria for deciding where to make this cut are complicated: many competing conditions must be satisfied that refer to interline spacing, interword spacing, space above and below paragraphs and around figures and tables. The precise formatting rules are somewhat heuristic: numerical values are given to various figures of merit and a global optimization is attempted of all relevant figures of merit.† Many (in fact, ultimately all) the values of such figures of merit are user-definable; however, there are so many conditions that interconnect that it is wise to make changes from the default values only slowly and with much experimentation.

Complexity is added to the process by the fact that the basic typographic building-block is considered to be the paragraph, so that the interword spaces, placement of characters and insertion of hyphens are done on a paragraph-by-paragraph basis. The result of this is that greater compositional balance can be achieved within a paragraph than is the case with other systems. The disadvantage is that small changes to a word or line can affect the entire paragraph (though this is familiar to practitioners of traditional typesetting), and the optimization process is rather CPU intensive.

While it is formatting text, TeX considers characters and symbols only as rectangular boxes. As mentioned previously, each such box has three independent dimensions (called 'width', 'height' and 'depth'). But each character also has associated with it a number of other dimensions that determine its placement relative to other characters. This set of values (usually about seven for an ordinary alphabetic character, but nearly two dozen for some mathematical symbols) is stored for each character in a TFM (*TeX font metrics*) file. It is the large number of independent placement dimensions that allows for the highest quality in the setting of mathematics.

As mentioned above, the output from the program is a DVI file which contains details of characters employed together with precise instructions for their positioning. The acronym DVI stands for 'device independent', and reflects the fact that TeX has been designed to give identical results on whatever system it is implemented. However, the printing of the output *is* device dependent, and requires two components: a device driver, which translates the instructions in the DVI file to a form understood by the output laser printer, phototypesetter or even dot-matrix printer; and (usually) a set of pixel files which contain bitmap representations of the characters to be printed (recall that to the formatting program these characters appear only as rectangular boxes).

The character fonts shipped with TeX were designed by Knuth himself using the program *METAFONT* (Knuth, 1986) which he wrote for this purpose. Since they were generated originally for his textbook project, they resemble the fonts already used by his printers, and are very distinctive. This historical accident is rather unfortunate, for although the TeX fonts are well designed and not unattractive, they do not match the fonts used in our journals (or indeed those used in most academic book and journal publishing). In consequence, the typesetting of material in the Times font we use is a more complex business with TeX than it might have been. This point will be taken up again below.

* Each such box has three independent dimensions — its width, height above a baseline, and depth below the baseline. Adjacent boxes are arranged with baselines all at the same height. This allows for a proper treatment of letters with descenders—j, p, y *etc*.

† As an example, scores are awarded to normal interword spaces, and demerits are incurred if the normal space must be stretched or shrunk to fit words into a line. However, if consecutive lines end with a hyphen, another numerical penalty is incurred. Thus the interword spacing is adjusted to give a 'best' value; but if the penalties for end-of-line hyphens are high enough, the interword spacing may be changed away from its 'best' value so that the overall score is optimized.

Output devices: PostScript

In the discussion on TEX, emphasis has been placed on the device-independent nature of the formatter. Device dependence enters, however, when the actual printing of the formatted document must be done. Output device drivers for TEX exist for a surprisingly wide variety of printing devices. This means that an author can submit a TEX file with an accompanying printout which he has made on, say, a dot-matrix printer. If we process his TEX file and output it to our laser printer, the line breaks, equation layouts and so forth should look the same (but the overall appearance of the document will be improved owing to the laser printer's higher resolution). In general, however, graphics cannot be reproduced on different types of equipment.

The laser printers we have most interest in contain an interpreter for the page description language PostScript (Adobe Systems Inc., 1985a,b, 1988). This is a stack-based programming language which is designed specifically to relate graphics elements (including alphanumeric characters) on a page; however, it has sufficient arithmetic and logical operations to act as a general-purpose programming tool. One of the windowing systems (NeWS) supported on our SUN workstations is written in an extended version of PostScript.

The great attraction of PostScript is that it gives (essentially) a device-independent description of a page.* Hence a PostScript file printed on our laser printer will produce identical results when printed on a phototypesetter that includes the appropriate software (except that the resolution of the output will be higher). Thus we can fine-tune a document in the office and know exactly how it will appear when typeset — line breaks, hyphenation and so on will all be reproduced precisely.

A word of caution may be in order here. Although PostScript is a full programming language and can be used to format documents,† most applications that claim to support PostScript in fact use it solely to describe the final output. Thus a typical PostScript file, as produced by, say, Microsoft *WORD*, is no more than an electronic representation of a printed page. Although we could easily print the page described by such a file, we cannot in general easily modify it to change the page format. Hence, PostScript itself is not a useful medium for transmitting text unless that text is already formatted precisely in its final form. PostScript files describing line illustrations, however, are potentially useful, for the only editing normally required on artwork is re-sizing, which can (usually) be achieved in a straightforward manner with PostScript.

Two criticisms are generally made of PostScript — that it is slow, and that it generates excessively large files. There is some truth in both these statements, but they require a certain amount of investigation. The size of PostScript files is due to three main factors:

1. Because it is interpreted within the printer, PostScript is sent as 'source code' in ASCII format. A binary representation would be more compact. However, the readable ASCII form does allow editing of files from various sources, and extracting portions of code for use in other applications. This is useful. The point is also made that the names of the PostScript operators are excessively long. This is true enough (**cleartomark**, **countexecstack**, **currenthsbcolor** . . .), but any of them can be assigned any arbitrary user-defined string as needed.
2. A typical PostScript file will have a long prologue section defining and initializing entities that will appear in the page description part of the file. This is a fixed overhead that can make description files for small documents disproportionately large. For instance, one application that we use takes a file containing less than 400 bytes and generates a PostScript file some 14 kB long (35 times bigger). However, the overhead diminishes in relative terms as the file size increases: the same application generates a 200 kB PostScript file from a 99 kB input (factor of 2 increase).
3. The PostScript interpreter is responsible for rasterization of the page in memory (*i.e.* its translation from a conceptual model of line segments, curves and filled spaces to a set of instructions to a laser to print or not print 1/300" × 1/300" pixels). Where this mechanism is followed relatively compact files can be generated. However, where the rasterization is done elsewhere, in the application that generates the PostScript file, the file must include the resulting bitmaps (plus perhaps instructions on how to handle the bitmap). This obviously increases the file size. Consider an extreme example: an instruction to draw an 8" diameter circle on a page could be written in about 60 bytes of PostScript code. A hypothetical scanning procedure which uses one bit to indicate the state of each pixel would require 720 kB to describe the same page! Of course, real applications rarely show so extreme a disparity. However, typographical applications which use fonts that are not resident in the PostScript device incur a large overhead as they need to download character descriptions; and graphics applications that produce raster rather than vector output can also lead to large files.

The other point mentioned above, that PostScript is slow, is related to these considerations. Where a file contains a lengthy prologue re-defining many of the original PostScript operations, or where there are extensive bitmaps, processing is much slower than when only resident PostScript operators and fonts are used.

In any case 'slowness' is a relative concept. Our PostScript applications generally produce publication-quality output from a laser printer faster than our old line printer used to generate barely-legible draft listings!

TEX and PostScript

Both TEX and PostScript are *de facto* standards in the sense that they are implemented identically on a range of hardware platforms and under different operating systems. However, they are programming tools, so that, while their implementation is standardized, their applications are not. In particular, there is no standard way in which TEX handles the resident PostScript fonts that are so useful to us—both because they resemble

* The exception to this is that half-toning and screening functions do require specific knowledge of the printing device.

† This is what we do when producing annual indexes, for instance; a PostScript program decides where to break the text into columns or start a new page.

a a
abcdefghijklmnopqrstuvwxyz 0123456789
ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz 0123456789
ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz 0123456789
ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz 0123456789
ABCDEFGHIJKLMNOPQRSTUVWXYZ

Fig. 5 Comparison between resident PostScript and Computer Modern typefaces. The top line shows a lower-case 'a' in Times and Computer Modern roman 17pt. The Times roman and italic and Computer Modern roman and text italic alphabets and numeral sets are displayed. Note in particular the different weights of the two fonts. The Times font appears to print darker; it is also taller and narrower.

more closely the fonts we already use, and because they give a more compact and resolution-independent output file. Figure 5 illustrates the differences between the two classes of font.

It has already been mentioned that the family of fonts distributed with TEX (known as Computer Modern) are described by a large number of dimensional parameters in order to allow complex and high-precision positioning of characters in mathematical expressions. The use of Times fonts in mathematical setting would require an output driver program that would allow the PostScript fonts to be modified to include these extra dimensions, and perhaps also some modification of the shapes of letters (for example, an italic *j* as used in running text is too deep for use as a subscripted character). Furthermore, characters are referred to by TEX in terms of their numeric position within a font table. Characters are differently arranged in the PostScript and Computer Modern fonts. There are two ways of making the two sets of fonts compatible—rewrite the procedures whereby TEX identifies any desired character to reflect the different location in the PostScript font; or rearrange the position of PostScript characters so that they occupy (as far as is possible) the same positions in the font tables as the Computer Modern characters. This latter procedure can be undertaken by a suitable output driver. We do have at least one output program (known as *dvitps*) which has the potential to allow all these changes, and we have already used it to good effect in constructing caps-and-small-caps fonts. However, a great deal of labour would be involved in making all the necessary changes to the PostScript fonts to allow them to be used in mathematical work. In consequence, we shall for some time follow the convention that Computer Modern characters will appear in mathematics. Aesthetically, this is not an ideal solution. However, it is not at present considered worth the effort that would be involved to do otherwise. This compromise has also been adopted by other publishers and applications programs.

One point that might usefully be made here is that the Computer Modern characters are downloaded to the printer as bitmaps. We have the program *METAFONT* that originally generated these fonts, and this gives us the ability to generate high-resolution (currently 1200 dpi) characters to meet the requirements of phototypesetters or other output devices.

Although *dvitps* is a powerful program, we may find ourselves unable to exercise its full potential if we wish to produce output that is exactly the same as that generated by *The Publisher* (of which more later). Work is in hand to assess the compatibility of the two output drivers.

The Publisher

Several references have already been made to this program, which is supplied by ArborText Inc. of Ann Arbor, Michigan, USA (see Figure 6). This is an integrated document formatter, table and equation editor, vector and raster graphics processor with on-screen previewer. Its 'engine' is TEX, and it is able to import TEX and LATEX documents and preview them. However, it also operates in a mode that allows input text to be 'tagged' by iconic symbols that store formatting information as 'style sheets'. The input window during this mode of operation is illustrated in Figure 7. The style associated with each such symbol can be modified globally, or on each local occurrence of the symbol. Manipulation of these symbols is by mouse and pull-down menu, and the user need never know that he is invoking TEX. Equations can be entered in a separate window which supplies an accurate representation of the result as the user enters material—in other words, this equation editor works in WYSIWYG mode (Figure 8). A similar type of window can be used for entry of tabular data.

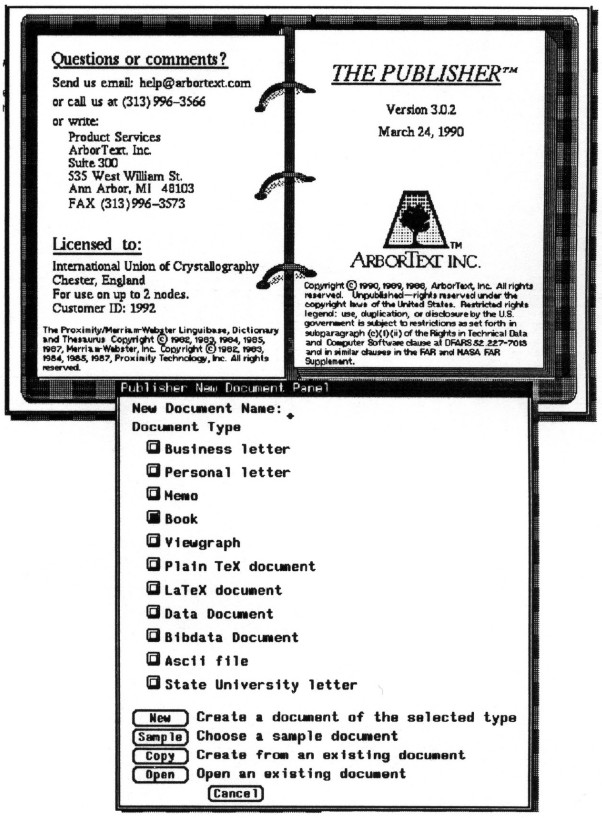

Fig. 6 *The Publisher* startup panel. A window appears offering various styles of document that may be processed. As house styles for various types of papers are encoded in *The Publisher* format, these will be added to the menu.

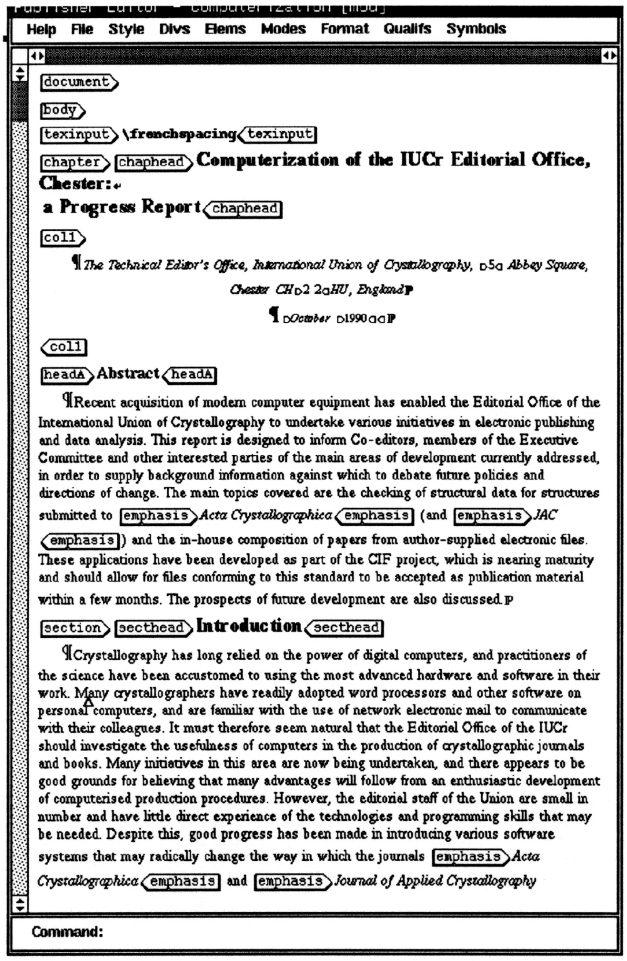

Fig. 7 The appearance of the input (edit) window to *The Publisher*. This indicates how the present document was prepared. Tags for various structural elements may be modified globally or locally. The edit window gives an indication of the appearance of the text as it is input, but a separate Preview option must be invoked to visualize the final page.

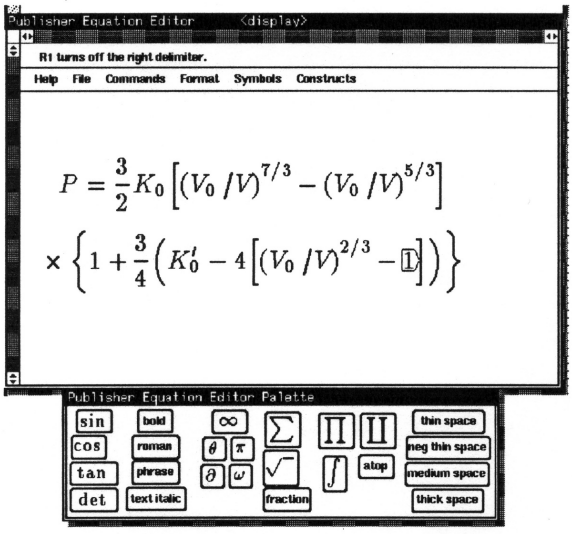

Fig. 8 The equation editor for *The Publisher*. Equations of great complexity may be constructed in this WYSIWYG fashion; but if this does not suffice, native TEX code can also be introduced to obtain extremely high precision in formatting.

We may list the main attractions of this package thus:

1. Its in-built formatting macros are fairly robust and are very well developed. It is notoriously difficult to compose multi-column pages with inserts in TEX, and one of our long-term projects will be to investigate the possibility of writing TEX macros to do this. *The Publisher* achieves multi-column page makeup very well (though not always perfectly). The choice of both global and local styles for document elements is also very useful for fine-tuning page layout.
2. The ability to incorporate external TEX files allows us to make immediate and direct use of author-supplied TEX . While such a file must still be edited as a TEX file (that is, the iconic tags and associated style sheets cannot be applied) equations and tables *can* be composed in the WYSIWYG editors and then inserted into the TEX document.
3. The mechanism of style sheets allows a uniform style to be applied to any document. This will provide a comfortable way of ensuring that all papers conform to a single set of style rules.
4. The package can translate a LATEX file into a form that can be manipulated with tags. That is, a LATEX file can either be read in and stored, still as LATEX source, and consequently edited and previewed; or it can be transformed into a '*Publisher*' document that may be modified in a quasi-WYSIWYG fashion. This will become a very powerful feature in the future, as we develop our own LATEX style rules.
5. Output is in PostScript, and there are efficient mechanisms for handling PostScript graphics. By default, *The Publisher* uses PostScript resident fonts for text and Computer Modern fonts for mathematics—this is the compromise position discussed above.
6. *The Publisher* can also input or output SGML files. This format ('Standard Generalized Markup Language') is a standard developed within the printing community for compatibility between electronic documents (International Standards Organization, 1986). In principle, an SGML file generated in some quite different application could be read into *The Publisher* and manipulated using the local software. We have not yet any practical experience of how well this will work; but it seems to be sensible to conform to accepted standards wherever possible.

We have also a set of filters which claim to translate various word-processor formats to a format compatible with *The Publisher* (in fact, into *WordPerfect* form, which *The Publisher* can import directly). Our experience so far with these has been rather disappointing. It is often the case that such translators are inefficient (the basic models of the document used by different word-processors may be mutually incompatible). We find that text is in general preserved (but text alone can usually be saved in ASCII output files from word-processors). Non-text symbols, mathematics and table alignments are usually not translated, and it is just these features that would be of most use.

Import of word-processing files

It is clear that the ability to submit *via* diskette or e-mail the output from common word-processing packages would be welcomed by many authors. This is a possible

development that we keep under review. At present, our priority is to consolidate the preparation of papers in-house by TEX (as plain TEX or *via The Publisher*), with the consequent ability to accept TEX contributions from authors. When this has been achieved, we shall look in more detail at import of other files. It is likely that the most fruitful approach will be to translate other file formats to a *Publisher* document, partly to provide a single system that will be familiar to all staff, partly because few word-processing packages are sufficiently powerful to meet the style requirements of IUCr journals.

Some comments on this have already been made in the earlier discussion of *The Publisher* package. *The Publisher* itself can import and translate where necessary files generated by TEX and LATEX, *troff*, *WordPerfect* and from SGML applications. We have a program available to translate into LATEX from *Scribe*, and we have a little experience with handling IBM *Script* documents. The *Filtrix* suite of translators converts files in *Interleaf*, *DisplayWrite*, Microsoft *WORD*, *SunWrite*, *WordStar*, *XyWrite* and a few other formats to *WordPerfect* (and thence to *Publisher* format). We find that these translations are reasonably good at preserving text; however, the formatting instructions are often lost altogether, or mis-translated. Although this is of some use (the text of a paper need not be entered manually) the need to introduce formatting commands by hand can cause processing of a document to consume a great deal of editorial time. In any case, the translator may not be able to cope with different versions of word-processing programs. Hence, at present, there appears to be no great advantage to us in receiving a word-processed file as opposed to an ASCII file containing the desired text and a hard copy manuscript.

This conclusion is not presented out of any desire to be negative. We shall continue to review the matter, acquire further translators where possible, and consider (in future) writing our own translators. We shall certainly be happy to experiment with large contributions where the trouble of translation and modification of the input file will clearly offer savings over a more traditional approach. It may also turn out to be the case that, when we have developed detailed styling rules for the journals within *The Publisher*, portions of word-processed text can be cut and pasted into *The Publisher* without the need for further formatting.

V. The timetable for progress

This document has reported at some length on the technical aspects of the computing facilities within the Chester office. Development is proceeding along many fronts almost simultaneously, and the only way to guarantee that this is done productively is to work to a (rough) schedule of priorities. The main thrust of our activities over the past year or so has been to enable structural papers to be handled in the CIF format, and our priority is now to transform the prototype procedures we have been developing for this purpose into a production model. Our (informal) timetable for this is summarized below:

- Extraction from CIF file of data for checking and conversion to suitable formats for input to various checking programs (1 November).
- Conversion of data on a hard-copy *pro forma* into a CIF (1 November).
- Conversion of data in a CIF file into a TEX-encoded file for production of hard-copy *Acta* paper (1 January).
- Assess workload (1 January).
- Finalize design of hard-copy (and electronic) *pro forma* (1 December).
- Finalize format of paper (1 December).
- Arrange TEX training course.
- Develop e-mail handling procedures (1 February).
- Develop housekeeping procedures (1 February).
- Publish in *Acta*:
  - ☐ Editorial
  - ☐ Submission instructions for CIF-type submissions + examples
  - ☐ Full *Notes for Authors*
  - ☐ Deadline and policy of *Acta* B *versus Acta* C.

  (1 March).

This unofficial schedule is, of course, subject to change as the various factors involved interact with each other. Nevertheless, we are currently maintaining the schedule.

Beyond this six-month period, we see the need for the following matters to be addressed:

1. Development of procedures to allow authors to submit their favourite word-processor files.
2. Development of an IUCr LATEX format and efficient handling of author-generated TEX files.
3. Rationalization of the checking procedures to allow routine and correct structures to be handled in a more automatic way.
4. Development of a service providing checked CIFs direct to Databases.
5. Development of a service to other journal publishers involving the structural checking of CIF-based data.
6. Complete overhaul of the 'journal housekeeping' software. This would probably involve investment in a system that is more portable than the current UX-BASIC-based one. It would also be more powerful, and should interact directly wherever possible with CIFs and electronic manuscripts to reduce manual keyboarding.
7. Possible development of a package which we could distribute that generates CIFs, reads and checks them, visualizes structures and provides other means of handling the data contained in these files.
8. Development of techniques for handling half-tone graphics in PostScript.

There is no particular order in the way these possibilities have been listed, and no attempt to suggest a timetable for approaching them. These are matters which will need to be considered by the officers of the Union on a continuing basis.

References

Adobe Systems Inc. (1985a). *POSTSCRIPT Language Reference Manual*. Reading, MA: Addison-Wesley.

Adobe Systems Inc. (1985b). *POSTSCRIPT Language Tutorial and Cookbook*. Reading, MA: Addison-Wesley.

Adobe Systems Inc. (1988). *POSTSCRIPT Language Program Design*. Reading, MA: Addison-Wesley.

ALLEN, F.H., KENNARD, O., MOTHERWELL, W.D.S., TOWN, W.G., WATSON, D.G., SCOTT, T.J. & LARSON, A.C. (1974). *J. Appl. Cryst.* **7**, 73–78.

BURZLAFF, H. & ZIMMERMANN, H. (1985). *Z. Kristallogr.* **170**, 241–246.
CRUZ, F. DA (1987). *KERMIT: A File Transfer Protocol*. Bedford, MA: Digital Press.
DIAMOND, R. (1990a). *Acta Cryst.* A**46**, 423.
DIAMOND, R. (1990b). *Acta Cryst.* A**46**, 425–435.
GABE, E.J., LE PAGE, Y., CHARLAND, J.-P., LEE, F.L. & WHITE, P.S. (1989). *J. Appl. Cryst.* **22**, 384–387.
HALL, S.R. & STEWART, J.M. (1990). Eds. *XTAL3.0 Reference Manual*. Univs. of Western Australia and Maryland.
HIMES, V.L. & MIGHELL, A.D. (1987). *Acta Cryst.* A**43**, 375–384.
International Standards Organization (1986). *Information Processing—Text and Office Systems—Standard Generalized Markup Language (SGML)*. ISO 8879-1986. Geneva: International Standards Organization.
KNUTH, D.E. (1984). *The TEXbook*. Reading, MA: Addison-Wesley.
KNUTH, D.E. (1986). *The METAFONTbook*. Reading, MA: Addison-Wesley.
LAMPORT, L. (1986). *LATEX. A Document Preparation System*. Reading, MA: Addison-Wesley.
LAWTON, S.L. & JACOBSON, R.A. (1965). *The Reduced Cell and its Crystallographic Applications.* USAEC R&D Report IS-1141.
LE PAGE, Y. (1982). *J. Appl. Cryst.* **15**, 255–259.
LE PAGE, Y. (1988). *J. Appl. Cryst.* **21**, 983–984.
MIGHELL, A.D., HUBBARD, C.R. & STALICK, J.K. (1981). *NBS*AIDS80: A FORTRAN Program for Crystallographic Data Evaluation.* NBS Tech. Note 1141.
MUGNOLI, A. (1985). *J. Appl. Cryst.* **18**, 183–184.
NARDELLI, M. (1983). *Comput. Chem*. **7**, 95–98.
SPEK, A.L. (1988). *J. Appl. Cryst.* **21**, 578–579.
THOMAS, D.J. (1990). *Acta Cryst.* A**46**, 321–343.
UGLIENGO, P., BORZANI, Y. & VITERBO, D. (1988). *J. Appl. Cryst.* **21**, 75.