

TECHNIQUES FOR INTERACTIVE
COMPUTER GRAPHICS

A Thesis submitted to the University
of Leicester for the Degree of Doctor
of Philosophy

by

Roger Jeffrey Hubbard

March 1971

Engineering Department
University of Leicester

UMI Number: U380761

All rights reserved

INFORMATION TO ALL USERS

The quality of this reproduction is dependent upon the quality of the copy submitted.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if material had to be removed, a note will indicate the deletion.



UMI U380761

Published by ProQuest LLC 2015. Copyright in the Dissertation held by the Author.
Microform Edition © ProQuest LLC.

All rights reserved. This work is protected against
unauthorized copying under Title 17, United States Code.



ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106-1346

CONTENTS

page

Acknowledgements

Foreword

Chapter 1 Introduction	1
2 Application Software Design for Interactive Graphics	6
3 Graphical Communication	21
4 Data Structure Design	34
5 LUISA - An Application System for Structural Analysis	51
6 An Interactive Three-Dimensional Drawing Program	66
7 BAID - A Program for Basic Architectural Investigation and Design	79
8 Data Generation for Three-Dimensional Finite Element Analysis	85
9 General Discussion	103
10 Conclusions	110

Figures 3.1 to 8.3

Appendix A : Matrix Algebra for the LUISA System

Appendix B : Basic Software and Display
Organisation

References

Acknowledgements

The author wishes to acknowledge the initiative of Professor G.D.S. MacLellan in founding and leading the Computer-Aided Design project in the Engineering Department of Leicester University, supported by the Science Research Council. The author is grateful for financial support from the S.R.C. for the year 1967/68, and to the University of Leicester for appointment to the post of Research Demonstrator from October 1968 to September 1970.

Special thanks must go to Dr. A.G. Young for his supervision, and to members of the C.A.D. group for useful discussions. The author is indebted to Dr. G.A. Butlin, with whom he has collaborated on a number of projects, for much encouragement and guidance.

Foreword

The work described in this thesis was undertaken while the author was working as a member of the Computer Aided Design Group in the Engineering Department of the University of Leicester. The following statement is included to put into perspective his own contributions to the work of the group. The thesis describes a number of projects with which the writer has been closely involved. The general chapters on data structures and interactive graphics contain an outline of these areas and ideas and conclusions derived from work on the projects.

A period of over twelve months was spent on the design and early implementation of the LUISA system described in Chapter 5. This work was undertaken jointly with, and largely under the supervision of Dr. G.A. Butlin, who was responsible for the original concepts of copying and joining of finite elements on which the system is based. The author's own contributions were made to the design of the data structure, algorithms and matrix analysis.

Chapter 6 contains a description of the TDD system for three-dimensional drawing, developed by the author. The approach adopted is different to other three-D systems excepting one developed at Brown University, which only came to light after the completion of the TDD scheme.

The ideas behind the BAID programs, described in Chapter 7, were due to Boyd Auger ARIBA., the writer's contribution being to aid substantially in the translation of these ideas into a data structure and algorithms. The work is presented because it reinforces some of the general conclusions drawn from other sections of the thesis.

Chapter 8 outlines the author's design for a data generation system for structural analysis. A major part of the discussion is devoted to a description of a paging scheme used to overcome the secondary storage problems encountered. The ^h scheme has been implemented and tested on an I.C.L. 4130 computer.

INTRODUCTION

The field of computer graphics is still comparatively young. Early work of about ten years ago was presented for the first time during 1963; work largely the efforts of Ivan Sutherland and Tim Johnson^(21, 22). Sketchpad, and its counterpart for three-dimensional problems Sketchpad III, paved the way for a new form of communication between man and computer. At the same time, also at M.I.T., development of the first time-sharing system was underway and 1963 saw the commissioning of CTSS (Compatible Time-Sharing System). Since that time many developments have taken place, including the design of better hardware and improvements in software techniques.

Present day displays still operate in the same way as the early prototypes, but the last seven years have seen many developments in the software enabling C.R.T.'s to be programmed in higher level languages. A major objection to such display systems has always been their high cost, but now cheaper types of display are becoming available. Of these the direct view storage tube (D.V.S.T.) has great potential. Its major advantage is that no buffer store is needed to hold the picture information: the device operates like a high speed digital plotter and the picture is retained by an electrostatic charge on the scope surface.

The storage tube display is a relatively recent advance and has only been available in Britain for about one year. Consequently much of the expertise in graphics has been gained with the more expensive display systems. These displays rely

on the use of a buffer store to hold the picture information in digital form. A display controller is used to decode this data and to drive the analogue C.R.T. to produce the picture. Often associated with the display are additional pieces of equipment such as the lightpen. Throughout this thesis the emphasis is on this type of display system variously described as a refresh type display, fully interactive display or dynamic display.

Although a great deal of publicity has been devoted by computer manufacturers to the potential of the graphical display as a design aid for the engineer, many industrial organisations remain unconvinced about the economic advantages offered by this equipment for certain engineering purposes. Many of the difficulties encountered in programming displays are brushed aside by software experts as a trivial piece of programming. Without the necessary basic tools the development of application systems can be severely hampered, even to the point of abandonment. The implementation of these basic software tools can only progress once the problems have been identified.

A main aim of the work of the author has been to identify the problems of programming the interactive display, and to find programming tools to solve some of these problems. From the outset it was considered important to investigate the use of the lightpen and dynamic capability of the display. Unless these could be shown to have some advantages the refresh display would rapidly be superseded by the storage tube.

The refresh type display requires some core store to hold the picture information. Interrupts generated by lightpen hits and pressing of function keys must be processed by a computer. These requirements have encouraged the belief that a small computer should be devoted to the display system for performing these tasks.

It is shown that such a configuration can form the basis of a powerful facility, if the small computer is connected as a satellite to a larger time-shared machine[†]. The price of a satellite computer and display system is low enough to be attractive to a large number of industrial groups, but such facilities will only be viable as a design aid if various software problems are solved. Some suggestions are made about the division of labour between the satellite and main computers for highly interactive programs.

Many aspects of graphical communication are a function not only of hardware but also of the training of programmers who implement interactive systems. This training often dictates the mode of communication favoured by the programmer. Many interactive programs use the typewriter keyboard as the means of inputting data, even though a lightpen is available^{††}. The absence of a conveniently positioned teletype with the Leicester display system has led to many devices being developed which use the lightpen and function keys as input. The lightpen can be used in a variety of ways to aid the communication process between man and computer, based on its two major functions of seeing and tracking. Suitably programmed the movements of the lightpen can be used interpretively, so that it can be employed for manipulating three-dimensional models and even stereo views of objects.

[†]Either locally or remotely, but probably the latter for economic reasons.

^{††}Some examples of this can be found at Cambridge University.

The use of pictures as a mode of communicating with the computer has for some time highlighted another major problem area with interactive systems, that of data organisation.

(15)

The techniques of list processing grew out of a study of problems of ordering data describing a frequently changing situation. With large problems, the quantity of information often grows too large to be contained entirely in core store, and methods for information storage and retrieval using secondary storage must be developed. The need exists to link pictures with the data behind them and to express the relationships, topological and others, between elements of the data. These topics have been grouped under the general heading of data structuring. Many of the existing solutions to data structure problems fail to satisfy the requirements of highly interactive systems and raise the usual conflicting demands of generality versus efficiency. These problems can be simplified by accepting that an efficient solution is application oriented, but can only be provided if the techniques and necessary basic software tools can be identified and implemented.

The concepts on which many of the general proposals and conclusions in this thesis are founded originate from experience in designing and implementing several interactive graphics systems. Three of these are described, they are:

- (a) LUISA, an interactive system for structural analysis based on the finite element method,
- (b) TDD, a set of programs for three-dimensional drawing,
- (c) BAID, a system to aid the architect in the design of high density housing layouts.

Another chapter contains an outline of some of the problems and design features of a system which combines the efforts from LUISA and TDD and which is to be implemented in the near future.

APPLICATION SOFTWARE DESIGN FOR INTERACTIVE GRAPHICSIntroduction

In this chapter, two alternative approaches to application software design are discussed. An attempt is made to put these into perspective, taking into account various possible hardware configurations and economic factors. The two approaches are referred to as "topping and tailing" and "modular design", and it is shown that they can be combined in the design of a specific application system.

The design of a C.A.D. system is intricately associated with the mode of communication, which in turn may influence the type of data structure used. These topics are dealt with separately in Chapters 3 and 4 respectively.

Some conclusions are reached about the type of hardware needed for a highly responsive fully interactive computer graphics system.

Non-interactive C.A.D. systems

A non-interactive computer program of the type commonly used in a batch-processing environment does not always offer the designer sufficient flexibility for trying new sets of data. Long turn-around times often associated with this procedure can be a serious obstacle to the effective use of a computer in the design process. The necessity of re-running a whole program when only small changes in data are to be made leads either to many days or weeks being used to investigate the effects of different input data, or to only a few sets of data being tried.

On-line C.A.D. systems using a teletype terminal

Increasing use is now being made of multi-access computing facilities to provide remote job entry from a teletype terminal, from which data may also be specified. This situation has much to offer for on-line program development where programs are held on disc, in source code, and are edited from the remote teletype using an appropriate text editing program.

Several systems are now operating where the terminals are remotely connected to the computer over telephone lines[†]. These facilities offer rapid access to data files, but in large analysis programs involving the output of large quantities of data, it becomes both inconvenient and inefficient to transmit this information to a device which, typically, has a speed of only 10 characters per second.

In such a situation two alternatives exist. The first of these is to output the results at the central site and to post them to the user. Secondly, a line printer can be installed at the remote terminal for rapid data output, and this can be supplemented by a card reader for high speed input.

The line printer is employed to output large quantities of data rapidly. Frequently, such data cannot be quickly assessed and therefore some alternative form of presentation is required in an interactive system.

The graphical display offers such an alternative.

[†]G.E.I.S. and Time Sharing Ltd. offer such services and, more recently, the Ministry of Technology Computer Aided Design Centre has initiated such a system.

The "Topping and Tailing" approach to software design

The processes of data presentation and assimilation can be improved significantly when interactive facilities are added by grafting graphical input/output routines on to each end of an analysis program. To supplement a remote access teletype, the direct view storage tube (D.V.S.T.) would appear to be a very suitable type of display hardware. User interactions could be handled by the teletype with the D.V.S.T. used to monitor program code and data. This approach of adding graphical input and output at each end of an analysis, with little or no change in the analysis program itself, is referred to here as "topping and tailing" and is shown diagrammatically in Figure 3.1.

A "Modular design" approach

An alternative design of software, the modular approach, involves the subdivision of the input, analysis and output phases into a series of modular units. This procedure can be considered at two stages: during the early design stages of software or during the extensive modification of existing programs.

In general, problem areas can be broken down naturally into distinct sections or phases for computer solution. In the first instance these may be input of data, analysis and output of results. For programs to perform structural analysis an initial division of separate tasks might be:

- (1) A description of geometry and properties of the structure.
- (2) Specification of displacements (support constraints).
- (3) Specification of applied loads.
- (4) Analysis.
- (5) Output of results.

If the program is expressed as a block diagram, it is a simple step to divide these modular units into smaller basic operations which can be implemented as individual subprograms. The basis of the "modular approach" is the programming of the smallest unit operations which are then used as building bricks. The result is a group of programs, some of which are atomic, in that they are the smallest units and others are molecular: they too perform unit operations but themselves contain calls to the smaller atomic units. In this context, the definition of atomic rests with the program designer and must be sensible in terms of the application system requirements. If possible programs should be written so that they can be incorporated in a program library where they are available for other systems.

Many computer installations make extensive use of library facilities and this encourages the design of modular software. However, when viewed in relation to the mode of interaction and data structure of a conversational system, the modular approach can be shown to have several other desirable features. The concepts of modular software design arise from early work of the A.E.D. group of project MAC at M.I.T. on language design⁽¹⁾; here they are discussed in the context of application software.

The modular approach is represented diagrammatically in Figure 3.2.

Some discussion of the two approaches

A major limitation of the topping and tailing approach is that there only exists one entry and one exit point to and from the main analysis program. The design loop remains essentially unaltered,

but the processes of data preparation and assessment have been speeded up. This advantage, gained from a graphical mode of communication, makes topping and tailing a worthwhile step, but no opportunity exists for the intermediate assessment of results at various stages of calculation. Some examples exist which show that with certain problems this monitoring and modification of the course of computation is essential if the correct solution is to be found⁽²⁾.

The advantages of a modular design of software can be shown to be numerous. A saving in effort can be effected by avoiding unnecessary duplication of programming statements and by using library routines. Wastage of computer storage associated with such duplication is also minimized.

Another advantage of a modular approach only assumes importance when attention is focused on conversational programs. If the analysis program (or indeed the whole system) has been designed as a series of small units, many entry and exit points can be established allowing interaction between man and computer, without the necessity of including interactive facilities within the modular units themselves (see Figure 3.2). It is then necessary to provide a set of programs which will cope with the desired interactions at the interfaces between the modular units.

The form which the interactions take is dependent on the hardware configuration available. A display with dynamic capability offers more scope for interactive facilities than a storage tube (dynamic picture modification using a tracking symbol, or drawing of "rubber band" lines for example). With only a teletype the alternative means of specifying data are more limited.

Different forms of interaction can be provided by re-writing the interaction handling programs without the need to alter the analysis modules. In order to achieve this, the input/output options for the modules must be clearly defined and must allow for the full range of possible interactions.

Program modules which do not have interaction capability built in can be effectively used by batch programs. If a particular form of input or output is included within a program there is a reduced likelihood of it being a useful library program for a conversational system.

An analogy of the type of program module described is now given. If a module is imagined to be a black box, the various input and output arguments can be represented by strings which pass into the box through a slot. The algorithm (the content of the box) is represented by a mechanism to which the strings are connected. Manipulation of the input strings causes the outputs to change in a way determined by the mechanism.

Different sets of strings exist for various forms of interaction (interaction implies pulling of the strings) and these may be connected to different parts of the mechanism. In order to construct a system from a given set of boxes the appropriate connections must be made between the strings. In a batch program the connections are direct, in an interactive system certain outputs may be monitored by the user who then determines suitable inputs for the next stage. The purpose of the interaction handling programs is to enable the user to form the necessary connections, to change them during the running of the program, and in some cases to pull the strings (i.e. feed in values). To do this he

need not be aware of the contents of the boxes, only their functions, inputs and outputs which are meaningful to him. Those connections which are not meaningful to the user, but which need to be made for administrative purposes should be formed automatically, where possible.

The task of the application programmer can now be defined, if the necessary interaction handling facilities are available. During the process of designing a system, a series of boxes can be drawn representing the functions of the modular programs, with lines representing the input and output facilities, suitably labelled. Boxes can then be compared to ensure that the necessary values can be passed between modules for any likely sequences of operation. If, in practice, the user attempts actions which the system does not logically permit, and this is due to a mis-match of arguments, the interaction package should be capable of detecting the error. If the error is problem-oriented rather than a system fault, the error detection facilities must be internally programmed in the modules.

Having divided a program into a series of modules, these must be linked together to form a comprehensive problem-solving system. Two means exist for doing this. The argument lists of programs can be used to pass values to the next phase. The major link, however, lies in the design of a common data base or data structure for the given application area on which the modular units perform operations, and it is this which enables a significant flexibility in the order in which these operations can be performed. This imposes a restriction on the use of such modules as library items for other systems, unless a common data structure exists, or unless the module in question is independent of the problem data structure.

From Figure 3.2 it can be seen that many alternative routes may be chosen to achieve the desired end. The nature of the problem will almost certainly impose some constraints on this choice, but in general these will be detectable from the state of the data structure (because the data structure is a model of the physical problem) and the necessary checks may be incorporated in the program modules. If a mistake is made, or a wrong decision taken, it may be possible to detect this quickly and perform any necessary alterations without major recalculation. With the topping and tailing approach a mistake may only become apparent when considerable cost has already been incurred. Proceeding in the small steps, characteristic of the modular approach, gives a higher probability of obtaining a suitable solution efficiently because of the opportunities for intermediate assessments and modifications.

The concept of advancing in small steps is attractive from a practical viewpoint for several reasons. Graphics programs very often require considerable core store even to perform relatively simple operations. Dividing the problem solving process into discrete small steps enables efficient use of core if a suitable program overlay facility exists. If the program modules are too large, unacceptably long response times are likely to result. In practice there is often the need to optimize and compromise to obtain a balance between size and number of modules in order not to over-complicate administrative tasks. (Some operating systems impose limits on the number of subroutines in a given program suite[†].)

[†]The ICL 4130 Fortran system allows only 100 Fortran subroutines.

The points discussed so far have represented an idealistic viewpoint, but in many respects are consistent with the real procedure of application system design. The process of design is often presented as a loop in which modifications are made many times before an acceptable solution is obtained for a given problem. This is often the case with software design, and it is interesting to observe that the same rules apply to programming as to the design processes in engineering problems. In particular, part of the function of design is the discovery of what is acceptable or possible. In practice, therefore, the design of software involves many modifications and major changes can be made more easily if the programs are in modular form.

Three types of system development can be identified. Firstly, there are systems whose limits have been clearly defined and which are implemented and then "frozen". Secondly there are exploratory systems (usually the work of universities) the aims of which are to discover the limitations of hardware and basic software design, and to investigate new techniques (as in Chapter 6). Finally, there are many systems which are designed to be open-ended. These systems often lead to insight into what is really needed by way of hardware and software for such development (as in Chapter 5).

These second two types of system are candidates for modular software design, but defining and using routines as building bricks poses the following problem. Because of the dependence of other programs on them, the basic units must be efficient and free from errors. As modifications are made to the system, changes to some of these atomic modules will inevitably occur and perhaps this will lead to changing of all the calls to the routines in question.

For such a process, a context editing program is essential: a modest system developed in a modular fashion can easily contain a hundred subprograms, with many calls to several of them.

Achieving a balance between the two approaches

In spite of its many advantages, the modular approach to software design does not immediately lend itself to the use of existing analysis routines. Because of the high cost of software design and implementation it is desirable to make the greatest possible use of existing programs. Many analysis routines will benefit from the addition of graphical input/output using a topping and tailing approach. In particular in the field of structural analysis, the preparation and checking of the data prior to the calculation phase can be invaluable. The preparation of data for a finite element analysis requires a data generation scheme such as that described in Chapter 5. With some classes of finite element problem, the generation of suitable meshes can be performed automatically⁽³⁾, but this is difficult to program for complex situations where a high degree of control over the input is required if the results are to be satisfactory. In particular, in finite element problems, the mesh shape is important: the occurrence of poorly shaped elements will lead to ill-conditioned matrices during the analysis, giving rise to inaccurate results.

The use of existing analysis routines can become unsatisfactory with highly interactive systems if the resulting response times become large, while calculations are performed. This is also true of data input and output using displays: the response time must be of the order of a few seconds at most. If a delay of more

than two seconds is to be expected, some instant sign must be given to the user so that he is aware that the computer is at least processing his request.

If existing analysis programs have been tested and proved satisfactory there is little point in not making use of them. Returning to the topping and tailing approach, the contribution of a graphical display in this situation is to provide the input and output, which should lead to an improvement in efficiency of the overall system. Because of the ~~need for~~ ^{desirability of} fast response these input and output routines can be very conveniently designed in modular form. The resulting flexibility in the way in which data is input and examined is then well suited to the different preferences of a number of users. General packages of programs can be used to aid the writing of input and output routines. Such packages include programs for the administration of messages and lightbuttons,[†] and the filing of picture items and their data structure on backing store.

Data prepared in this manner would be filed on disc, magnetic tape or drum to be accessed by a batch program which performs the analysis. Results would be similarly filed ready for examination using the display for output. This process is shown in Figure 3.3.

Application system design for a remote satellite computer

Often, complex analysis programs are of such a size as to require powerful (and expensive) computing facilities. An interactive display, whose picture is refreshed several times per

[†]Programs of this nature have been written at Leicester University.

second does not warrant the use of a large computer system to perform simple picture manipulation tasks. If a large machine is used it is important to operate in a time sharing mode in which priority is assigned to the display system, with other jobs running in the background[†]. This situation has led to an increasing use of satellite computers whose part function is to control the picture handling. Lightpen tracking and display file manipulation are also performed locally.

Figure 3.3 shows a typical division of labour between main machine and satellite. The analysis routines and their data are held in the main computer, with the satellite used as little more than a powerful display processor. Such a division implies that the satellite computer is only a convenient means for gaining access to the large machine without loading it with the overheads of servicing the display. If a satellite is to be used, there is much to be said for extending its power to perform ^{some} calculations which would otherwise be performed centrally. The arguments in favour of this relate to a remote environment with a relatively

[†]The display user needs to have priority over other jobs in order to obtain good response times. If more than one display is attached to a multi-access computer, the assignment of priorities cannot be clearly established without prejudicing one or other of the users. If a powerful and fast machine is used to overcome these problems it is not economically sensible to load the machine with routine simple tasks such as picture manipulation.

slow telephone connection between computers[†].

A change made to a displayed picture often implies some modification of the problem in hand and the data associated with it. The time to update application data held in the central computer will be unacceptably long for most interactive systems. If, however, the necessary data were stored in the satellite the response times would improve considerably. It is possible to arrange this by designing a data structure for the input/output operations to be performed using the satellite. The programs for this input/output can be organised on a modular basis, the aim being to create a file of data for an analysis program. The analysis routines are held in the main computer and the prepared data file is transmitted and filed on disc or drum at the central site. A request is also sent for the analysis program to be run as a batch job and for the output to be filed at the central site. At some convenient time, the output data file is transmitted to the satellite where the information is put in a structured form suitable for interactive interrogation. This scheme of operation is shown in Figure 3.4 and is designed to make economic use of both the satellite and main computers.

During the period when calculations are being performed in a batch run at the central site, the user can be employed in using the satellite for other purposes: either program development or preparation of data for other problems. A realistic size of

[†]Typically 2400 baud. Fast lines are not readily available at the present time and indeed are not necessary for the scheme outlined. Theoretical line speeds can be lowered by frequent errors and the need to check for them.

satellite for such a scheme is about 32K words of store and a disc of 1 million word capacity[†]. The operating system software could be expected to occupy up to 12K of store. A further 5K will be needed for program code at any one time assuming a segmentation facility is available to operate with the disc. The remaining 15K is a suitable size for holding data in structured form for fairly large problems. The disc is needed for the segmentation of program code and for holding library routines, operating system and compilers. In addition, it can be expected that a number of data files will be held on disc.

Conclusions

A highly interactive graphics system, which will be sufficiently fast to allow good response times, requires some dedicated core store. If the programs are organised carefully this can be achieved using a medium speed machine (2μsec core). It does not appear sensible to have a very powerful machine, since some of the operations to be carried out do not justify such expense. For these reasons an effective and economic solution is to organise such a system around the use of a small, medium speed computer whose core store is devoted to the display system. If the total software design, including application programs, general packages and the operating system, is suitably arranged, large problems can be tackled by linking the graphics computer to a large multi-access machine. In order to utilise the satellite machine most effectively, some of the problem data should be

[†]Preferably with interchangeable disc packs.

stored locally and this can best be arranged if local fast backing store is available (disc or drum).

If the problem is small enough, it can be tackled locally if the satellite has a good operating system with program segmentation facilities. The presence of such facilities is an encouragement for development of modular software and of packages for interaction handling. A modular design is particularly well suited to extension by adding further units.

In order to include some of the features described, the size of computer needed as a satellite is about 32K words of core store, with some fast backing store. Smaller machines would be adequate for directly linked machines where high transfer rates are possible. For remote graphics using telephone lines the suggested size is more realistic unless response times are to suffer when access is required to data which has to be held in the main machine if the satellite is too small.

With less flexible graphics systems such as the storage tube/teletype console little or no local computing power is necessary since picture manipulation of the type employed with a refresh type display would not be attempted. Such a system could well be organised by connecting it directly to a multi-access computer. The discrete step form of interrogation offered by the teletype can be adequately serviced by a time sharing system without any assignment of priorities.

GRAPHICAL COMMUNICATIONIntroduction

In this chapter several aspects of man/computer communication are discussed. The emphasis is mainly on the use of fully interactive displays whose picture definition is held in a display file. An outline is given of some aspects and modes of communication which require a display whose picture may be modified dynamically. Discussion is devoted to the organisation of interactions and the way in which qualitative as well as quantitative requests may be handled by using a variety of software devices.

At the end of the chapter an outline is given of some of the problems encountered with visual representation and some of the methods available for tackling them.

Forms of communication

Graphical man/computer communication is based on two forms of representation, pictorial and symbolic. Here pictorial is taken to cover the representation of objects by images, suggestive of the external form or shape of those objects and therefore in some way related to their actual shape. Symbolic representation covers the use of alpha-numeric characters and other shapes whose meaning is in some sense pre-defined. A symbol is used as a representation not by exact resemblance, but by suggestion or, in this case, by convention.

The traditional means of communicating with the computer via punched programs and data is symbolic, i.e. in terms of alpha-numeric characters. A conversational program operated from a teletype relies on symbolic communication, and in graphics programs, some symbolic representation is always present. The link between pictures and the user is often provided by type-written commands, physical buttons, or lightbuttons, which are a symbolic display of various options within the program. Other examples of symbols are messages, dimensions on a drawing and numbering schemes or names of components. The essential feature of a symbol is that it has a pre-defined meaning. With a picture, however, it is the shape which is significant and the aim is often to change the shape until it satisfies various constraints. The picture is not pre-defined therefore, in that it may be modified at will. Once a definition has been assigned, the picture may become a symbol as with a sub-picture, such as a transistor in electronic circuit design.

With a storage tube display the graphical representation of a problem is limited to output only, and input is usually via a series of commands typed by the user at an on-line teletype console. Some users of displays equipped with a lightpen have abandoned the use of lightpen facilities in favour of a typewriter mode of communication. A common reason for this is that the computer operating system in use does not allow for the most effective use of the equipment. This is usually because of the need to service other users in a time-sharing environment. The teletype is a ~~discrete~~ ^{discrete} step form of input and is therefore well suited to some time-sharing systems where the graphics user receives processor attention in a series of time slices.

Aspects of communication requiring a dynamic display

The lightpen only assumes superiority over the typewriter when it is used to communicate in a manner not possible with typed commands. Techniques for achieving this can be grouped generally under the heading of "wand waving", or using the lightpen interpretively. By this is meant using the lightpen to vary an input to some calculation whose output is a dynamic change in the picture. Such dynamic effects cannot be produced using the direct view storage tube because of the need to "repaint" the whole picture[†]. With the dynamic display whose picture is refreshed several times per second it becomes possible to update a part of the picture by editing and modifying the display file.

It is possible to point to several examples where dynamic picture output is of importance. Four areas of such use are: fluid flow visualization^{††}, design of mechanisms^{††}, simulation problems including traffic flow^{††} and aircraft flight⁽⁴⁾, and dynamic analysis of structures (e.g. vibration analysis of aircraft structures).

In order to examine how the fully interactive display can be used to improve the process of man/machine communication, some attention must be paid to the hardware features of the equipment and in particular to the use of the lightpen.

[†]The storage tube is in many ways like a very fast digital plotter.

^{††}Projects involving these topics have been conducted on the graphical display at Leicester University.

The lightpen has two major functions: to indicate picture elements and to perform pen-tracking[†]. Pen tracking is often used in drawing programs where the tracking cross is used to specify the end positions of lines. It can similarly be used to position complete parts of a picture, usually with the aid of some algorithm if very accurate positioning is required, once the approximate coordinates have been input with the tracking cross.

Many makes of storage tube are now being offered with a cursor^{††} which may be positioned on the screen by means of a hardware joystick^{†††}. This enables coordinates to be specified in much the same way as pen-tracking, but is of limited use with a display used primarily for static images. The ability to adjust the position of an object dynamically using pen tracking represents a major advantage of the refreshable display.

[†]Pen-tracking offers a means of making available to a program the position of the lightpen on the CRT screen. This is achieved by displaying a small symbol (usually a cross) which moves so as to stay in the centre of the field of view of the lightpen. As the lightpen is moved across the surface of the CRT, the tracking symbol moves with it and its coordinates are continuously held in a pair of registers which may be examined by the user program.

^{††}DEC, ADAGE

^{†††}Some dynamic displays use a rolling ball (tracker ball) to position the tracking symbol (Marconi X2000).

Broadly speaking, such facilities enable the user to "prod or poke" his representation of a physical system, as though with his finger, in order to gain an appreciation of the dependence of the system response on various parameters. In Chapter 5 a description is given of a system which allows this type of examination with civil and mechanical engineering structures. It has often been stated that a major use of display systems should be to gain insight into the behaviour of physical systems. Often the display is used as a convenient means of entering and examining data, without any attempt to create a loop in the process, whereby the response can be directly associated with some variation of an input parameter. In order to indicate how this might be achieved some general concepts about the nature of communication in an interactive program must be identified.

The organisation of man/machine communication

Interactions involve the setting up of a command which will cause the computer to take some appropriate course of computation, and perhaps to indicate this through the medium of a displayed picture. A command, either implicitly or explicitly, has two components: an ^{operator}~~operation~~ and an operand. A further two components may be present whose functions are to specify the values of variables used in executing the command, and to indicate where these values are to be found.

A simple command can be used to change the scale of a picture part. The information which has to be specified is:

- (1) the operation "change scale",
- (2) the name of the picture item whose scale is to be changed,

(3) the new scale value,

(4) where the value is to be obtained (e.g. typewriter).

Sections (2), (3) and (4) could be implicitly defined in the command "change scale". For example, perhaps the whole picture is to be changed to a new scale of half the previous value.

General programs usually contain very little implicit information.

Choosing an operation such as "change scale" amounts to identifying a subroutine, or group of routines, to perform the desired action. The additional information which must be specified can be passed to the subroutine via its argument list. Although communication with the computer may be through the medium of displayed pictures it must be remembered that it is the digital model which is modified when a change is made. In order to achieve this, without the user being aware of it, a relationship must exist between the picture elements and the data describing the physical situation which they represent.

Indication of a picture element with the lightpen should be used to identify the part of the digital model to be modified by a subroutine. This can be achieved by assigning a name to the picture part whose value is a pointer to the relevant part of the data structure[†]. This value is passed to the subroutine as an argument.

The order of events can be summarized:

- (1) Indicate the area of interest using the picture,
- (2) Use this information to identify the part of the digital model (data structure) which the chosen picture element represents,
- (3) Modify the digital model as desired and then update the picture to portray the new situation.

[†]Pointers and data structure are discussed in Chapter 4.

It is important that the operation specified by a command should lead to a change in the digital model, from which the picture is displayed. This ensures that the displayed picture portrays the state of the application data.

The algorithm to perform (3) corresponds to the chosen operation (e.g. change scale). It can be selected from a menu of lightbuttons or typed on a teletype. Other information will be obtained from different sources, for example typed values, from lightpen hits, as output arguments from other subroutines. General programs can be written to cope with the setting up of commands using such inputs[†]. A package of Fortran subroutines has been written at Leicester University to perform this task⁽²⁰⁾, and is used by the applications described in Chapters 5 and 6.

Qualitative Assessment and man/machine communication

In order to create a situation where qualitative as well as quantitative evaluation of a problem can be made, a loop must be formed within the program. In many processes where parameters are to be adjusted until some criterion (perhaps visual) is satisfied, it is often more convenient to change a value to be "a bit bigger or smaller" rather than to have to be concerned with its magnitude. Often the magnitude is of significance only after the process of adjustment is completed. This is particularly true when judgement is on a visual basis, taking advantage of man's superiority over the computer for visual judgement. This assumes that it is possible to display an unambiguous representation

[†]See Chapter 2. Sometimes such facilities are provided in the form of Command Definition Languages.

of the situation[†].

The use of "devices"

If a continuous loop is not created for the process of adjustment, where a command is continuously executed with parameters modified in value for each execution, the assessment becomes a repeated "single shot" operation which can be very laborious. If the value of the parameter to be modified is obtained from a typewriter execution of the loop becomes discontinuous while the value is typed. To overcome this problem various devices are used which can be operated with the lightpen and function keys of a fully interactive display. In particular these devices rely on the use of pen-tracking to specify some coordinate value which may be transformed and used as the magnitude of a parameter. Where convenient the devices are programmed to correspond as nearly as possible to the type of parameter modification to be performed.

If an angle is to be specified, a line can be displayed to represent a joystick and the angle between this line and some reference direction used as the parameter value. If the device is then used in a loop, a picture part may be caused to rotate in sympathy with movement of the joystick, whose free end is positioned with the lightpen.

By the use of such techniques a relationship may be established between the user's actions and the effects they produce on the physical system under examination.

[†]See later this Chapter for discussion of some visualization problems.

An example of another device is the pseudo tracking-cross mentioned in Chapter 6. The pseudo tracking-cross moves in three dimensions in various planes in space, enabling the two-dimensional coordinates of the real tracking cross to be transformed, to control three degrees of freedom. The transformation is defined by the plane in which the pseudo tracking-cross is currently constrained to move.

Another device of a similar nature also uses the ordinary tracking-cross coordinates and transformations of a suitable nature. The "spider", as the device has been called, is effectively a pseudo tracking-cross which is constrained to move on a mathematically defined surface such as a cylinder. As the ordinary tracking-cross is moved around the screen its movements are interpreted by a transformation routine fixing the position of the "spider" on the surface. This is effectively a problem of mapping the screen coordinate system on to some other mathematically defined surface. Such a device can also be used if stereo pairs are displayed, and it overcomes the problem of how to use one tracking-cross with two images.

Techniques of communication involving dynamic model manipulation are particularly valuable for gaining insight into problems where interest is focused on variation of design parameters rather than assigning absolute values to them. By including the source of information as an argument in the routines, various devices may be used including the typewriter, if a particular parameter value is of interest. Devices such as the pseudo tracking-cross enable parameters to be varied in a way which it is difficult to describe using commands. Requests such as "move this point a little further to the left" cannot

always be easily expressed numerically. The burden of typing numerical values should not be unnecessarily added to the tasks performed by the user, unless he is interested in specific numerical values.

Visualization problems in pictorial representation

Although pictorial presentation of a problem offers scope for simple schemes for assimilation of large quantities of data several major problems exist. These problems are usually grouped under the heading of "visualization", and relate mainly to difficulties of using a flat scope surface to represent three-dimensional, real objects. Solutions to these problems are expensive and acceptable solutions are not widely available. The use of orthographic projections in engineering drawings is an example of a convention to avoid certain visualization problems. The aim is to make the drawings unambiguous, but they do require interpretation and it is here that errors can occur. In addition they are not suitable for many areas of design such as surface definition.

A major obstacle to three-dimensional display is the hidden line problem. In the case of raster displays, as opposed to vector displays, the problem is one of hidden surfaces. The use of software operating in processors of the present generation, to compute the solution to the hidden line problem, is generally too slow for fully interactive systems. A small addition to the displayed scene may necessitate several seconds, or minutes, of processing time to update the picture and dynamic rotation of the image becomes impracticable for the same reason.

Much effort has been devoted to software development for solving the hidden line problem⁽⁸⁾. With most of the available algorithms the computation time rises in proportion to the square of the number of objects in the scene. Warnock's algorithm⁽⁸⁾ requires a time proportional to the number of objects and is therefore more suited to complex pictures which have several component parts. The most effective economic use of such algorithms would seem to be in the production of hard copy plots, where the plotted view has the hidden lines removed.

Another problem with the CRT is to look at a part of a picture and to compute which part of the whole scene is to be displayed. This is known as windowing and can be solved in two ways. One is to provide the scope with a beam deflection system which will cover an area larger than the viewing area so that the picture is cut off at the physical edge of the viewing area. This however involves "drawing" more than the picture actually observed by the user and wastes time. The second solution is to clip the picture at the edge of the viewing area by computation. The picture drawn then corresponds to those items which lie inside the window. Calculations must be performed to determine the points at which the picture crosses the window bounds, and this is useful because it is not then necessary for the window to correspond to the scope viewing area.

If the technique of clipping is extended to three dimensions, it is possible to take slices through a three-dimensional model and to eliminate unwanted detail lying in front of, or behind,

the zone of interest.

A different technique, brightness modulation[†], or depth cueing, is a particularly valuable aid to visualization and is relatively inexpensive. The technique is to vary the brightness of a line depending on its depth coordinate normal to the display screen such that points near to the user are brighter than those which are distant from him. The displayed picture is two-dimensional but the illusion of depth is created by the brightness variation. This technique requires suitable hardware facilities in the form of several ~~discrete~~^{discrete} brightness levels (about seven) or a continuously variable brightness (available on the ADAGE ~~storage~~ ~~the~~ display). If discreet brightness levels are available, the depth-cueing can be provided by a general purpose 3-D clipping program which also offers 2-D windowing.

Perspective views can be of use in some problems, particularly where displayed objects have a number of parallel edges. They are effective too in problems such as highway design⁽²⁹⁾. Stereoscopic views can be computed but can create some problems of communication: the need to look through a viewer makes it difficult to operate the typewriter and lightpen.

Interesting work is in progress at the University of Utah on problems of communication^(5,6,7), which hopefully will bring about an improvement in display processor design. Until that time, ingenious techniques for the display and manipulation of graphical images must be devised to overcome visualization problems.

[†]Used by the Cambridge University C.A.D. group.

Conclusions

Apart from problems which require a dynamic display for output, the use of dynamic picture modification can be a significant aid to the user both for input of data and in aiding visualization.

The use of the lightpen with suitably programmed devices offers a flexibility of input not attainable with the typewriter and storage tube display. There is particular scope in the handling of three-dimensional models for devices of the type mentioned, and the need to specify precise values for parameters is not always necessary. Requests such as "Make X a bit bigger" or adjusting the value of a parameter X until some function $F(X)$ has a particular value can be dealt with.

Many problems can be tackled without recourse to such techniques and a storage tube and teletype may be adequate in these cases. A major advantage of the dynamic display is the ability to adjust parameters until some desired criterion is satisfied. If the problem to be tackled is not well understood (i.e. the effects of variation of parameter values) dynamic techniques may prove particularly useful.

The most economic solutions to visualization problems associated with the hidden line problem appear to be 3-D clipping and brightness modulation. Some manufacturers of displays are now offering brightness modulation performed by hardware, and 3-D clipping enables unwanted parts of the picture to be blanked out completely. Hidden line removal is probably best reserved for the production of hard copy plots where computation times are less critical than with on-line design situations.

DATA STRUCTURE DESIGNIntroduction

At the beginning of this chapter an outline is given of why data structures are needed. This is followed by an example for which various data structures are presented. The first data storage scheme is a simple one using arrays to store information, and it is shown that several problems arise when the quantities of information are constantly changing as is usually the case with an interactive program.

A description is given of the general principles of free-storage schemes and the use of beads (records) and pointers. Then follow descriptions of various data structures, for the example problem, which use a free storage scheme. The designs presented become more complex towards the end of the chapter and their advantages and limitations are put forward.

Finally some discussion is devoted to the storing of structured data on backing store, along with some criticisms of paging schemes. It is shown that an efficient solution to these problems depends on the particular application.

What is data structuring and why is it necessary?

The computer-aided solution of a problem involves the design and implementation of a digital model. This is a task usually undertaken by an applications programmer who makes use of the basic software provided by systems programmers. D.T. Ross⁽¹⁾

defined the "model" by an equation:

$$\text{model} = \text{data} + \text{structure} + \text{algorithm}.$$

The equation is a particularly useful one to consider, in that it expresses the balance which must be obtained between the component parts of the model for various degrees of sophistication of the application program. This sophistication is to some extent dependent on the hardware and the basic software available. Many requirements arising from the application must be satisfied, amounting to extra features to be built into the model.

In the above equation the data are the values pertaining to the physical problem to be modelled, such as lengths, areas, volumes, Young's modulus. The algorithm is used to perform operations on these data, to calculate other values from them.

Consider the line shown in Figure 4.1. The data needed to describe the line could be stored as:

- (1) the coordinates (X1, Y1), (X2, Y2) of P1 and P2,
- (2) the coordinates of P1, the length L and angle A,
- (3) the coordinates of P2, the length L and angle A.

The data in each case can be obtained from either of the alternatives. However, it may be useful to store all of this data if some calculations are to be performed which require each of the alternatives at some time. This is equivalent to stating that an increase in the data term in the equation can lead to a reduction in complexity of algorithm. This has in fact been done in the case of the drawing program described in Chapter 6, where three sets of coordinates are stored for each point in space.

Consider an example where there are several lines between a set of points. If the lines are to be defined in terms of the points, it is necessary to determine which pair of points belong to a given line. This can be done by storing a reference table containing extra data about the relationships, but the complexity of the algorithms increases to include a search of this additional information before operations can be performed on the original data. Data structuring (the structure term in Ross's equation) is concerned with finding the best way of storing the data to simplify this search problem for a given application, subject to the many other constraints to be satisfied. A typical constraint is the amount of core store available and sophisticated data structures very often fill up core space at an alarming rate.

Data structure design for a particular problem

The problem of designing a particular data structure will now be discussed to illustrate the many alternative solutions, and to serve as a basis for an examination of their relative merits.

The example chosen concerns the storage of information about a set of triangles which are to be displayed on an interactive graphics console. The basic data to be stored are the coordinates of the nodes of the triangles. Although a simple problem, the example has been chosen because of its similarity to the finite element system described in Chapter 5. It will be assumed that the operations to be performed by the user are copying, deletion and joining together of triangles to form larger shapes with some common boundaries and nodes.

The triangles to be considered are typically like that shown in Figure 4.2.

It is assumed that the boundaries of the triangles will be displayed and that associated with each boundary is a unique name or item number for the purposes of the display system. These item numbers are used to gain access to other data in order to perform copying, joining and deletion.

Typically the information (both basic and associative data) can be held in two-dimensional arrays. Two such arrays might be used, as in Figure 4.3. The BOUNDARY ARRAY has one row assigned to each boundary (boundaries are numbered consecutively) containing the reference numbers of the points defining the boundary, and of the triangle to which it belongs, and the display file item number for the boundary. The COORDINATES ARRAY contains the coordinates of the points.

Such a data storage scheme is extremely simple in concept but presents problems of implementation. A particular difficulty arises if information is to be deleted, as certain rows of the arrays will no longer contain useful data. It is possible to re-order the remaining data in order to remove the redundant spaces, but this may require considerable relocation of values and must be programmed for each array which is to be treated in this manner. Alternatively, the algorithm may be designed to process the whole array including any redundant information but this is a waste of both computing time and space occupied by data no longer required.

A major objection to a simple array storage scheme is the need to specify the amount of space required at some point before

it may be known. Even a block-structured language, such as Algol, with dynamic array allocation is not adequate for very interactive work since each separate array has to be declared, although the array sizes can be allocated at run-time. The problem of re-ordering of data is still present so the total workspace available in core is unlikely to be used in the most efficient manner for a given application.

The requirements for interactive work which an array storage scheme fail to satisfy would appear to be:

- (1) How to allocate space as it is needed during the running of the program,
- (2) How to avoid the problem of re-ordering data when operations like delete are performed.

From investigations of these types of problem came the techniques of list processing using free storage schemes with "garbage collection"⁽¹⁵⁾.

Free storage schemes and garbage collection

Free storage schemes usually operate on the basis of assigning a workspace or free storage zone which will contain the information otherwise held in arrays. In some implementations the free storage zone is itself an array, some parts of which contain the data, and the remainder is considered free. Somewhere in each free area (often the first word) the address of the start of the next free block is stored. Addresses used in this way are termed POINTERS. The free areas are therefore chained together by a series of pointers. The length of each free block is also stored. Information is held in records, hereafter

referred to as BEADS using AED terminology⁽¹⁾. A bead is a block of contiguous computer words whose size is specified by the programmer and which may be allocated during the running of the program. When a program requests the allocation of a bead, perhaps as a result of a user interaction, the free storage routines determine whether the requested space is available. If it is, the required block is removed from the list of free-store beads and is assigned to the program for the storing of data. In order to know where in store the new bead has been allocated, the free storage routine returns the address of the start of the bead. In the case of schemes which use an array as the free storage zone, this address is the array subscript and is therefore relative to the head of the array. This is a useful feature which will be discussed later in the context of the use of mass storage.

When a bead is made available by the deletion of the information it holds, it is added to the list of free blocks. If two such blocks are contiguous they are merged to form one larger block by suitable updating of the pointers.

A simple chain structure for the example problem

In the chosen example the BOUNDARY and COORDINATES arrays can be split into separate rows, which can then be imagined to be beads which are linked together by pointers as shown in Figure 4.4. Previously the boundaries and points were numbered consecutively, now they are not: their reference numbers are the pointers to their beads and are stored in the previous bead

thus forming a chain or list structure. The lists are cross-linked by pointers from the boundaries to the end points which define them.

An algorithm which is designed to process each bead in a list can therefore be written as: get the next bead (if it exists) and update the information it contains. By following the lists in this manner, no problems arise about processing redundant information and the same beads can be placed on different lists depending on the operations to be performed (as in the program described in Chapter 7).

If a bead is to be removed the bead occurring before it in a list must be located and modified to point to the next but one bead. The deleted bead is then returned to the free-store list. Since the processing algorithm is written to follow the lists the redundant information will not be processed because it has been by-passed in the list.

Tree structures

A typical aim in application programming is to provide a general problem solving system for the given application area. In line with this, several attempts have been made to provide general data structures for interactive programming^(10,11,12), but the overheads incurred do not make this an attractive proposition. A typical general structure is based on a tree arrangement^(13,14), and very often, for simplicity, mixed pointers and data are not allowed in the same bead. Such a restriction enables the necessary general structure building commands and pointer manipulation routines to be easily defined, but implies that data are always

held at the base level. This is not in keeping with the usually hierarchical nature of the problem to be solved, and data should ideally be stored at different levels to reflect the physical system which the model represents. This may have a significant effect on the time to access certain of the data.

A critical examination of a tree structure reveals several disadvantages but also some merits. Figure 4.5 shows a tree structure for the triangle problem.

The major advantage of the tree structure is its simplicity resulting in straightforward programs for structure building and manipulation. A considerable problem encountered, when the information is held in structured form, is that of transferring it to mass storage, because of the need to change the pointer values when the information is read back into core. One way of achieving this is to un-structure the information prior to storing on disc or drum and to re-structure when transferring back to core. This is achieved more easily with a tree structure than one with more complex pointer relationships because only one pointer needs to be "cut" to release a complete section of structure.

The disadvantages of the tree structure also stem from its simplicity. They arise because of the lack of cross references between beads, and sometimes from the inability to travel up the tree rather than down it. If extra pointers are introduced these problems can be overcome, but the main advantages of using trees are then removed.

In the case of the triangle the lack of pointers becomes apparent if the whole triangle is to be translated in space.

The operation to be performed is the modification of the coordinate values, held at the lowest level. These can only be accessed by searching down the tree from the triangle bead via the boundary beads, requiring two accesses per point.

In order to save storage space, common elements can be used at the lowest level, but the structure is no longer a pure tree and becomes more difficult to segment. Figure 4.6 illustrates this for the triangle example, and it is evident that less accessing is needed than previously because duplication of data has been removed. This is sensible from another viewpoint in that if only one set of values is to be changed the necessity to search for duplicate data is removed. This type of structure can sometimes be convenient for schematic graphics diagrams where use is made of subpictures corresponding to the common data beads.

Ring structures

The processing time can be speeded up by stringing together those items included in a common processing operation. It would be helpful to put the point beads on a list and to process this list when the triangle is to be transformed. This would imply that the start of the list is known, which may not be so, and it is useful therefore, to arrange that the last item in the list points back to the first, thus forming a RING structure. With a ring, processing may begin at any point and is terminated when the starting point is again reached. Figure 4.7 shows a ring structure for the triangle problem.

Such a structure does not, however, allow for travel back to the parent triangle from the boundaries or points and would involve a search and comparison method of determining the triangle from a pensee on a boundary.

The hierarchical nature of the structure has evolved from a consideration of the triangle problem and the link back to the parent triangle can be obtained by pointing from each point bead to the triangle bead. This also enables it to be reached from a boundary bead from which there are pointers to point beads.

Some conclusions about the use of associative data structures

Several conclusions can be deduced from an examination of the various associative data structures so far discussed.

The use of pointers can simplify the problems of data relocation caused by actions such as deletion. The need to move large quantities of data (usually much larger than in this example) is replaced by the simpler operation of manipulating a few pointers.

If there are many pointers, considerable updating may be needed both within the application program data structure and within the free storage routine lists. If many deletions are performed, the free storage zone may need to be collapsed (i.e. compressed to remove spaces), but at least the task of organising this is removed from the application programmer. For the majority of data structures about half a dozen pointer manipulation routines will suffice to perform most operations including addition of new beads and deletion.

Because of the ability to manipulate the data using a handful of simple routines, the use of data structuring enables the design of application routines to be very straightforward. Various functions can be parcelled into unit operations on the data structure and its contents. Information about where to locate data is contained in the data structure, so that algorithms may operate in much the same way as a stranger who, attempting to locate an address in a town, proceeds by travelling a short distance and then asking again for new directions.

Pointers can be used to link component names (or pictures) with the data describing them, by arranging that the name (or item number in a display file) has the value of a pointer to the bead containing the data. Because the pointer refers to a unique location there is no possibility for ambiguity in the definition of the component. In the case of interactive graphics a lightpen "see" will yield the pointer to the relevant data for the item chosen. The use of subpicture techniques or common components can be dealt with by having common beads in the data structure. The number of uses of the component would then be indicated by the number of pointers to the common bead.

For the structure proposed, the facility of storing mixed data types in the same bead is needed. The ease with which this can be done is a function of the hardware to be used and the level at which the free storage routines are implemented. If the data structuring facilities are included at the language level as in AED-0⁽²⁴⁾ the bead definitions are assigned by declaration statements. If the structuring facilities are provided as a package

callable from a procedure-oriented language (Fortran in Chapters 5, 6 and 7) and the free storage zone is an integer array, the application programmer must take account of the storage mechanisms for the various data types[†]. This problem can be eased by using macros if such a capability is available, where the macro definition takes account of the data type in question.

It is desirable to make beads on any given ring of the same size and type (format). This enables the algorithms which search round a ring for a given piece of data (sometimes known as MOUSE algorithms) to be simplified because each algorithm only needs to operate on one type of bead. Even so, it is wise to store a code at the head of each bead to indicate its type and to ensure that only valid operations are performed concerning the data held in the bead.

The hierarchical ring structure combines the advantages of rings and trees for most processing operations (except use of backing store). However, the large number of pointers involved in such a data structure make it unsatisfactory if core store is at a premium, unless routines are available for storing structured data on secondary storage.

[†]For example storing of a real value in an integer array. With some computers (I.C.L. 4130 and 1900 series) real values are stored in two consecutive integer locations and a routine can be written to store real values in two locations of an integer array. The application programmer must use the routine to reference all real values.

Structured data and the use of mass storage

Some attention will now be paid to the problems of storing structured quantities of data for which insufficient core space is available. As already mentioned, one way to tackle the storage of structured data is to unstructure it first.

This unstructuring approach requires the definition of a format for the data to be stored on mass storage. In addition the structure building routines must be written to accept this formatted data and to rebuild the structure. This rebuilding process re-assigns any necessary pointer values. The simple formatted data strings can be held in a suitable file on disc or drum. The reason for taking this approach is that it obviates the necessity to update pointers, a process so complex that it is often impracticable to re-program for each application.

An alternative is to change the pointer address system by use of a ~~hash-coding technique and~~ paging scheme. This is operated by using a relative addressing scheme, where the storage area is divided into pages (typically of 1K words, but often variable by the programmer) which are assigned a page number. Any addresses are then specified relative to the start of the page. Swapping of programs and data then takes place as whole pages are shifted between disc or drum and core store. In place of absolute addresses ~~a hash-code~~^{an} address is used which specifies both the page number and address within the page of the required location. ~~Suitable hash-coding techniques will allow~~^A a considerable number of pages ~~to~~^{can} be used and ~~will therefore give~~^{is available,} an effectively large storage area, only some of which is resident in core at one time.

Such techniques have been widely implemented by software and hardware[†]. Automatic paging schemes usually create very bad response times for an interactive system with data structure. Cross-references within the structure which traverse the page bounds cause an inordinate amount of page swapping.

The answer to this problem lies in maintaining a given piece of data structure within a whole page. If the structure has been designed to reflect the component nature of many physical systems, and the page size is large enough, this can be done.

If the data structure is suitably segmented, the paging scheme can still be allowed to operate automatically providing the programmer can ensure that a given segment of structure is placed entirely within one page. This implies that new pages can be allocated when desired (even if they are not completely filled). A paging scheme which fulfills these aims is described in Chapter 8.

By using arrays a simple paging scheme of this type could be implemented where the arrays correspond to the pages and the array subscript corresponds to the address relative to the page head. If a structure copying technique is programmed, using structure building routines which re-assign pointer values, parts of a page could be transferred by copying them into a buffer page which would then be written to backing store. The reverse process would be used to read them back into a page in store.

A useful technique ^{is} ~~would be~~ to arrange programs such that picture elements on a C.R.T. correspond only to those segments

[†]I.C.L. 1906A, Atlas.

of data in core. Usually the display file contents need not correspond to the core memory contents and the suggested scheme would need to be deliberately programmed. This approach would enable the user to be aware of the page swaps incurred by a request for an item not held in core.

With an increasing interest being shown in the use of satellite graphics computers, the decision of how to divide a problem into those sections which reside in the central site and satellite machines must be made. Just how this division is organised depends on whether the satellite is remote from the main machine, or adjacent to it with a high bandwidth connection. Much of the software development for satellite computer configurations is being developed on the basis of experience with closely linked machines. As a result, much of this software does not offer adequate facilities and good response times for operation in a remote environment with only a relatively slow telephone line connection. Too often the satellite computer's task is seen simply as to perform picture processing, with the application programs residing in the central computer. Such a division of labour ignores the nature of the associations between the displayed picture and the engineering data which lies behind it.

For this reason it is desirable to contain much of the engineering data locally, implying the need to have a disc or drum attached to the satellite to make this possible. Transfer of information to a main machine for analysis involves many of the same problems as transfer to and from disc. The main common feature is that routines must be available for altering the way in which data is structured.

Conclusions

Several general purpose data structuring schemes have been implemented and perhaps one of the best known of these is ASP⁽¹¹⁾. However, the balance between generality and efficiency often makes the overheads of a general system too high. In his early Sketchpad system, Ivan Sutherland used a general purpose data structure having a ring of points and a ring of lines. For a system mainly concerned with pictures constructed from lines this is adequate, but for various applications it is usually necessary to incorporate more complex relationships than those pertaining to picture topology.

Because the requirements of each application are so different a data structure should be tailored for each new application area. Sometimes an "off the peg" data structure can be suitably modified to cope with new applications. For most problems, about half a dozen pointer manipulation and data structure building routines are sufficient to cover a wide range of operations, if a free storage and garbage collection facility is available[†].

Many difficulties still exist in storing structured data on backing store. As with data structures in core, the most efficient solution lies in organising a scheme around the application problem. This is not very convenient for the application programmer, and one answer appears to be to organise a paging scheme which the application program can administer. This can be achieved by ensuring, as far as possible, that pointers within the data structure do not cross page bounds. Within each page the pointers are

[†]Simple free storage routines can be designed and implemented without difficulty. The applications described in Chapters 5 and 6 make use of three such routines provided by I.C.L.

relative to the page head. This must be taken into account when the data structure is organised in order to attempt to keep data segmented to fit within the page size. If several pages are held in core at once, references from one to another would be acceptable but would have to be kept to a minimum.

When the problems of storing structured data using a virtual memory ~~concept~~ have been solved, larger problems will fit on to smaller computers used as satellites.

CHAPTER 5

LUIA - AN APPLICATION SYSTEM FOR STRUCTURAL ANALYSIS

Introduction

The early design and specification of an application system which is being implemented at Leicester University[†] on an ICL 4130 with ICL 4280 graphical display console is described. The system has been named LUIA (Leicester University Interactive Structural Analysis) and it is written mainly in Fortran. The basic software used is described in Appendix B.

Finite Element Analysis

The structural analysis within LUIA is based on the finite element method. The finite element approach involves the sub-division of a structure into elements whose behaviour is assumed. (35)
When these elements are put together an approximation to the real structure is obtained. The assumed behaviour of the individual elements ~~corresponds~~ ^{amounts} to a choice of either a displacement field or a stress field; LUIA is based on the displacement method. In selecting displacement functions, care is taken to ensure compatibility of displacements across the boundaries of the

[†]The author has been associated with this project for a period of 2 years and worked full time on the early system design and implementation for 10 months. The work was done jointly with Dr. G.A. Butlin, Research Fellow, at Leicester. The original system has been described previously in two papers^(16,17).

elements. The internal displacements are expressed in terms of the displacement values at the nodes of the element. Using these and the stress/strain relationship for the material, a set of nodal force parameters can be defined. By applying the principle of minimum potential energy, a set of equations relating nodal displacements and force parameters can be obtained, expressing a state of equilibrium. The solution of these equations for a given set of loads and displacement constraints enables the displaced form of the structure to be calculated. Again using the stress/strain relationship, the stress values can be found from the calculated displacements.

Data Preparation

The traditional procedure in finite element analysis involves dividing a structure into a mesh of elements. The experience of the engineer is used in determining the mesh shape: areas of high stress concentration will have a finer mesh, as will other regions of the structure where the results may be of special interest. Elements and nodes are assigned numbers. Typical data input consists of:

- (1) A list of node numbers and their coordinates (relative to some set of global axes.
- (2) A list of element numbers and the nodes corresponding to each.
- (3) A list of material properties for the elements.
- (4) Vectors of applied loads and displacement constraints for the nodes.

During the preparation of this data, errors often occur which are not easy to detect. Wrong values for coordinates will mean

an incorrect mesh shape, whilst errors in node lists for the elements will cause a total mis-representation of the structure. For structures of normal size (say 250 nodes for a bridge) the time required to prepare the data is frustrating, particularly if a series of analyses are to be attempted. The engineer is often one stage removed from the computer, in terms of access, and this too can lead to delays in obtaining results.

The engineer is interested in the structure and its behaviour, and to force him to become concerned with internal details of program organisation is undesirable but usually necessary to some degree. The node numbers are of little interest to the engineer from a structural viewpoint, but are important as his means of communication with the computer representation of his problem (mathematical model). Some saving in effort required to prepare data can be effected by incorporating automatic mesh generation procedures. However, if a batch program is used, the difficulty of communicating with the model still exists and may be complicated by assigning node and element numbers automatically (i.e. internally).

Programs for automatic generation of element and node distribution can be written for particular classes of structural problem, and the author has done this for various examples of bridge design⁽³⁾ (see Figure 5.1 for an example mesh). It is not easy to ~~conceive~~^{conceive} of a general scheme for mesh generation because of the wide variety of constraints to be satisfied for different analyses[†].

[†]Sophisticated techniques for mesh generation have been incorporated in the ASKA work in Germany. ASKA (Automatic System for Kinematic Analysis) has been developed at the Institut für Statik und Dynamic at the University of Stuttgart.

As well as for communication, node numbering schemes are important for the analysis phase, where it is desirable to obtain a banded stiffness matrix relating nodal stress parameters to displacement parameters.

Conventionally, the output from finite element analyses is very large. Much of the information is not of direct interest but must be available to check that the other results are correct. A recent analysis of a supertanker produced an output listing which was longer than the ship itself. This difficulty of data assessment is an area where a graphical output would be particularly useful.

Advantages of an interactive graphics approach

Some of the difficulties mentioned are removed when an interactive graphics program is developed for data generation. It has been stated that the designer should not need to be concerned with internal details such as node numbering. A graphics approach should remove the necessity of this and allow the designer to indicate "this node" rather than "node number 10". The nodes do still have a numbering scheme, but this is now an internal detail of the programs.

The ability to adjust a mesh and to judge its shape by eye are particularly important. In practice the engineer is interested in the precise coordinate positions of only a few nodes on the boundary of a structure. Facilities for the fixing of these can be provided in the programs, but in general it is the overall element distribution which is of interest.

Since

~~If~~ node numbering need no longer concern the user, and because of the facilities which can exist for modification of an element idealization, the graphical display allows a new approach to finite element analysis. Philosophically the approach is the reverse of the traditional procedure. Instead of dividing a structure into elements, the structure is put together from individual elements. For several reasons this offers interesting possibilities, but it does create some problems.

The quantities of data involved in the analysis of large structures are too great to be held entirely in core store in structured form. By dividing the physical structure into manageable substructures, the data describing any one of these can be held in core. The solution of the total structure involves the joining together of the substructures, a process which in LUISA is achieved with the same algorithms used to join individual elements. One aim in LUISA is to provide a flexible approach to mesh generation, allowing many different structures to be described by working with a few general algorithms. Because of the ease with which structures may be modified, the process of description has evolved as one of joining together a chunk of elements, or using some previously designed chunk stored on disc, and adjusting its shape to be that of the desired structure. Figure 5.2 shows an irregular mesh adjusted to represent one quarter of a plate with a hole in the centre.

Modes of Operation

In order to perform an analysis in the manner outlined, several basic operations are needed. Those which were originally planned for inclusion in the system are as follows:

- (1) It should be possible to retrieve the data describing previously designed structural components. These data would reside either in core store or on backing store when a request is made for them. If the data are resident in the core store access to them is provided by pointing at the desired item with the lightpen. If the data concerning an item reside on backing store, a nameword or filename previously assigned to the item is used to retrieve the data. The structural components may be single finite elements (which form the starting point for a problem) or some larger structure previously composed from individual elements.
- (2) During the process of constructing a structure the ability to copy any component which has been designed is essential. Such copies may then be used for further building of the structure (as in (3)) or filed on backing store for subsequent use as in (1). This enables copies of various stages of construction to be filed allowing the user to "backtrack" through a problem if he so desires.
- (3) The system should enable the user to compose new components by using previously designed components (or elements) whose data reside either in core or on backing store. The joining of two components involves indicating the boundaries to be joined. The second boundary chosen defines which component must be scaled and rotated in order that the two boundaries are geometrically compatible. If one pair of boundaries have already been joined and further joins are to be made, the operation involved is one of linking two parts of the same object. Under these circumstances, no scaling or rotation

is involved, merely a translation of the nodes to form the join and modification of the boundaries defined by those nodes affected. Figure 5.3 illustrates both types of join. 5.3(a) shows the two components to be joined. Initially boundaries AB and DE are linked, involving scaling and rotation and translation of component II. Boundaries BC and EF are then joined necessitating the translation of point F.

- (4) During composition of a structure, various facilities for modification of the geometry of the components are needed. These include changing of scale and translation of nodes. It is possible to translate nodes according to various constraints to ensure for example that several nodes all lie on a straight line after being moved.
- (5) At any stage of composition it should be possible to apply displacement constraints such as specifying engineering displacement assumptions and fixing of boundary conditions.
- (6) Corresponding to (5) the facility should exist to apply forces at points on the structure other than where displacements have already been fixed. A stiffness matrix is assembled for the structure, and once either the forces or displacements have been specified at all the nodes the nodal displacement parameter values are found by inverting the stiffness matrix and multiplying it by the load vector. The nodal stresses may be calculated from the displacement parameters.
- (7) The user should be able to request the display of a selection of those displacement and stress profiles calculated from the nodal values, and to print them on the line printer if desired.

Data Structure

A diagram showing the major features of the data structure used in LUISA is shown in Figure 5.4. Three types of bead are used: an ELEMENT BEAD, a NODE BEAD and a BOUNDARY BEAD. The contents of these beads are shown in Figure 5.5. The node beads and the boundary beads are grouped on separate rings which are pointed to from the element bead. A ring of element beads exists to indicate the component parts of the current level of assembly of the structure. Each structural component has a data structure of the type shown in Figure 5.4 and when components are joined to form a larger unit a new data structure is created which also has this form.

The data structure for the problem may exist, therefore, in several chunks each of which corresponds to a structural component. Whatever the size and shape of a component, the arrangement of its data structure has the same basic configuration, so that the same algorithms may be used to operate on components which differ considerably both in appearance and material properties.

For the purposes of using secondary storage it was planned that the data should be sorted into unstructured ~~form~~^{format} and stored on disc or magnetic tape as formatted strings. These strings were to contain the necessary information to restructure the application data when reloading them to core. Since that time a scheme has been proposed for secondary storage of

information in structured form[†]. This involves the recreation of the data structure for the specified component in a buffer area, which is then transferred to backing store. When returning the data to core store the structure is copied back from the buffer into the main free storage area. During this process the basic data remain unaltered and the pointer values are re-assigned, but the overall configuration of the data structure remains unaltered (i.e. the associative links remain undisturbed).

Interpretation of stored data

In the finite element method, a continuous displacement field is represented by a finite number of generalised displacement parameters. The nature of these displacement parameters is defined here by a set of codes, which serve to indicate the direction of the displacement and the order of the derivatives in each direction. Each code has four parts:

- (1) 1, 2 or 3 corresponding to displacement in the x, y or z direction.
- (2) An integer defining the order of derivative in the x direction.
- (3) " " " " " " " " y " .
- (4) " " " " " " " " z " .

This set of codes is stored in the Displacement INTERpretation Vector (DINTV).

The pointer to this vector is stored in a component (KDINTV) in the element bead. In a similar way, all the other vectors

[†]Suggested and now being implemented at Leicester by Dr. G.A. Butlin, with the assistance of Mrs. C.K. Grafton.

and matrices containing data associated with an element are pointed to by components of the element bead, the component name taking the name of the vector or matrix prefixed by the letter K.

Values of the displacements are stored, when known, in the DISplacement vector (DISP). The generalised forces corresponding to these displacement parameters are stored when known in the FORCE vector (FORCE).

The basic force/displacement relations in the finite element method are expressed in the equation

$$\text{FORCE} = \text{KMAT} \times \text{DISP}$$

where KMAT is a stiffness matrix. However, the intermediate specification (application) of non-zero forces leads to non-zero increments on the forces not yet specified. These Force INCrements are stored in the vector FINCR, and the more general force/displacement equation becomes

$$\text{FORCE} = \text{FINCR} + (\text{KMAT} \times \text{DISP})$$

When forces are applied, their corresponding equations are eliminated from those represented by the KMAT, and hence a condensed KMAT has to be calculated. A Transformation Matrix TMAT has to be calculated to relate the displacement parameters, corresponding to the eliminated equations, with the parameters retaining their freedom. A parameter is considered free when neither force nor displacement is specified. Only one or the other can be specified. Having specified one of them, the other is determined by calculation.

The derivation of the matrix analysis is given in Appendix A.

For each force and displacement parameter, it will be necessary to store a code indicating whether both force and displacement are unknown or which is known. This code amounts to a structural state variable and the vector of these codes is called the INDicator Vector (INDV).

The connections between elements are defined by a ~~trans~~
connection
~~formation~~ matrix, CMAT. This specifies the relationships between the nodal displacement parameters of a composition of elements and the nodal parameters of the separate elements from which it is composed. Each set of elements which are joined to form a superelement can be considered to be at different levels, where the C matrices relate the parameters at one level to the parameters at the next level above. Thus a complete hierarchy of connections can be represented.

The C matrices provide the means for back-substitution. When the displacement parameters at one level are known, the parameters at the level below can be calculated, using the relevant C matrix, and hence the internal displacements can be calculated along a 'path' penetrating the total structure to the point (or area) of interest. When the basic element is reached the stress/displacement Back Substitution Matrix (BSM) is then used to calculate the basic element stresses.

There is a need for a means of accessing the nodal displacements. This is achieved by storing the value of the component (of the displacement vector bead), which contains the first parameter of the group of parameters associated with a node, in a component of that node's data bead. The name given to this node bead component is KPCOMP.

The number of parameters at each node can, of course, be readily calculated from the values in KPCOMP of two consecutive nodes, but there is a frequent need for this number and therefore it is stored in a component of a node bead called KNPRMT.

The stresses are stored when known in the STRESS vector (STRES). A set of codes is used to interpret these in a similar manner to displacements, each code having four parts:

- (1) 1, 2, 3 9, corresponding to stresses σ_{xx} , σ_{xy} , σ_{xz} ,
 σ_{zz} .
- (2) An integer defining the order of derivative in the x direction.
- (3) " " " " " " " " " y direction.
- (4) " " " " " " " " " z direction.

This set of codes is stored in the Stress INTERpretation Vector (SINTV). Derivatives of stress are necessary to define the stress field in the basic element and will depend on the assumed displacement functions.

The component of the stress vector containing the first of the stresses and their derivatives associated with a node is stored in a component of that node's data bead, (KSCOMP).

For each boundary a visibility code VISIB is stored, to indicate whether a boundary is to be treated by the display as visible or invisible, for a given state of assembly of elements. The convention adopted is that the internal boundaries of a group of joined elements are made invisible once the stiffness matrix (KMAT) of the assembly has been calculated. The internal nodes are then displayed as a spot. However, by specifying a level to the display routine all connected boundaries above that level will then be displayed.

Example structure composed using LUISA

Figure 5.6 illustrates the use of some facilities of the LUISA system employed to define an I section beam. The figure shows a sequence of photographs taken of the display screen. Figures 5.6(a) to 5.6(e) show the processes of joining and copying to produce a larger component starting from one element as the basic unit. In 5.6(f) a quadrilateral device is used to identify those nodes lying within it. The positions of the nodes are then adjusted to lie on a straight line passing through the tracking cross position. Further copies are made and joined to give an I section in Figure 5.6(k) and this is adjusted to give the section of Figure 5.6(l).

Conclusions

Experience gained during early implementation of the LUISA system indicated that considerable flexibility could be attained in generation of meshes, but that a major effort would be needed to organise the large quantities of data involved for the analysis. No information was available to evaluate whether the system would prove practicable since no comparable systems could be found, and as development proceeded it became apparent that it would be necessary to reduce the flexibility, at least for the first attempt. In particular, difficulties were envisaged in the analysis and the decision was taken to use a straightforward analysis method where all the forces and displacements are specified and the resulting simultaneous equations solved. In order to trace back to earlier stages of the problem the

intermediate states of the data structure are stored rather than using the CMAT transformation matrices.

With these restrictions imposed it was clear that LUISA would be most suited to analysis of less sophisticated problems where removal of internal nodes would allow a more rapid response and enable insight into the structural behaviour to be gained. With the ability to use the interactive aspects of the system to "prod or poke" a structure, a general, but not detailed, assessment of its behaviour would be possible. The inclusion of a larger number of nodes allows greater detail but precludes the rapid response which would appear to be necessary for gaining insight.

The other major facility offered by the LUISA system is the flexible manner in which meshes of elements may be generated and modified. It is these data generation aspects of the system which are valuable in tackling large detailed analyses and which offer a rapid and reliable form of data input for conventional analysis programs. The graphical output facilities may be used to provide easy assessment of results not possible with conventional line printer output. One problem arising is that the data generation programs do not contain a sorting algorithm to ^{re-assign node numbers} ~~re-arrange the nodes~~ for an analysis requiring banded matrices. The LUISA analysis is arranged to cope with non-banded matrices, but some thought will need to be devoted to this problem in future if other analysis routines are to be used.

Implementation of the LUISA system has been severely hampered by operating system software faults and poor hardware reliability. All of the data generation facilities operate

satisfactorily and the programs for the analysis are now available. It is hoped that with the next release of the operating system software, the whole system will be operational and that the first structural analyses will be performed in the near future. Only at that time will a full appraisal of the system be possible.

CHAPTER 6

AN INTERACTIVE THREE DIMENSIONAL DRAWING PROGRAM

Introduction

A description is given of a set of programs written to investigate the use of a CRT graphical display with lightpen as a three-dimensional sketchpad. The language used is Fortran and the programs are implemented on an ICL 4130 computer with Elliott 4280 display.

The programs were written in order to study some of the problems arising from three-dimensional display including visualization and communication techniques. The data structure design formed an important part of the study and some modifications to the data structure are proposed.

The work has been described previously in a paper which is listed as a reference⁽¹³⁾. The system is sometimes referred to as TDD (Three Dimensional Drawing). The basic software for data structuring and display are described in Appendix B.

Some Visual appreciation problems

The term "interactive problem solving", used in relation to computer graphics, implies a two way communication between man and computer. Many obstacles exist which complicate this communication process, and they are particularly evident if the picture is three-dimensional. Much effort has been devoted to overcoming problems of visualization: for example, algorithms

for removing hidden lines⁽⁸⁾ and use of stereo views. However, as much as these may aid the viewer in understanding the picture on a display screen, they represent investigations into one aspect of the communication process, namely, the flow of information from the computer to the man. The programs described here are concerned with improving the flow of information in the reverse direction by attempting to provide a natural means for constructing three dimensional pictures.

A truly effective communication cannot be established without considering flow of data in both directions, and it might be thought that improving the man's understanding of the picture will aid him to communicate his ideas back to the computer. This is often true but the use of stereo views is an example which serves to show that this is not always the case. What needs to be established is a feed-back loop where the user can get a feel for what happens on the screen as he controls the various input devices available to him. The lightpen is a very suitable form of input for achieving this loop as it can be used to alter a picture dynamically.

Using the programs

Using a tracking cross and lightpen, only two degrees of freedom can be controlled independently at any one time, corresponding to the X and Y coordinates of the cross. Effectively, in drawing in three dimensions, three coordinate values must be controlled and if this is to be achieved with the tracking cross, one of the coordinates must be related in some way to the other two, or held constant. The choice of how to relate these

coordinates plays a significant part in determining how it will be possible to arrange the operation of the program.

The usual approach to 3D drawing is to define the plane of drawing. The method suggested here, however, is to define planes in which points may be specified and then to allow a drawing to be constructed by joining these points. Therefore ~~drawing~~ ^{Construction} of lines is not limited to the specified planes. In order to define points in space, use is made of a pseudo tracking cross which moves in three dimensions under control of the lightpen. The operation of the programs is based on the use of lightbuttons - words or symbols displayed on the screen which when selected with the lightpen cause some appropriate program to be executed, or option to be chosen. To control the movement of the pseudo cross, use is made of "dynamic" lightbuttons which appear, at the appropriate moment, clustered around the tracking cross on the screen, an idea used by Wiseman at Cambridge, in the PIXIE system⁽¹⁹⁾. Usually this corresponds to the current position of the lightpen and makes selection very rapid, because the relevant lightbuttons always appear close at hand. Some other lightbuttons appear at the right-hand edge of the screen and for major operations, menus of options appear at the bottom of the screen. The arrangement of these menus and the way in which they are used to construct commands using the lightpen are the subject of a paper presented at the Computer Graphics '70 Symposium⁽²⁰⁾. This paper describes a general package of programs for interaction handling developed at Leicester, which enable commands to be constructed and executed. Execution of a command causes an appropriate routine to be entered to perform the action specified in the command.

Commands available in the 3-D drawing system are shown in Figure 6.1 and they enable the following facilities to be used:

- (1) New objects may be drawn, or additions made to existing components. Lines may be deleted using a facility within the drawing routine. The pseudo tracking cross can be caused to latch on to any previously defined point when it approaches within a certain distance of such a point. The test for proximity can be made in both a two and three-dimensional mode and the checking distance can be varied by the user.
- (2) Any ~~previously~~^{previously} drawn or constructed component may be copied any number of times, the only limit being the core space available for data. In practice the drawings become too complex to handle before the available core is filled.
- (3) Items may be moved around the screen and joined to other objects (any their data structures are linked) enabling larger and more complex objects to be constructed from a series of simpler units used as building bricks. Objects are linked at their vertices, rather than by joining surfaces or boundaries.
- (4) Objects are drawn with respect to a set of local axes, and they may be rotated about these axes in order to redefine their orientations, under control of a software joystick, displayed on the CRT.
- (5) Components may also be rotated continuously as a check on their three-dimensionality and as a visualization aid.
- (6) Either the whole picture or individual objects may be rescaled.

When the command "DRAW NEW OBJECT" is executed a set of Cartesian local axes labelled X, Y, Z appears at the tracking cross

position. Objects are drawn with ~~respect~~^{respect} to these local axes and the view presented to the user is the PI-plane projection (i.e. there is an equal direction cosine between each axis and the plane of the display screen). The three lightbuttons XYPLANE, YZPLANE, ZXPLANE appear down the right-hand side of the screen. Selecting one of these defines a plane containing the last defined point in space, parallel to the XY, YZ or ZX plane. This plane is termed the ACTIVE PLANE.

To construct an object, once the drawing routine has been entered and the local axes displayed, requires the use of two console function keys. Both of these are used to defined points, but when one is used a line appears, connecting the present point to the previously defined one. Drawing commences by assuming as the first point the origin for the local axes.

Pressing one of the two console keys on the display unit causes three lightbuttons to appear clustered around the tracking cross. If the active plane is XYPLANE these lightbuttons will be X, Y and N. Choosing one of these causes the pseudo tracking cross to appear, in the active plane.

If the active plane had been YZ the options would have been Y, Z or N, and similarly Z, X or N for the ZX plane.

Points are defined in space using the pseudo tracking cross whose coordinate position is continuously displayed. If a line is being drawn, the length of the line is also shown. If the N option is chosen, the projected position of the pseudo cross in the plane of the display screen is assumed to be coincident with the ordinary tracking cross position. By moving the tracking cross, two coordinate values may be varied and the third determined

from the active plane, Choosing one of X, Y or Z further ^sconstrains the pseudo-cross to move parallel to one of the axes, depending on which lightbutton is seen.

The pseudo cross position may be continuously varied while the console function key is depressed, enabling "rubber band" lines to be drawn. The pseudo cross is programmed to grow larger as it is moved towards the user, a feature which has proved very useful and which enables the user to appreciate that he is not drawing in the plane of the display screen.

Figure 6.2 illustrates some of the steps in drawing the simple figure shown in 6.2(f). In 6.2(a), the arrangement of the screen is shown: various messages are displayed along the top edge of the working area to guide the user, and the menus of options from which commands are constructed appear at the bottom of the screen. The command "DRAW A NEW OBJECT" has already been executed and the local axes and active plane lightbuttons are present on the screen.

6.2(b) shows the dynamic lightbuttons corresponding to the currently active plane which is XYPLANE. A line has already been drawn from the local origin parallel to the Y axis and choosing the X lightbutton gives rise to the situation represented in 6.2(c). In 6.2(d) a rectangle in the XY plane has been completed and the active plane is now ZXPLANE. Selecting the N lightbutton leads to fig. 6.2(e). Continuing in this manner, the completed figure in 6.2(f) is obtained.

Figure 6.3 illustrates some examples of more complex objects constructed from simple units and then modified.

TDD data structure

The data structure used for TDD includes a large number of pointers and cannot be regarded as a compact structure, but it ~~enables~~ ^{permits} the use of simpler and more efficient algorithms. It is, however, a more compact structure than would have been achieved using a general scheme such as ASP⁽¹¹⁾. From a study of this some conclusions have been reached about how to achieve a more efficient structuring for the data without complicating the algorithms unnecessarily.

The data structure used in TDD is an extension of that used in another project at Leicester, the LUISA system^(16,17). The structure is based on the use of rings and beads. A BEAD is a block of data, or more correctly, a block of information, because a bead may contain both problem data and pointers. The POINTERS are used to show the relationships (associations) between groups or beads of data. Three free storage organisation routines are used, for which the free storage zone is a Fortran integer array. These routines were provided by I.C.L. and are also used by the LUISA system described in Chapter 5.

For the purposes of data storage in TDD an OBJECT is said to be composed of EDGES whose ends are defined by VERTICES. For each object, three types of bead are used:

- | | |
|-------------------------|---|
| an OBJECT BEAD | containing information about the object (things like colour or volume or density) |
| several
VERTEX BEADS | containing data about vertices (for example their coordinates in space) |
| several
EDGE BEADS | containing information about the edges of the object (e.g. length, orientation) |

In order to associate the appropriate edges and vertices with a given object, the edge beads are grouped on a ring pointed to from the object bead and there is a similar pointer to a ring of vertex beads. Because a given edge is defined by the vertices at its ends, a pointer to each end vertex is stored in each edge bead.

Figure 6.4 shows the contents of the data beads and Figure 6.5 illustrates the relationships between them. In order to make some of the algorithms more efficient several additional pointers have been included. These are:

- (1) A pointer from each EDGE bead to the bead for the OBJECT containing the edge.
- (2) A similar pointer from each VERTEX bead to the OBJECT bead.
- (3) A pointer from the object to the last VERTEX and another to the last edge on the vertex and edge rings respectively.

Both forward and backward pointers are used for beads on a ring.

In order to process the whole picture, which may include a number of objects, the object beads are grouped on a ring pointed to from a single bead termed the SCENE bead.

When the command "DRAW A NEW OBJECT" is executed a new object bead is allocated and added to the ring of objects. A set of local axes appears at the position of the tracking cross on the screen. The values of the global coordinates of the origin of the local axes are stored in the object bead along with the orientations of the axes.

As vertices and edges are defined, new beads are added to the vertex and edge rings for the object concerned. This involves some updating of pointers and several checks. In creating an edge, for example, it becomes necessary to ascertain whether a

new vertex is being created or whether the ends of the edge correspond to some previously defined vertices. When an edge is deleted it is necessary to perform a check on the vertices at its ends. If these vertices do not form the end of some other edge they too must be deleted and the appropriate edge and vertex beads returned to free storage. Removing beads necessitates further updating of pointers.

From an examination of the data stored in the vertex bead it is apparent that the local and global coordinates of the vertex are stored in addition to the coordinate position of the point on the display screen. Evidently this represents a duplication of data because the three sets of coordinates are related, but it enables simpler algorithms to be used. For example, in using the "latch on" facility in a three dimensional mode, a two dimensional proximity check is made first, using the display coordinates. If this check is positive the test is made in three dimensions using the global coordinates of the vertex. Similarly, the local coordinate values are of interest in some of the other routines.

Conclusions

As stated in the introduction, these programs represent part of a more general investigation. It has been found that the user intuitively watches the movement of the pseudo tracking cross and very rapidly becomes familiar with the concept of working in different planes in space. Making use of the lightpen in the manner described provides the feedback loop mentioned earlier.

One problem which has been experienced has been the time to process the whole scene to produce rotations for purposes of viewing the objects in three dimensions. This is because of the time taken to search round a ring updating the global coordinates. As a result of this a suggested modification of the data structure is to remove the global coordinates from the vertex beads and to place all of them together in one bead. In their place in the vertex bead a pointer is inserted which indicates their position in the global coordinates bead. For complete picture transformations only the global coordinates bead needs to be processed and search times are much reduced.

Experience indicates that it is not necessary to store local, global and display coordinates, and it would appear to be sufficient to store only local values. Global and display values may then be obtained by applying the appropriate transformations, as in the LUISA system.

The ability to rotate objects about the local axes is particularly valuable since it enables the active planes to be redefined with respect to the objects.

Visualization problems have caused some difficulties and conclusions about avoiding these have influenced the design for an application 3-D system described in Chapter 7. The use of a hidden line removal algorithm would not appear to be satisfactory since the picture processing times are likely to be too large.

Communication with a three-dimensional picture needs either a very rigidly defined set of operations involving codenames for identification purposes or the use of devices such as the pseudo tracking cross. The pseudo tracking cross could be used with stereo picture pairs, where it is displayed in each view and is

moved in sympathy with the normal tracking cross.

With three-dimensional problems, much more attention must be paid to the requirements of the display system than with two dimensions. This is mainly because of the need for visualization aids and the way in which such facilities affect the data structure design.

New advances in hardware including matrix multipliers and hardware clipping of lines may make the programming of three dimensional pictures much easier in the near future. There are however, many situations where the type of visualization is very problem dependent, and software solutions to these problems are particularly valuable.

CHAPTER 7

BAID - A PROGRAM FOR BASIC ARCHITECTURAL INVESTIGATION AND DESIGN

Introduction

The program described in this chapter was developed as one of the projects of the Computer Applications Workshop scheme in the Engineering Department at Leicester University. Two versions of the program exist: a batch version, and the one described here, an interactive graphics version.

The idea for the program was that of architect Boyd Auger. Much effort was needed to translate his scheme into a working computer program, but after preliminary discussions with the Leicester CAD group[†], it was decided that the problem was an ideal candidate for a graphics approach.

The program is concerned with the layout of high density housing sites. Once various details of the site have been fed to the computer as input data, a random number program selects the type (flat or maisonette), location and orientation of a new dwelling. A series of tests relating to this new dwelling and those already in existence is conducted to ensure that certain design criteria are satisfied. When the desired density for the site has been reached by repeating the selection and acceptance or rejection process, a variety of outputs can be obtained

[†]The author worked on this program for several months with Dr. G.A. Butlin and the architect Boyd Auger.

including hard copy plots of the completed site.

The program is based on the use of list-processing techniques and the data structure is described.

Organisation of the program

Location of dwellings is carried out by positioning them on a 99 x 99 square grid where the grid size must be large enough for the area to encompass the whole site. Coordinate axes x and y are chosen along two adjacent sides of the grid system. On the grid are plotted "pads", where a pad is a square of grids into which will fit any of the dwelling types to be used in the development. The number of grids defining the pad size may be varied. The pad is assumed glazed on two opposite sides and can have either of two orientations, xx or yy. It may also have one or two storeys (flat or maisonette).

As part of the input data, the designer specifies which areas of the 99 x 99 grid may be used in the positioning process. Existing roads, dwellings and areas outside the site boundary, but inside the grid area, are regions where new dwellings may not be located. Another grid system is used to specify the variations in height of the site.

An x and y coordinate value, measured in grids, is selected by a random number which also specifies the orientation (xx or yy) and type (1 or 2 storey) of a new pad. These last two properties may be varied by weighting factors, entered as input data, the two extremes being all flats or all maisonettes, and all xx or all yy orientations.

As each new pad location is assigned, three basic tests are conducted corresponding to three architectural design criteria.

These criteria are a measure of:

(a) The distances between the windows of different dwellings to test that these are greater than a minimum figure required for privacy.

(b) The amount of daylight falling on each glazed side of the pads.

(c) The number of hours of sunlight falling on the living room facade during ten months of the year.

If any of the calculated figures fall below a minimum specified in the input data, the proposed new pad location is rejected.

Once a pad has been accepted it is added to the data structure and the process is repeated. Checks (b) and (c) involve calculating the skyline shape as seen from the windows of the dwellings.

Any random choice which results in a new pad landing within a specified zone around an existing pad results in the new pad being placed on top of the old one. The size of this zone is part of the input data, and can be varied to encourage the "growth" of tower blocks. If this addition to an existing block causes the maximum allowable block height (again part of the input) to be exceeded, the new pad is left in its selected position and checked in the normal way.

Under certain circumstances, the orientation of the pad is switched if a pad fails a test, and it is rechecked.

The order in which the various criteria are checked is as follows:

- (1) For each new pad, calculate which existing pads are close enough to be tested for the various criteria and set a check parameter to indicate this for each of the existing pads.
- (2) Check the privacy distance.
- (3) Calculate the shape of the skyline from the windows of the new pad and perform checks (b) and (c)
- (4) Add to the skylines for the existing pads the effects of the new one and perform checks (b) and (c) for each of them.

Checks (2) and (3) are only performed if the new pad is on the ground, otherwise it will be on top of a ground pad which has already been appropriately checked.

Data structure

Because the situation is constantly changing as the site develops, the BAID program data can be very conveniently organised using list-processing techniques. Figure 7.1 shows the contents of a bead containing the data for each pad. A pad bead can exist on three possible lists. These lists are:

- (1) a list of all pads, used for various search operations, and for plotting and printing data about the completed site,
- (2) a list of ground pads used to check skylines of existing ground pads when a new one is added,
- (3) a list of roof pads used to construct the shape of the skyline.

Pointers used to construct these lists are held in the first three components of the pad bead. A pad which is sandwiched between a ground and roof pad, termed a middle pad, will only exist on the list of all pads.

The last component of the pad bead points to another bead, that containing the skyline data. This is only present in the case of ground pads and is not allocated if a new pad is placed on top of some existing block. The skyline data is packed, four values per word. The other components of the bead are defined in Figure 7.1.

When a pad has been checked and has passed the specified tests, its bead is added to the data structure by updating the pointers to link it to the appropriate lists.

Interactive aspects of the program

Although there are both batch and interactive versions of the BAID program, it was developed originally using graphical display equipment. The logic used in the program is of such complexity that the use of the C.R.T. for debugging operations clearly had particular value. An axonometric plot of the site is displayed during development.

Interaction via the display console

Interactive facilities within the program enable the user to:

- (1) Interrupt the random selection of pad properties and to specify particular locations, orientations and types for pads using the lightpen and function keys,
- (2) Using the lightpen, to select a dwelling and display the skyline as seen from its window (this can only be done for ground pads). Figure 7.2 shows a photograph of the display screen showing the axonometric site plot and three skylines.

The three views correspond to positions at the left-hand side, centre and right-hand side of the chosen window. The box in which each skyline is depicted represents the limits of the view from the window as defined for the purposes of the tests.

- (3) Change the scale of the displayed picture and also rotate it.
- (4) Request a hard-copy plot of the picture displayed on the C.R.T.

Features (1) and (2) proved to be invaluable as an aid to debugging, enabling extreme cases for each test to be tried rapidly, and for quick visual assessment of the calculated values in each of the tests.

The amount of output printed on a line printer can be varied by setting sense switches during the running of the program.

Output

Three separate forms of output are available: printed information from a line-printer, plots of plan, axonometric or section views, and punched paper tape. Each form of output is optional and may be specified with the input data. The requested information is printed when either the site density is achieved or a specified time limit exceeded. Figure 7.3 shows a plan and axonometric plot for a partially developed site.

Extensions to the graphics version.

The aim of the program is not to automate the process of design, but to produce complex and original arrangements which

will be assessed by the user. Because of the complexity of the tests conducted, without the aid of the computer there is a tendency for the designer to produce small groupings of dwellings and to repeat these groupings over the site area. The BAID program allows the designer to obtain a series of suggestions, which satisfy the design criteria. From these, he can select and "freeze" areas for re-entry as input data, allowing the remainder of the site to develop in a random fashion.

Using the graphics console, these frozen areas could be specified on-line, using a quadrilateral device similar to that incorporated in the LUISA system described in Chapter 5. The parts of the data structure corresponding to the frozen areas could be isolated.

The controlling values for the various tests are variables which the user can modify, and variation of these on-line would enable an assessment of the importance of each test in relation to a given site.

Another use of the program is to assess possible site densities for a given plot of land. If the architect so desires, he may design a site in the traditional manner and feed details of his layout into the BAID program to check the criteria.

Experience of designing the BAID program

Unlike the other applications described in this thesis, the scope of the BAID program was completely decided before the design process began. As a result of this, the entire system was flow-charted before any development or debugging was performed

on the computer. The system was implemented in modular form with each of the test and other operations existing as separate subprograms, allowing easy modification during debugging. The language used was Fortran, apart from two assembly code routines for number packing. The data structure routines are mentioned in Appendix B.

Chapter 8

Data generation for Three-Dimensional Finite Element Analysis

Introduction

The LUISA system described in Chapter 5 was developed to enable two-dimensional finite element analyses to be tackled using a graphics console. At that time it seemed likely, from a structural viewpoint, that an extension of the data structure used for two dimensions would suffice for three dimensions, but this view has been modified in the light of various display and interactive programming problems.

A description is given of the data to be generated using the system. It is shown that the data needs to be segmented in some manner and that this can be achieved by paging the data structure. The difficulties of using a paging scheme in an interactive program are put forward. This is followed by a description of a paging scheme which has been implemented on an I.C.L. 4130 computer in an attempt to overcome some of these difficulties. This is backed up by some test results. The implementation of the problem data structure using the paging scheme is explained, bearing in mind that response times must be kept small. Some discussion is devoted to software required for the representation of

the three-dimensional structures.

The Chapter concludes by outlining future plans for linking the data generation system with analysis programs.

The data to be generated

The basic data needed by the analysis are much the same as those required in LUISA, extended to three dimensions. Little work has been done in this area. Three-dimensional systems based on the use of Coons' patches (Coons⁽³³⁾, Armit⁽³⁴⁾)* have not been designed with analysis in mind, and the Coons' patch approach involves a different form of representation from traditional finite element procedure. Some work has been done on representation of surfaces using a formulation based on finite element functions, but again there is no provision for the requirements of analysis programs (Throsby⁽²⁷⁾).

Three-dimensional bodies to be included can be grouped under the headings of shells and solids. The data needed in each group are slightly different depending on the type of element used. The form of this data, however, can be specified and generally involves the definition of a set of nodes and assignment of various parameter values associated with each node. A reference is given to an introductory book on finite element theory which outlines

* Also the Numerical Master Geometry system implemented by the British Aircraft Corporation.

the organization of this data more fully, for conventional analysis (Zienkiewicz⁽³⁵⁾).

For the designed system it has proved possible to employ a standard data structure, much like that used in LUISA but made more compact in the light of experience and with some modifications made to allow for segmentation. A variety of sub-systems are being evolved for the handling of different types of shell and solid structures, but the overall system organization described here applies to each of these types.

In each group the required data includes a list of coordinates of nodes and a list of elements and their nodes. These fix the mesh shape, the coordinates being specified relative to some global set of axes. For shell structures additional data are required to specify the shape, including various derivatives to define shape functions. These shape functions are polynomials which describe the geometry of curved shell elements.

In addition, vectors of applied loads and displacements must be entered, and the system has been designed so that these can also be prepared using the graphical display. The description which follows relates to the topological and geometric data, because the loads and displacements do not require a complex storage method (ordinary Fortran vectors are adequate).

The data must be stored in some structured manner in order to perform a variety of modifications to the

topology and geometry of the components to be analysed. Moving a node, for example, involves updating any elements to which this node is common and these links are modelled by pointers in the data structure. For most finite element problems the quantities of data are large, and if the topology is to be represented in the manner needed for design using a graphical display there is usually insufficient core space to store this data. Thus some scheme for segmenting the data, making use of secondary storage, is unavoidable.

Correspondence between the physical structure and the data structure

In order to determine the best way of segmenting the data structure a study was made of the quantities of data involved in typical problems using the LUISA data structure. However, the data structure described in Chapter 5 can be considerably reduced in size (LUISA was designed to permit analysis as well as data generation). When this is taken into account, the element data bead requires 7 words, the node bead 10 words and the boundary bead 4 words. These figures are for a full three-dimensional structure, but assuming that components are linked by nodes not by boundaries as in LUISA. No allowance has been made to include surfaces as separate entities because these can be defined in terms of nodal parameters from which the

boundary shapes can be calculated and displayed.

Experience gained from IUISA indicated that a data structure which allows deletion of elements as well as geometry modifications should include duplication of the data describing linked nodes and boundaries. This duplication is removed before performing the analysis by "assembling the structure". This operation is accompanied by the formation of the structural stiffness matrix for the assembly. To distinguish between the two states, the initial structure will be termed the linked data structure, and the other the composed data structure.

Using the above figures for the lengths of the beads in the data structure, the following space requirements can be calculated for the linked data structure. The size of problem chosen is a structure with 400 finite elements. The analysis of a bridgedeck is an example of a problem of this size. The space per element is found by multiplying the bead sizes by the number of nodes and boundaries for the element type. For a parallelepiped with 8 nodes and 12 boundaries the space needed is

$$7 + 8*10 + 12*4 = 135 \text{ words.}$$

For 400 elements the number of words needed is therefore 54,000.

Because the associations between data in finite element problems exist between beads describing physically adjacent pieces of engineering structure, some correspondence between

physical sub-structuring and data structure segmentation has been attempted. The size of a physical sub-structure is, of course, determined by the core space available for storing the data in structured form. It should be emphasized that the use of sub-structures is very important in large analyses in two ways. Firstly, the total problem can be broken into manageable units which are sensible to the design engineer. This provides a means for segmenting a large data structure. Secondly, during the design and analysis process it may be necessary to change a part of the design. This often only requires a re-analysis of the appropriate sub-structure instead of a large computation involving the total structure. It is important, therefore, that the scheme chosen for implementing the data structure allows for the incorporation of physical sub-structuring.

The segmentation of the data structure is achieved by paging the available core and disc space^(28,29,30,31). Paging was chosen because it provides a solution to handling all the data in the same general way. The alternative would be a filing scheme written specially which would not provide this generality. Because of the highly interactive nature of the data generation system, and of graphics programs in general, the choice of paging scheme had to be made with some care to prevent excessive page swapping. This could have a drastic effect on response times. A study of some paging schemes developed elsewhere^(28,29,30,31) revealed

that changes could be made to improve their performance for particular types of problem, as is discussed below.

Paging schemes for list data structures

A major difficulty with the paging of list data structures is that if the space allocation is performed automatically a given list may spread over several pages. Operations which involve searching lists may therefore result in a reduction in efficiency due to page swapping. Page size is a factor which affects this situation. Usually some assignment of priorities to different pages can also help to alleviate this problem.

With small pages it is possible, for some applications, to hold in core all those pages required for the operation in hand. This allows easy handling of the total structure by keeping relevant sections of lists in core together.

In a survey paper on demand paging, Kuehner and Randell (28) discuss the paging of both program code and data, and it is in this area that most work appears to have been done. Different strategies may be applied to the control of page swapping, particularly with regard to the choice of pages to be swapped. The prediction of page requests is a subject which has received much attention⁽²⁸⁾ (Joseph⁽²⁹⁾), but the results of this work cannot be readily applied to interactive programs. In such programs foreknowledge of page requests may not be available because the data structure can change dynamically. Program code is different in that

usually it does not change once compiled and algorithms can be devised to transfer to core (autonomously) those pages which may be called by the one currently being processed.

Some systems have been developed which deal specifically with paging of list data structures (Bobrow and Murphy⁽³⁰⁾, Cohen⁽³¹⁾), but these systems do not relate directly to interactive situations in which response times figure prominently.

Priorities are assigned to pages to determine which page is replaced by a new one being swapped into core. Cohen⁽³¹⁾ has maintained that the priority should correspond to the time of inactivity of a page. That is, the page which has been in core for the greatest time without being referenced should be the one replaced. Another criterion is to assign a priority related to the number of references made to a page while it has been in core.

Bobrow and Murphy⁽³⁰⁾ present an algorithm which is designed to group lists on a single page with a consequent reduction in page swapping. This is a useful feature for an automatic system to possess, but is no more efficient in cases where the page boundary is crossed by a number of pointers. This is a situation which is seldom avoidable and which can only be improved by relating the allocation of space to the problem in hand.

The scheme described below has been designed with this difficulty in mind and allows the application program to control space allocation by arranging that each page is a separate free-storage zone in which beads may be allocated for data storage. This also overcomes some of the problems which occur with garbage collection⁽²⁸⁾ since this is done separately for each page.

It must be emphasized that the scheme has been designed with the long term aim of running it on a medium sized machine without multi-programming. Most of the available references relate to larger computer systems where many more factors must be taken into account, and where it may be less efficient (in an overall sense) to use a paging scheme with a high degree of programmer control. In such systems the time-sharing supervisor must control the swapping of programs and data, and a paging scheme which will operate efficiently in conjunction with this is necessary⁽³²⁾. However, some of the general difficulties of paging apply equally to a dedicated computer system.

Description of the paging scheme developed on
the I.C.L. 4130 computer

The scheme uses a random-access file for storage of pages on disc and an array for working space in core. Word transfers can only be made from or to the start of a disc sector (64 words) and page sizes are an integral number of sectors. Pages are of variable length and may be extended if desired. Although pages may be of any size which is a multiple of 64 words (subject to them fitting into the assigned array in core), the data generation system will make use of large pages for storage of structural analysis data (typically 5K words). Such a size of page is adequate for the storage of complete sub-structures (for example it will accommodate 37 parallelepiped elements with a linked data structure). This enables a sub-structure to be defined by the fact that it is contained in a single page, and connections between sub-structures to be represented by pointers which traverse the page boundaries. This is a major reason for choosing a large page size.

One general disadvantage of a large page, mentioned by Kuehner and Randell, is that considerable unwanted quantities of data may be transferred to core when a page is needed. With the described scheme, those items of data which are most closely related are grouped, as far as is possible, on one page. Thus, although the requested data may be only a fraction of the page content, the next operation will very likely be performed on some related item in the same page.

Different classes of data can be identified, depending on the time taken to process them. For example, messages stored as character strings take very little time to output but an engineering component may take some time to draw because its description must be extracted from a complex data structure. Thus, in some senses, the engineering component data is of higher priority than the message data. It would be unwise to arrange that a page of message data, of low priority, replaced a page of structural data which is of higher priority. This is overcome by assigning a class number when a page is allocated. Pages with different class numbers are loaded into separate areas of the array in core. The class number may be changed if desired, but only a limited number of different classes should be defined, depending on the size of the array used for paging and the size of page to be accommodated in a given class.

Within a given class, pages are loaded one behind another into the appropriate class area in core, when they are requested. When no space remains in the class area the process begins again from the bottom of the area. Any pages in that class which have been updated since being fetched into core are written back to disc before being overwritten in core. This "end around" storage procedure

makes the management of the core space very straightforward. If the application program is designed to group together references to a particular page, then the criterion approximates closely to the time of inactivity used by Cohen.

The algorithms which control the paging are written in Fortran and assembly code, but all the routines are callable from Fortran. Because of this the Fortran programmer can administer any part of the control of paging if he wishes. The routines will automatically update the administrative data used by the scheme.

Administrative information is held in three COMMON areas. these are:

- (1) A page directory,
- (2) A class directory,
- (3) A free space map for the disc file.

The page directory contains the following information for each page:

- (a) The class number,
- (b) The disc address,
- (c) The core address (-ve if the page is not in core),
- (d) An eight character Alphanumeric name (optional),
- (e) The length of the page,
- (f) Aflag to indicate whether the page has been updated since being fetched into core.

The class directory contains for each class:

- (a) The lower limit array index for the class area,

- (b) The upper limit array index for the class area,
- (c) A pointer to the start of the free area.

The free space map is arranged as a list of beads each containing the address and length of free areas on the disc. Free areas are created when pages are deleted or extended. When there are twenty such spaces, or if no free area is large enough to allocate a new page, the disc file is re-ordered to move the spaces to the end of the file. A problem with software paging is the need to use modified addressing for each item in a page, allowing for the position of the page in core. In the scheme described this is normally performed by checking the page directory, and trapping requests for pages not in core. A significant improvement can be gained if the application programmer wishes to make many successive accesses to one page. After ensuring that the page is in core, the position may be determined once and used to reference the page contents without trapping every access.

Results of tests with the paging scheme

Figure 8.1 shows a table of average swap times for a variety of page sizes. These times were obtained from test runs and they include updating of the directories.

The results show that there is little difference in the time to swap a 64 word page from that required for a 5K word page. The scheme appears unfortunately slow and this is attributed to two factors:

- (1) The computer has moving head discs. During the tests, the read/write heads were observed to move across about 30 cylinders of the disc per access, requiring 120 msec. The heads are repositioned at the start of the file prior

to each access by the assembly level filing routines. Thus the actual disc access time is increased to an average of 240 msec. for each transfer.

(2) The scheme is written mainly in Fortran. It is therefore considerably less efficient than if it had been written entirely at a lower level. The availability on the 4130 of extra fast registers and the ability to use fixed locations would remove the need to pass large numbers of parameters between the routines as arguments. Fixed locations were not used because it was not clear at that time how the operating system moves programs around the core when multiprogramming. The scheme would make use of fixed locations on a dedicated computer.

In order to assess the suitability of the developed scheme for the data generation system, the way in which it would be used must be examined.

A complete sub-structure can be accommodated in one 5K page. Manipulation of such a sub-structure involves no swapping. An important time is the period required to change from one sub-structure to another. If only a small ^{are} page size is used, many page swaps ^{are} necessary, and alternatively if only one page of 5K is used, only one swap is needed. Figure 8.2 shows a graph of time to swap to a new sub-structure plotted against page size. The two extremes obtained were 34 seconds for a 64 word page and 0.6 second for a 5K page. The latter is an acceptable speed for the application program, subject to special provision being made for the handling of boundaries between sub-structures. The difference in times is another incentive to use a large page for structural data.

The data structure for the data generation system

The data structure contains three basic types of bead. These are:

- (1) An element bead containing information about a single finite element of the linked data structure, or collection of elements after assembly.
- (2) A node bead containing information about the node, including its coordinates and a pointer on a list of connected nodes.
- (3) A boundary bead containing pointers to end nodes.

Figure 8.3 shows the contents of these beads in more detail.

Joining of elements is achieved by linking nodes. Node beads corresponding to these links are put on lists to indicate which other nodes they are connected to. A node can only exist on one such list. Different substructures are held in separate pages, so that requests for node information contained in other substructures can be trapped by examining the node pointer. In this way the links between separate substructures can be handled differently to avoid excessive page swapping.

When a change is made to the boundary of a sub-^{is}structure, an updating list constructed for each reference to pages not in core. The updating list contains a pointer to all affected locations and their new values. When a new substructure is brought into core, the updating list is scanned and the new values are inserted in the data structure. Only one update per location is permitted on the list, to avoid ambiguity.

Linking of substructures is performed using another list which contains linking information (i.e. pointers to

linked nodes). Affected pages are brought into store one at a time and the list information is used to insert pointers in the node beads to place them on the linked node list.

The representation of the three-dimensional elements

The usual problems of visualization arise and a variety of techniques and conventions are necessary to prevent these from leading to serious limitations on the types of structure which can be handled. Fewer problems are apparent with shell analyses than with solids, but in both cases some common software needs can be identified. A major requirement is a package of display programs capable of treating the display as a three-dimensional device. Particularly important is a three-dimensional clipping program for performing windowing. The major use of this will be for removing unwanted detail in the Z coordinate direction (normal to the display screen), thereby providing, as far as is practicable, a user-oriented hidden line removal feature.

A rotation program is included to enable selection of the most convenient view of a structure. Rotation achieved by software has the advantage over hardware that various axes of rotation can be selected.

It will be possible to assign a visibility level to each displayed element. This enables different elements to have a variety of brightnesses including an invisible state. Such a procedure allows dimming internal boundaries of a structure to distinguish them from its external surfaces (in the case of solids).

Manipulation of the structures

The definition of structures needs a variety of programs for assignment of parameter values. For shell problems one way of working is to distort initially flat sheets of elements to conform to the surface which it is desired to represent. It is possible to formulate this procedure using finite element theory and some results are available from work already performed in this field (Throsby⁽²⁷⁾). The resulting parameter values are those needed to define the shape functions already mentioned.

For solid problems the most promising approach appears to be the distortion of blocks of elements to represent the desired component. It is particularly important to employ three-dimensional windowing for this purpose. The display file only contains information about displayed items inside the window. Modifications are performed on the application data, from which the display file is compiled. Thus any items related to those being changed are updated although they are not^{necessarily} displayed.

For communicating with the digital model via the displayed picture a number of devices can be used. These include the pseudo tracking-cross of the TDD system and the quadrilateral device from LUISA. In addition a "spider" device will be implemented. The "spider" is an extension of the pseudo cross concept with the additional restriction that the cross is confined to move on a mathematically defined surface (e.g. a cylinder). Thus for shell problems, it will be possible to modify mesh shapes whilst ensuring that the shape is accurately represented. Stereo views of the constructed components may eventually be made available as an option. This is easily organized when using homogeneous

coordinates by assigning appropriate values to the terms of the transformation matrix and calling the display program twice. The pseudo cross and spider devices can be programmed to appear in each image. These devices operate with respect to the local coordinate systems for individual substructures, which can simplify some of the coordinate transformations needed. This kind of use of local coordinate systems is to be found in the APT language⁽²⁶⁾.

Requirements for interfacing to analysis routines

The method to be adopted for interfacing with the analysis routines has recently been implemented and tested in the LUISA system*. Because the LUISA system has shown the feasibility of separating the tasks of data generation and analysis, the interface is not discussed in detail. However a brief description of the interface is given below.

In each node bead is stored a global node number. This number identifies the position of the data for that node in the vectors and matrices used in the analysis. These vectors and matrices include the applied loads and displacements and nodal stiffness matrix for the structure. The same relationship, using global node numbers, allows the results of the analysis to be associated with the described data structure, permitting the display of displacements and stresses on the graphical display. Thus the system provides for the viewing of the results as well

*This is a development beyond the early system design described in Chapter 5.

as data generation facilities for input.

Because of the straightforward manner in which the interface is achieved, the analysis program can be run as a separate batch program, either locally or on a larger remote computer to which the local machine is connected as a satellite.

CHAPTER 9

GENERAL DISCUSSION

The design of interactive system software is a task requiring much effort. Several factors contribute to the existence of this state of affairs, one of these being the problem of software transferability. One estimate has stated that only about two percent of programs are transferable between different computers because of alternative internal machine organisation. Another contributory factor is the lack of acceptable low-level software and knowledge of techniques for graphical communication and data structuring. Much expertise has been gained in these areas by many research groups working with computer graphics, but the results of such research are often not readily available in a form suitable for use with application programs.

One means of gaining this knowledge and identifying the necessary software is to study a variety of applications. Three application systems are presented in this thesis and from them several common features have been isolated. General purpose software has been designed to set up and use messages, light-buttons and data structures⁽²³⁾. Filing schemes are being developed to simplify the transfer of application data to disc, drum or magnetic tape. It is important to emphasise that the common features are being found by studying and implementing trial application systems[†]. Provision of general packages

[†]A Computer Applications Workshop is in progress at Leicester University where industrial knowledge of applications is being combined with University research to identify common software needs.

can be made without reference to the applications, as with some manufacturers' software, but the accompanying restrictions often result in such facilities being ignored by application programmers. At the other end of the spectrum, if the general programs are not provided, each application programmer will implement his own programs for file handling and other tasks, with consequent duplication of effort and absence of compatibility between programs on the same computer.

General low-level software packages should be provided for three-dimensional work with displays. In particular, it would be valuable to be able to treat the display as a three-dimensional output device by specifying three coordinates to the display routines instead of just two[†]. If the application is concerned with a three-dimensional problem it should not be the concern of the programmer to have to produce a two-dimensional projection for display purposes.

A major area of interest to application programmers is data structure design. From the applications presented it has been concluded that a pointer structure (plex structure in AED terminology⁽¹⁾) offers the most flexible approach to designing data structures which are entirely core-resident. Such structures do carry the overhead of space needed to store the pointers, but this is a necessary compromise for interactive programming.

Software packages should be of sufficiently low level of implementation to be efficient without being too restrictive. There is a danger that the lowest common denominator of common

[†] Available with GINO⁽²⁵⁾ used by Cambridge University C.A.D. Group.

features is very small. Data structuring facilities can be included in a language compiler[†], but if the structured information is larger than the core space available a number of problems arise. It was as a result of these problems that paging schemes were developed based on a virtual memory concept. General, automatic paging schemes for data, however, create such poor response times, if many page swaps are needed, that they are impracticable for highly interactive systems. It is essential, if these problems are to be overcome, that some control of page allocation and content definition is assigned to the application programmer. Programs should be arranged to keep page swapping to a minimum or, at least, to make the user aware of any page swaps he may incur by issuing a request at a particular stage of analysis. A scheme of this type ~~is proposed~~ ^{has been developed} for the system outlined in Chapter 8, ~~but it is very application dependent.~~

Until general solutions which can be implemented in an acceptable manner for use by applications systems, at a low level for the sake of efficiency, can be found good progress can be made by using Fortran level packages for data structuring^{††}, as in the application systems described.

General paging schemes for data may be suitable for interactive programs operated from a teletype console, where computation takes place in a series of discreet steps. Such programs can run adequately under a time-sharing operating system where processor time is allocated to the user in slices. Bigger machines having a time-sharing system capable of supporting a large number of teletype users often provide for paging of data.

[†]As in the AED-O programming language⁽²⁴⁾.

^{††}See Appendix B.

If enough processor attention can be devoted to the display, various devices can be used to aid the man-computer communication process[†]. These devices provide the means to manipulate models in a flexible and dynamic fashion. The devices amount to a set of control mechanisms, and their use can be likened to the function of a steering wheel in a car, where the amount by which the wheel is turned is gauged in terms of the car's response. In this way, the response of a physical system can be linked with the variation of some input parameter enabling insight into the behaviour of the model to be gained. A simple linear movement of a tracking symbol can, for example, be transformed into a complex variation in the shape of a mathematically defined surface.

Provided the behaviour of the digital model is well understood, less sophisticated techniques of communication will suffice and, with the exception of dynamics problems, for many applications the facilities offered by a storage tube and teletype are adequate^{††}. In other areas, and in particular in some finite element analyses, the problems are not well understood and the ability to probe using flexible communication techniques is very valuable.

Although advances in hardware are being made, it is desirable to search for the software solutions to the many problems which exist in the use of present computer graphics hardware. Many of the software solutions can then be transferred to various output devices and cheaper forms of hardware. For example,

[†] Examples are the pseudo tracking cross in Chapter 6 and the quadrilateral device used in the LUISA system (Chapter 5).

^{††} This would be true of one version of the BAID program described in Chapter 8.

transformation routines and programs for three-dimensional work can be implemented for dynamic graphics, storage tubes or digital plotters.

A modular approach to software design is desirable from a number of viewpoints. Breaking a problem into its component parts enables their separate aspects to be individually studied, allowing their different requirements to be satisfied. Modular programming allows efficient overlaying of program code for large systems. Development of program modules can proceed in a series of well defined steps with each module tested and filed as the system is implemented. The effort required for this process must not be belittled and many changes will result as development proceeds until the final system has evolved. A context editing program is needed to make possible the modifications to the system, particularly a modular system with many cross-references between modules[†]. This should be arranged to operate on source code files stored on disc, with the ability to compile and test-run the program when the desired changes have been made. For the sake of efficiency, an editing program of this type should be used in a time-sharing environment.

Graphics can fulfil a particularly useful role in the debugging of complex programs. Errors which would not normally become evident from a numerical print-out are often brought to the attention of the programmer by the occurrence of alarmingly

[†]An example modification is to change the common blocks in a large number of Fortran subroutines. This is easily achieved with a context-editor which will locate each occurrence of the block and make the desired modification.

distorted pictures. This aspect of using graphics was of significant value in developing the BAID program described in Chapter 7.

Dynamic and flexible communication using a variety of devices requires more processor time than simple teletype programs. For some operations continuous attention is demanded, and this has led to an increasing tendency to use a small computer to handle interactions, and to connect it as a satellite to a larger machine for performing calculations.

There are two distinctly different approaches to using satellite computers. One is to situate the satellite adjacent to the main machine enabling a complex operating procedure to be operated with high data transfer rates. With such a configuration the satellite becomes a sophisticated peripheral whose main task is to service the display console. A different approach, gaining in popularity for economic reasons and because of a shortage of suitable main computers, is to connect the computers over telephone lines. This configuration requires totally different concepts for dividing labour between the two machines with much more computing done at the satellite. Connection of machines over telephone lines is generally distinguished from the other configuration by describing it as remote satellite operation.

Suitably organised, many graphics programs only require a small or medium size of computer, of medium speed core cycle time (2μsec). A remote satellite machine with 32K words of core store (18 bit word length) and disc storage would provide adequate local computing power to make a 2400 ^{baud} ~~band~~ telephone line

sufficiently fast for most applications of the type described. The cost of this hardware is within a range which makes it attractive to many industrial organisations, if its ability to solve realistic engineering problems can be demonstrated.

For a given computer configuration it is important that compatible packages of software are provided. With a satellite and main multi-access computer configuration used for the development of realistic engineering systems where large amounts of data must be handled, it is important to be able to access files in the main computer from the satellite.

It is undesirable that interactive requests are dealt with over the link, with a remote configuration. The response times are too large for such a procedure to be satisfactory. Ideally, programs should be organised to use the main computer for "number crunching" with data passed to and from for interactive examination executed at the satellite.

Computer graphics has been an area of active research and development for more than ten years. During this period its potential as a design aid has always been recognised. Only now that the software problems are being tackled does it appear that this potential will be fulfilled.

Chapter 10

Conclusions

Eight main conclusions have been drawn from the work presented. They are:

1. Application software development requires a considerable and well directed effort. Two major reasons for this are a lack of suitable general purpose software for interactive programming, and a shortage of techniques for graphical communication and data structuring. Provision for both of these shortages must be related to the application programs to be tackled.
2. A number of common features can be identified for a cross section of application areas, and this has been done for the systems described in the thesis. Packages developed as a result of the investigations conducted include programs for organization of messages, lightbuttons, paging of data structures and file handling. These packages were extensively affected by the applications which use them.
3. Plex structures for data storage are very powerful but carry the overhead of the space needed to store the pointers. This is an acceptable restriction in the problems tackled because the advantages outweigh the inconvenience of performing the digital modelling in some other way. (i.e. the algorithms can be designed more simply if space allocation and garbage collection are performed by a general package.) If backing store is to be used for larger problems, paging can provide a suitable mechanism for segmenting the data structure. Most paging schemes are not sufficiently problem-oriented to provide an

efficient solution to the difficulties of storing data in lists on secondary storage media. The paging scheme described in Chapter 8 overcomes some of the usual difficulties, and from test results appears satisfactory for the types of application discussed. Using the paging scheme, realistically large problems can be tackled in 5K words of data space.

4. A variety of devices, provided by software, have been developed which permit implementation of dynamic techniques for graphical communication with a digital model. An example of such a device is the pseudo tracking cross used for three-dimensional drawing, which could be equally well employed for communicating with stereo pairs. Programming of these devices extends the range of operations which can be achieved with the lightpen. They are based on the two major functions of the lightpen of seeing and tracking.

5. Software solutions to various difficulties are in many respects more attractive than hardware solutions. Software should be developed in modular form to permit easy extension of system capability and efficient overlaying of program code in the core space available (large systems invariably require an overlay facility-LUISA, for example, is too large for a 64K machine with 24 bit words unless overlays are employed). Development of such large programs could be boosted significantly by an on-line context editing facility.

6. Discrete^{te} representation of engineering components, as with the finite element method, is ideally suited to the organization of data in structured form in the computer store. Segmentation can be arranged to coincide with the

division of an engineering component into substructures or sub-components. This provides an opportunity to segment the data in an efficient manner such that response times are kept to a minimum.

7. Graphical representation has been of particular value during the debugging of some programs by avoiding the necessity to scan many sheets of output.

8. Programs of the type described could be efficiently implemented on a small computer connected as a satellite to a large time-shared machine. With this configuration, the larger computer should be used for performing the analysis computations, but most of the data should be stored locally to achieve fast response times. The performance of the satellite would depend on the provision of an overlay scheme for program code and paging of data. A satellite computer with 32K words of 16 or 18 bits would be adequate for the applications described if overlaying and paging are available.

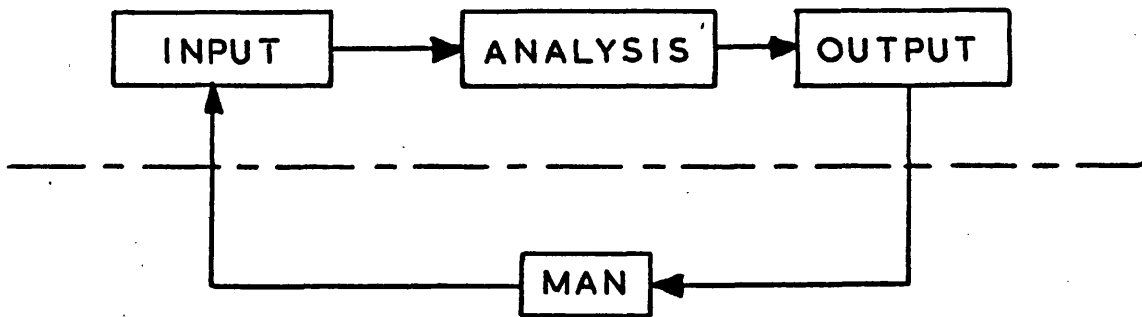


FIGURE 3.1

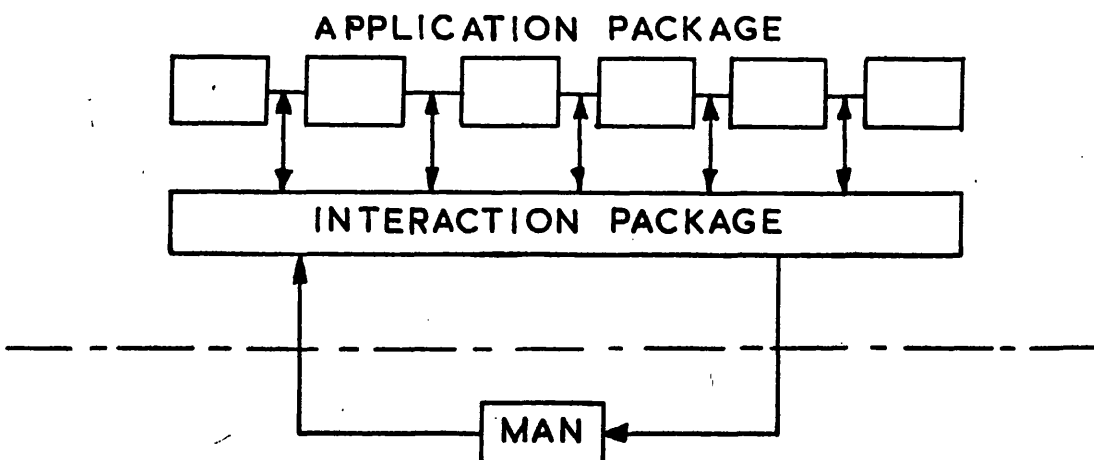


FIGURE 3.2

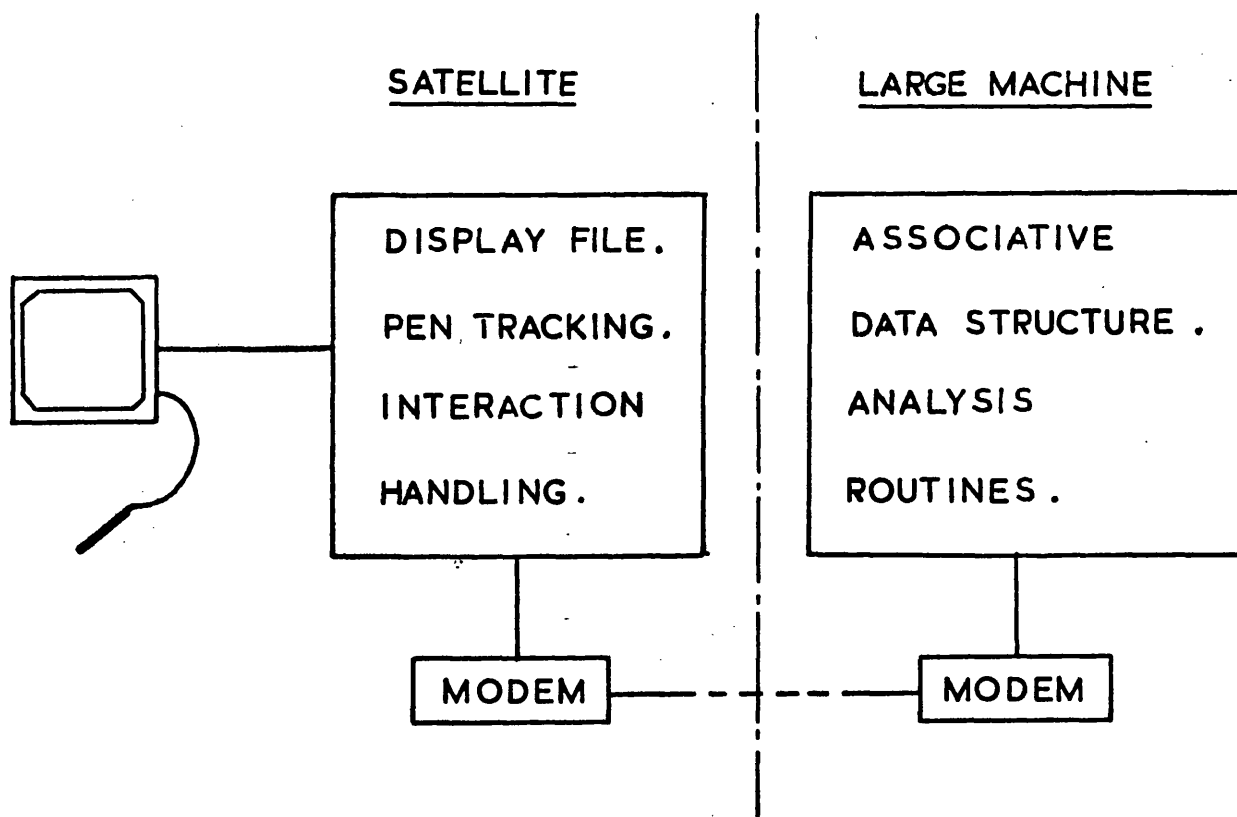


FIGURE 3.3

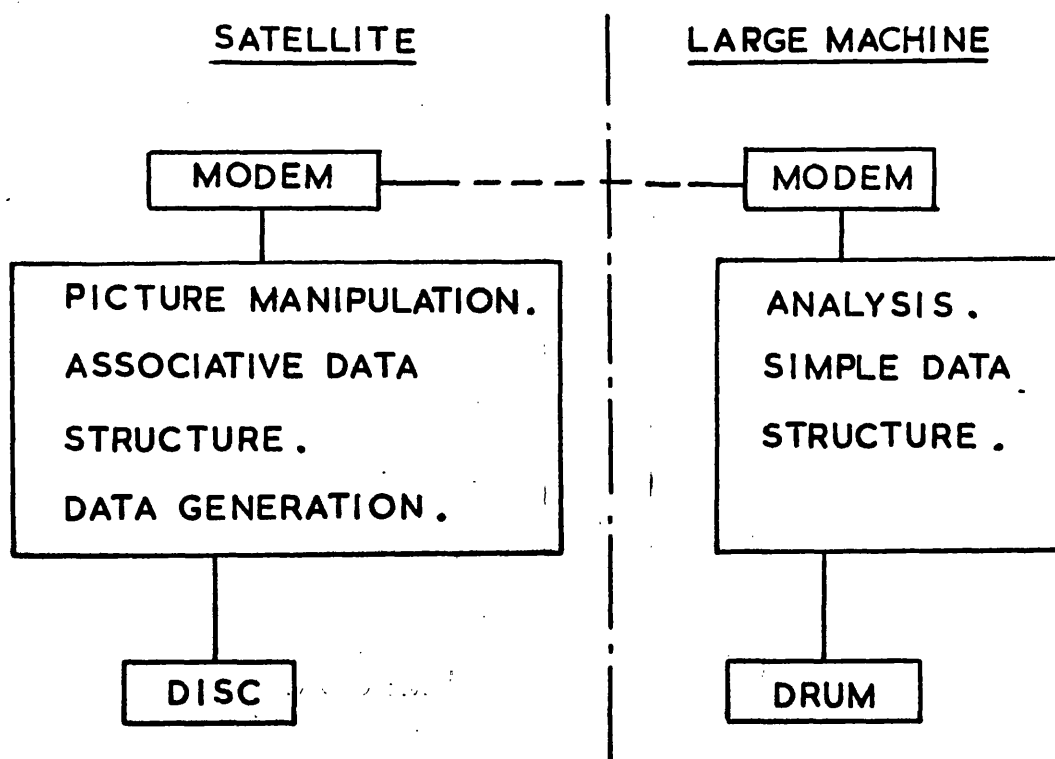


FIGURE 3.4

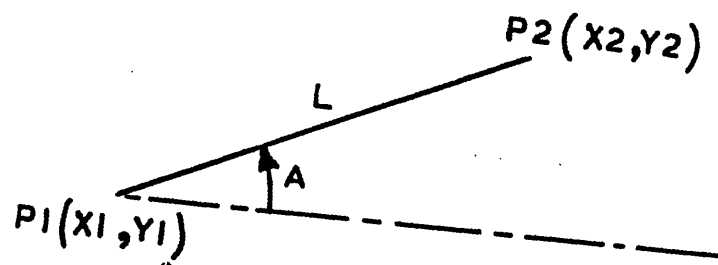


FIGURE 4.1

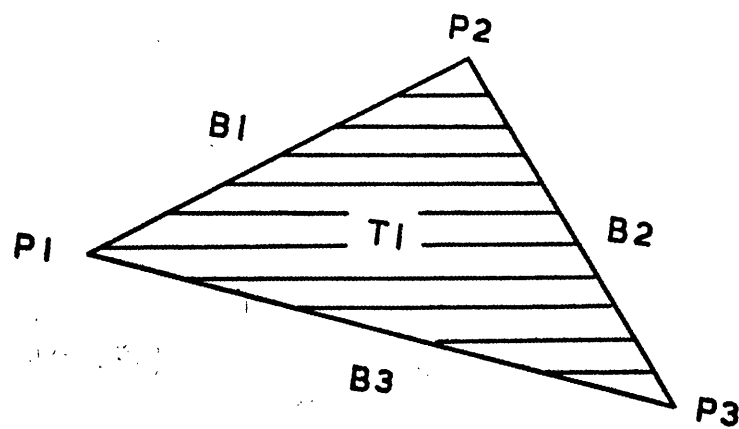


FIGURE 4.2

END1.END2.TRI. ITEM.

B1	P1	P2	T1	I1
B2	P2	P3	T1	I2
B3	P3	P1	T1	I3
B _n	P _k	P _l	T _q	I _n
B _{n+1}	P _l	P _m	T _q	I _{n+1}
B _{n+2}	P _m	P _k	T _q	I _{n+2}

BOUNDARY

	X	Y	Z
P1	X1	Y1	Z1
P2	X2	Y2	Z2
P3	X3	Y3	Z3
P _k	X _k	Y _k	Z _k
P _l	X _l	Y _l	Z _l
P _m	X _m	Y _m	Z _m

COORDINATES

FIGURE 4.3

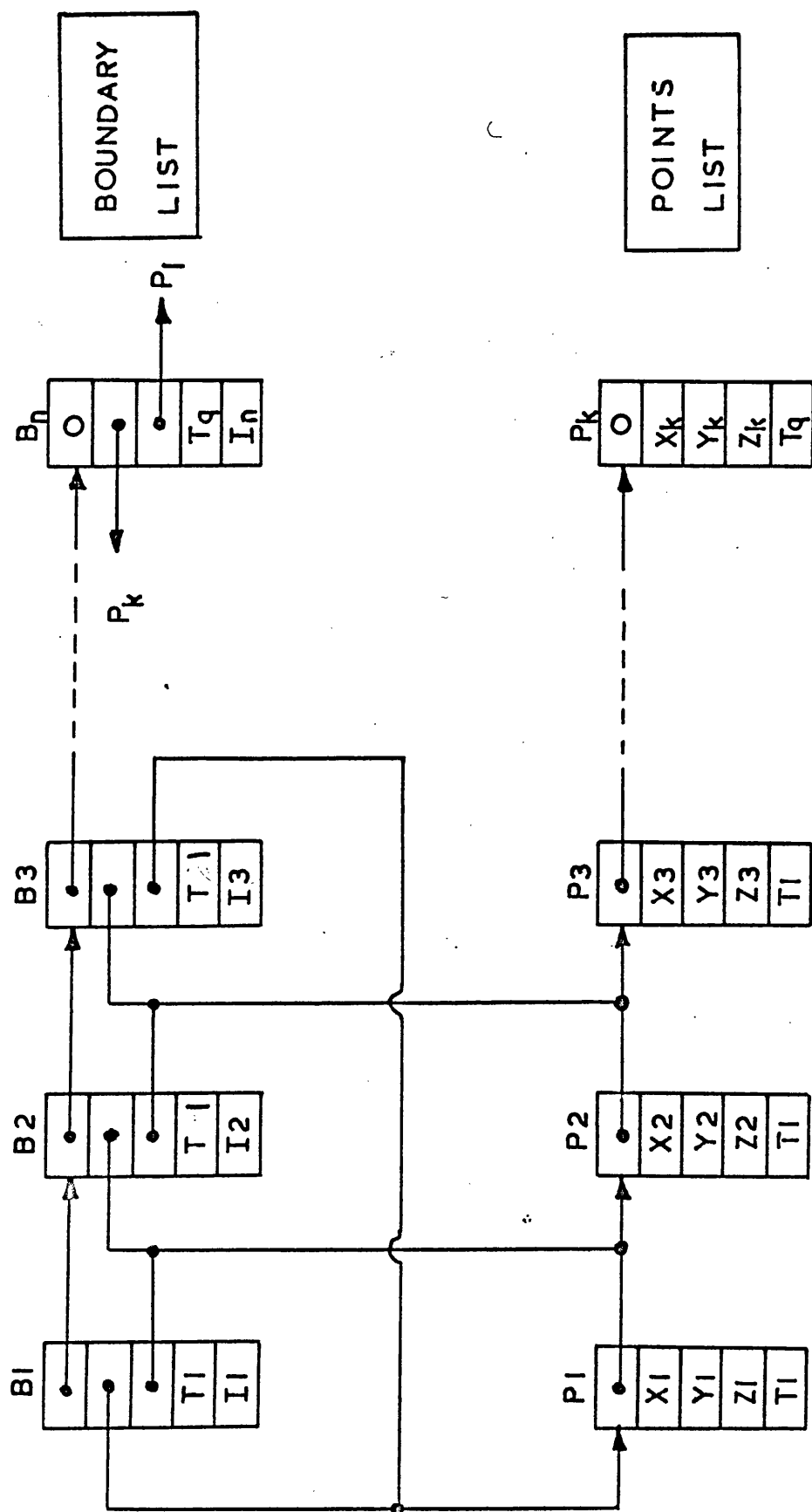


FIGURE 4.4

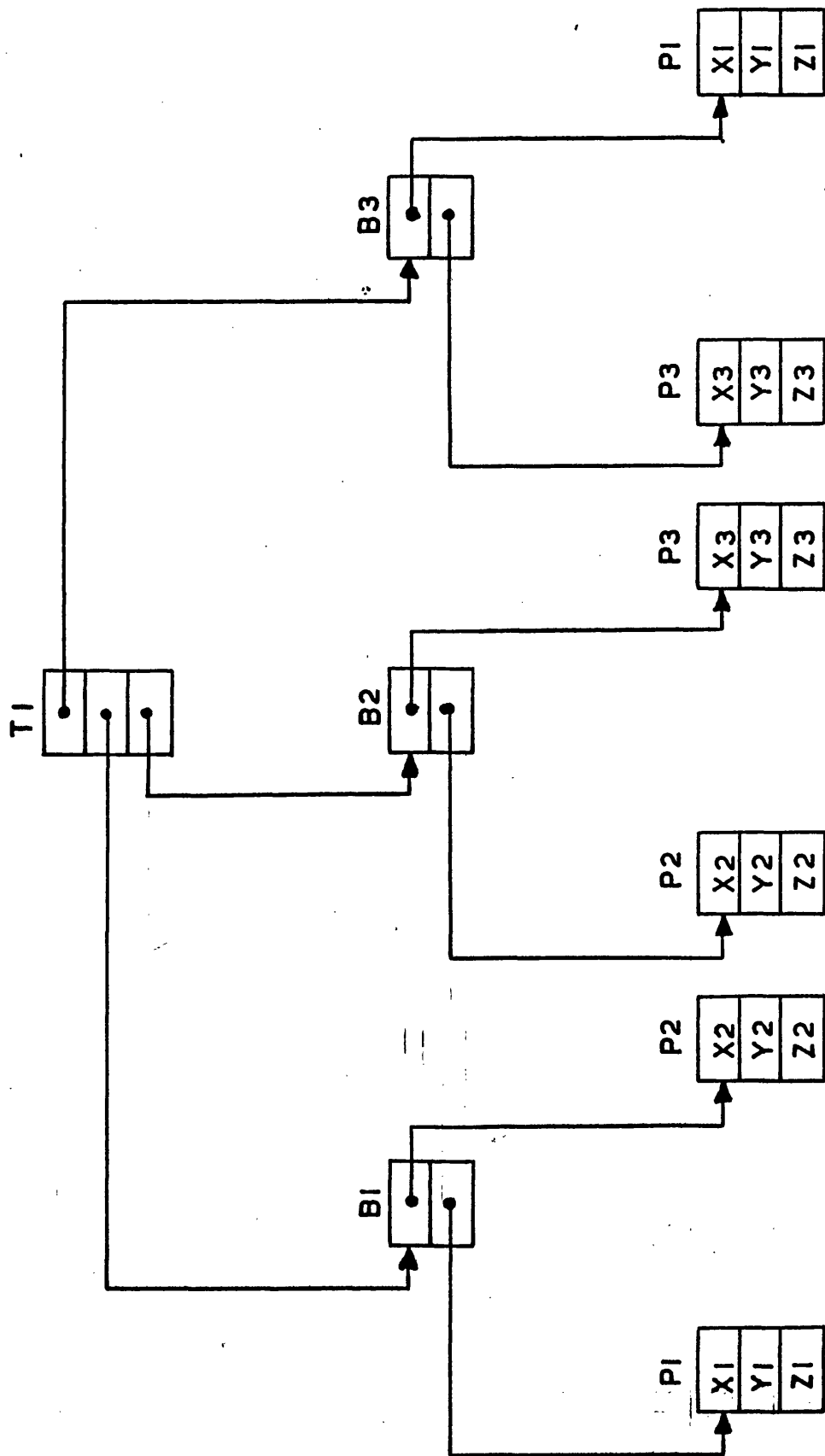


FIGURE 4.5

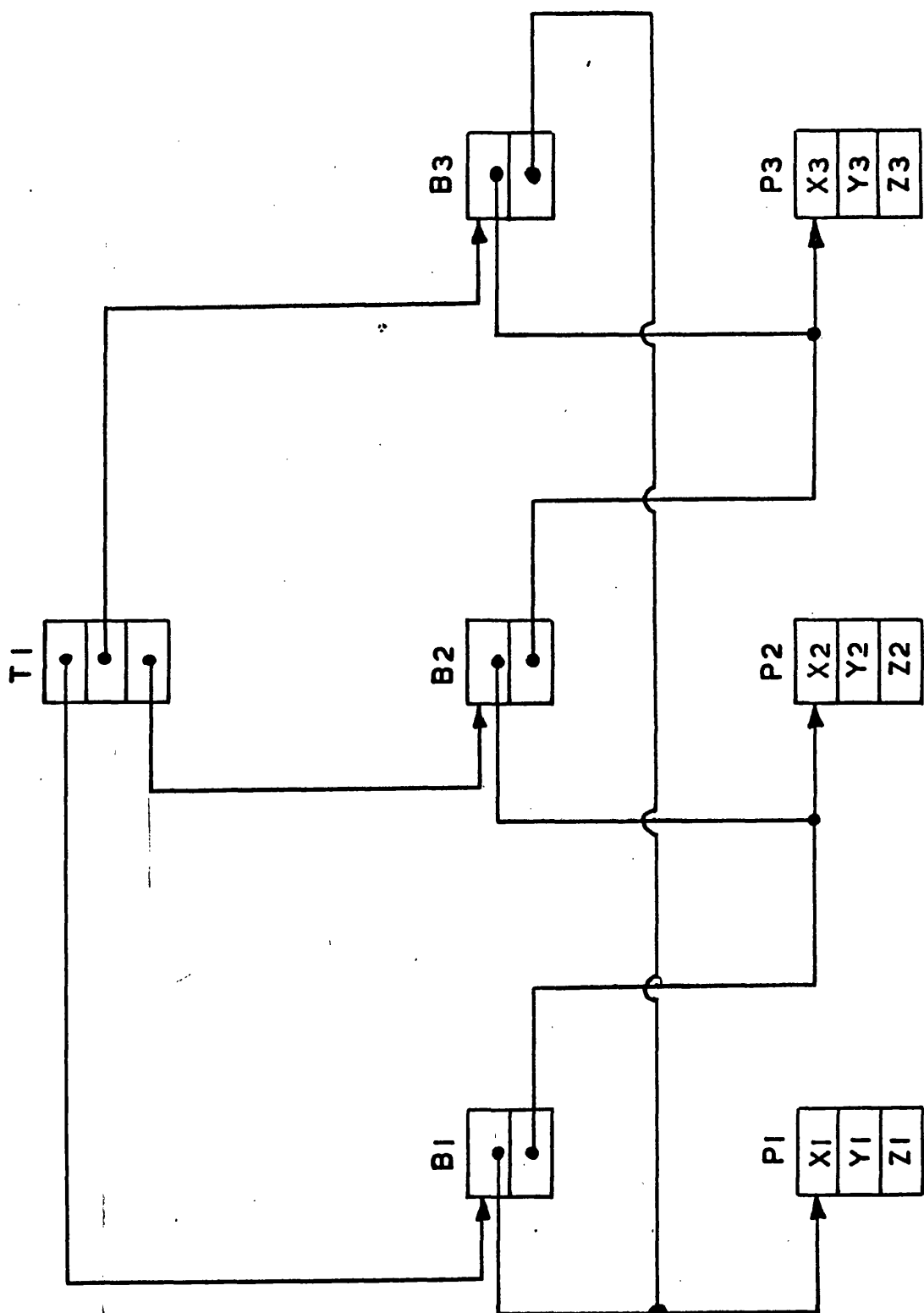


FIGURE 4.6

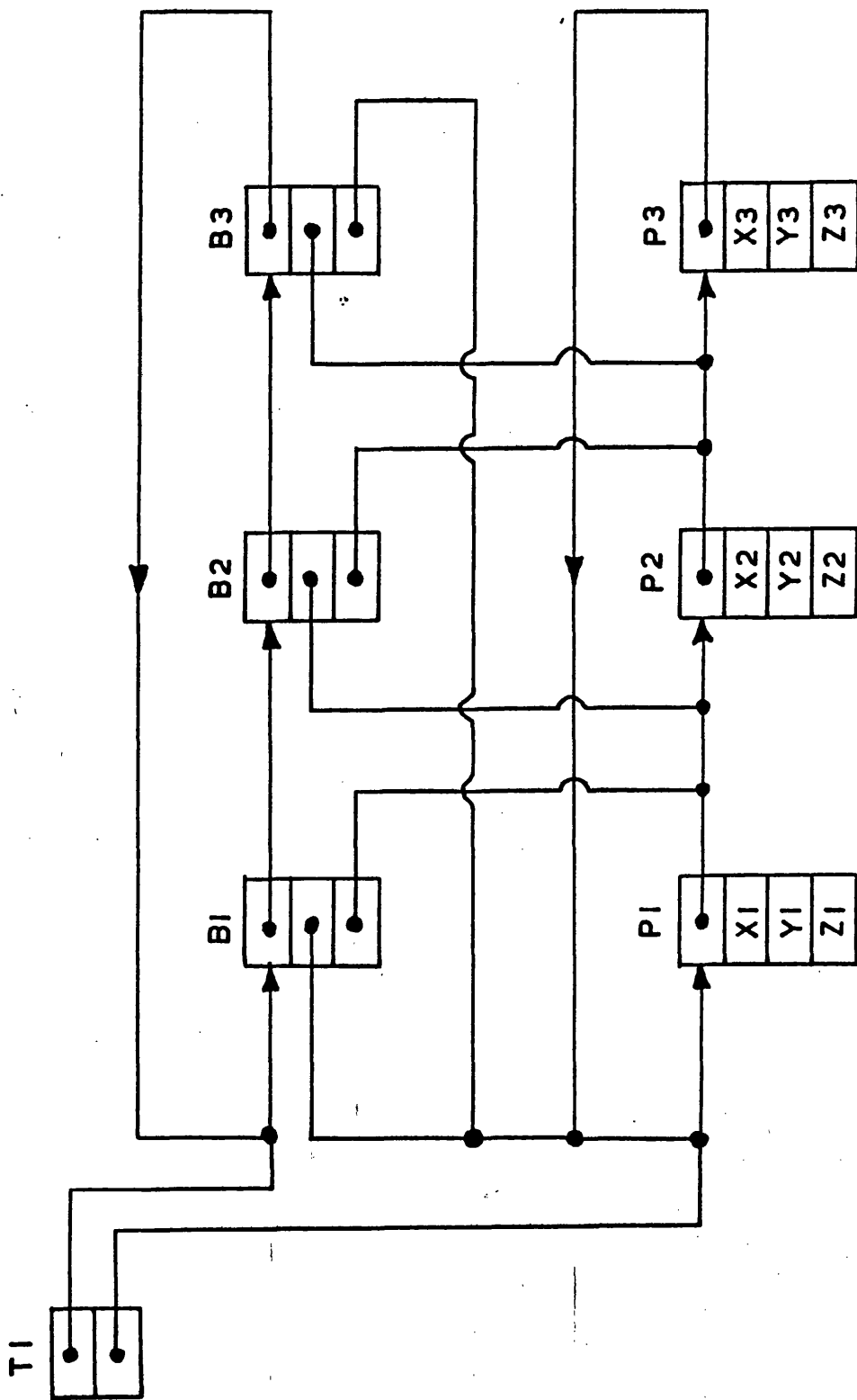


FIGURE 4.7

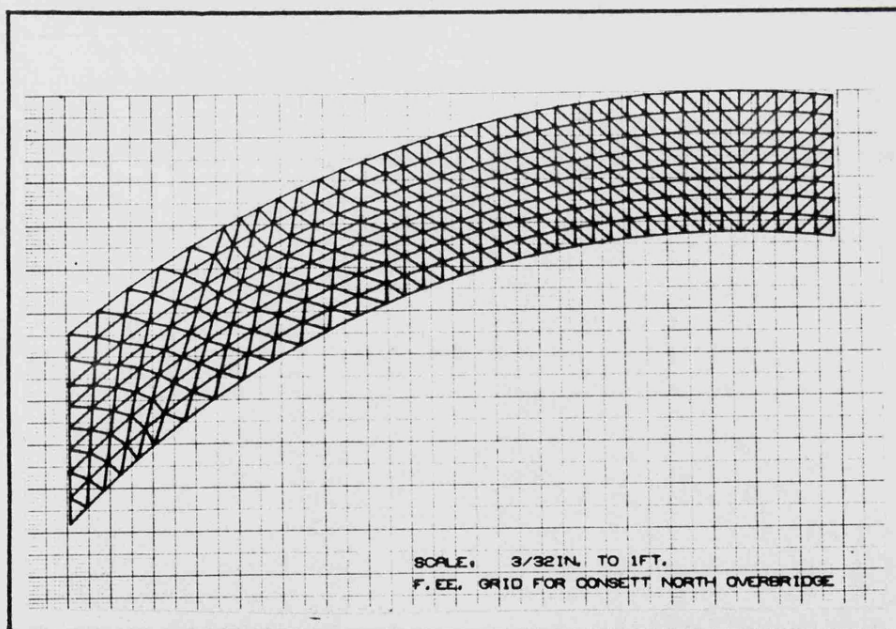
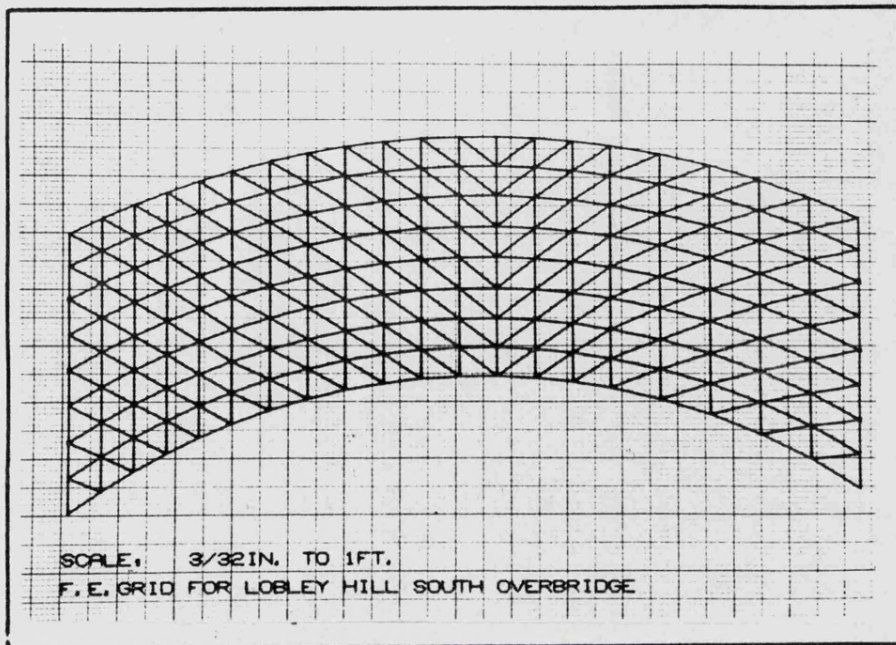


FIGURE 5.1

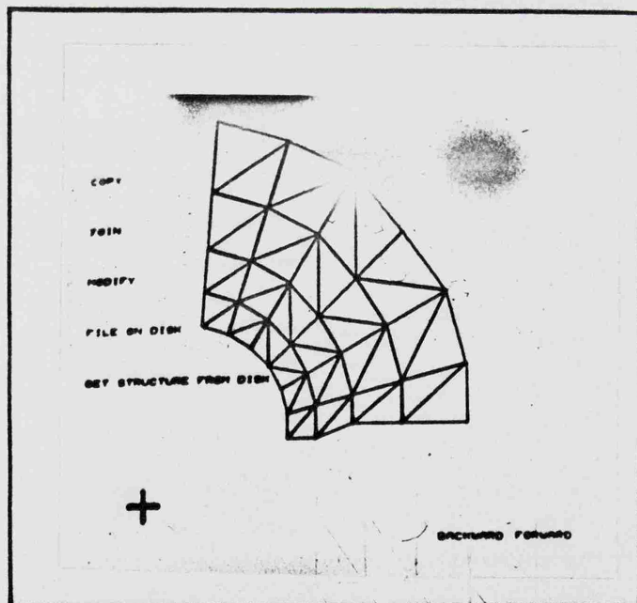
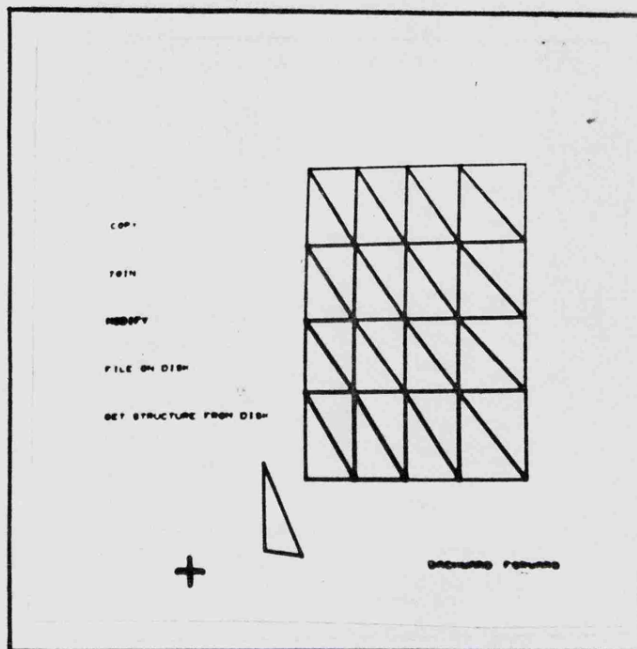


FIGURE 5.2

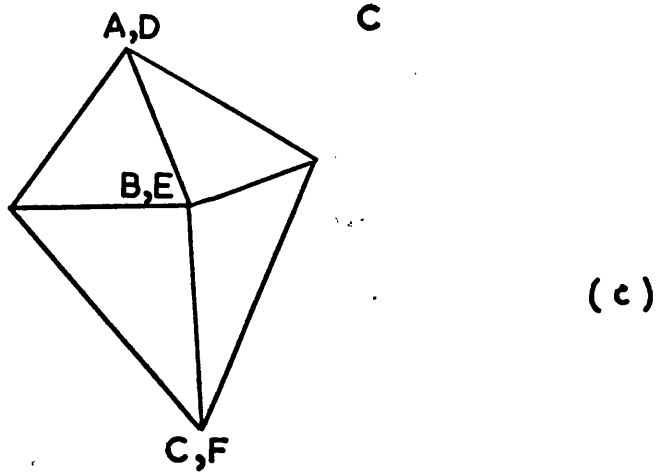
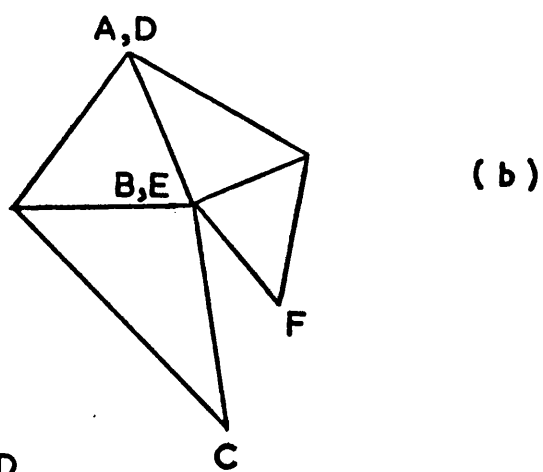
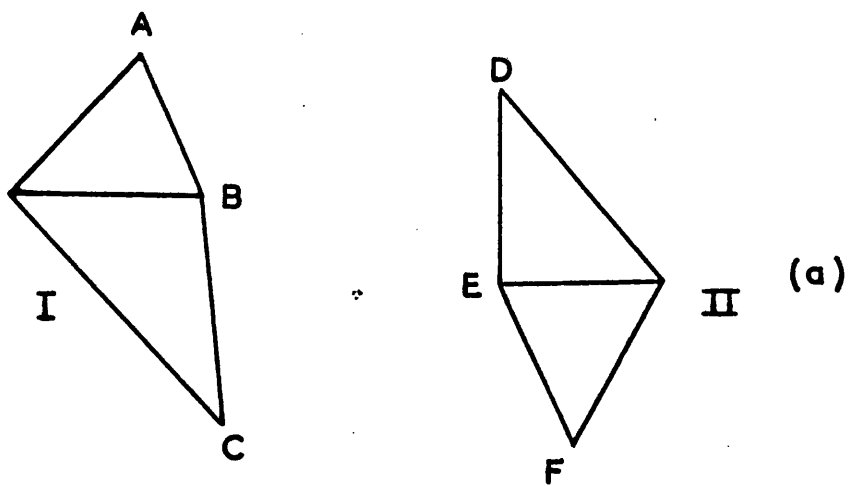


FIGURE 5.3

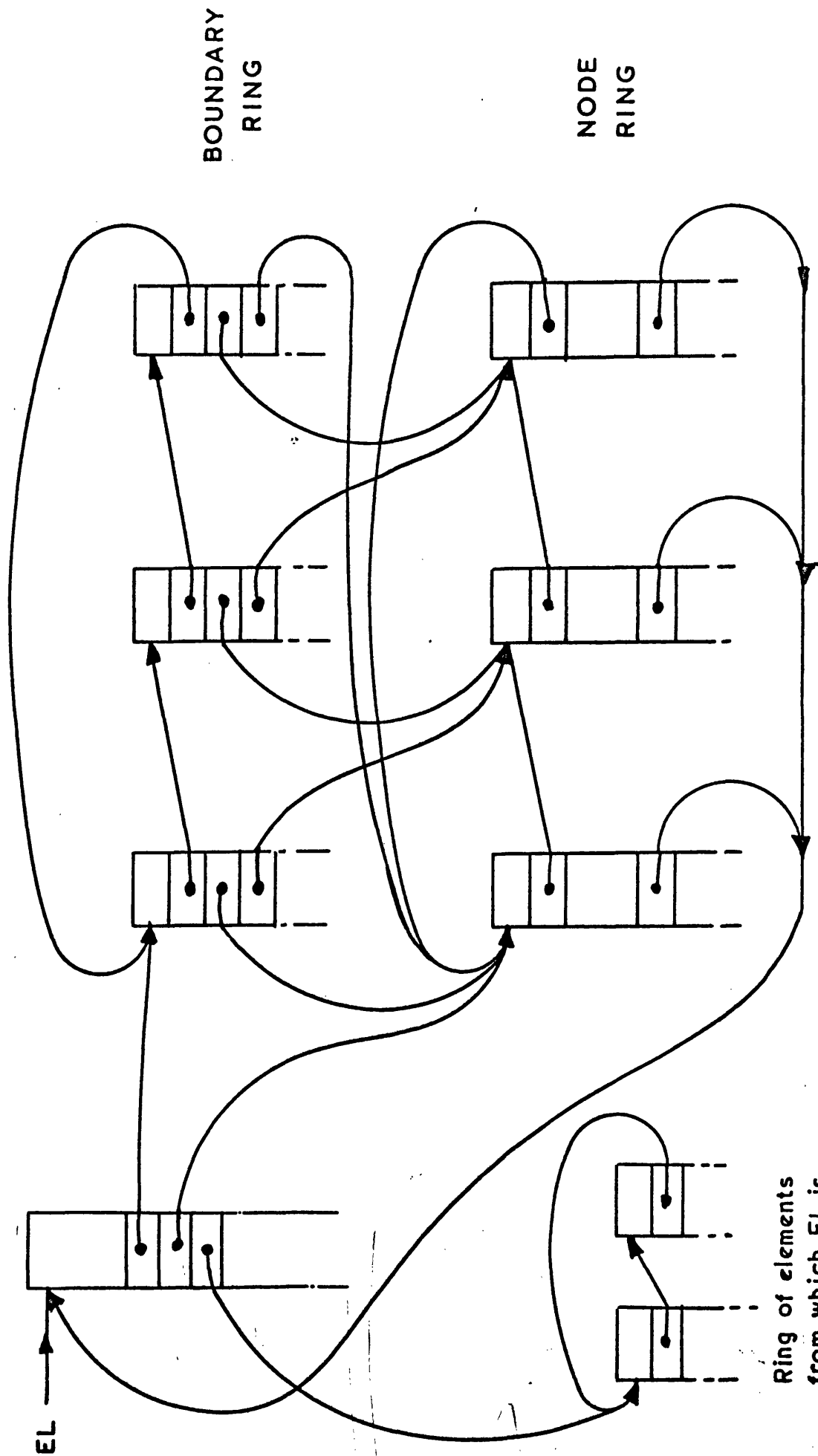


FIGURE 5.4

ELEMENT BEAD

COMPONENT	REPRESENTING
KTYPE	Type of bead
KNEXT	Pointer to next element
KTNON	Number of nodes in this element
KTNOB	Number of boundaries in this element
KTNEL	Number of elements joined
KNOD	Pointer to a ring of nodes
KBOD	Pointer to a ring of boundaries
KCOFEL	Code of first element of ring of elements
KGLX	Global x-coordinate of element's local origin
KGLY	" y " " " " "
KORIE	Global value of local axis orientation
KSCALE	Scaling factor between local and global axes
KYMOD	Young's modulus
KTNOP	Total number of parameters for this element
KINDV	Pointer to indicator vector INDV
KFINCR	" " force increment vector
KFORCE	" " Vector of forces
KDISP	" " " " displacements
KDINTV	" " displacement interpretation vector
KKMAT	" " stiffness matrix for element
KCMAT	" " transformation matrix
KTMAT	" " " "
KSTRES	" " vector of stresses
KSINTV	" " stress interpretation vector
KBSM	" " back substitution matrix

FIGURE 5.5

NODE BEAD

COMPONENT

REPRESENTING

KTYPE	Type of bead
KNEXT	Pointer to next bead on ring
KX	Local x-coordinate of node
KY	Local y-coordinate of node
KPEL	Pointer to parent element
KNPRMT	Number of parameters at this node
KPCOMP	Position in element parameter vector
KPORIE	Parameter orientation w.r.t. local axes
KSCOMP	Position in element stress vector

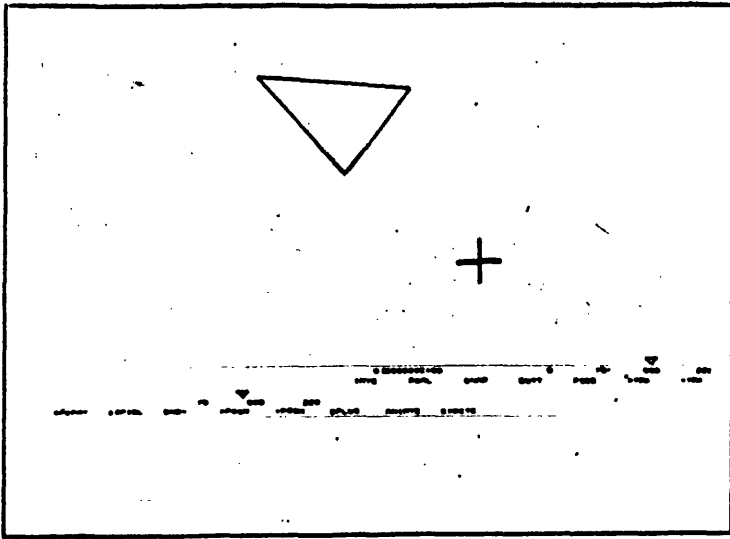
BOUNDARY BEAD

COMPONENT

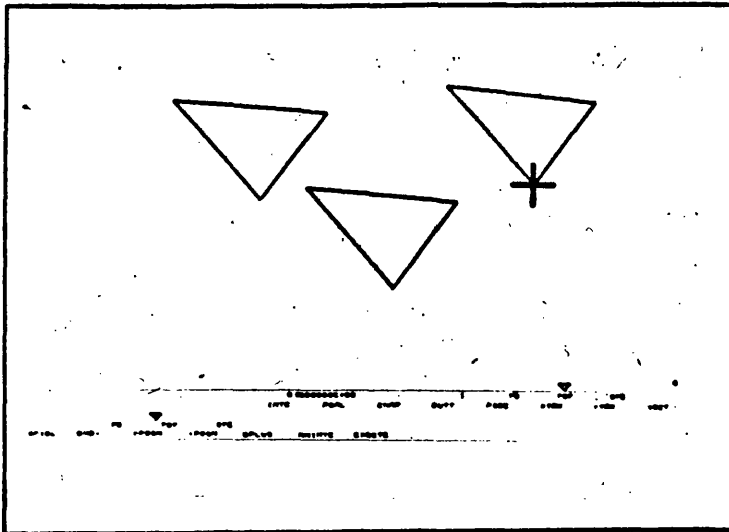
REPRESENTING

KTYPE	Type of bead
KNEXT	Pointer to next bead on ring
KBN	Pointer to back node for this boundary
KFN	" " forward " " " "
KVISIB	Visibility level for this boundary

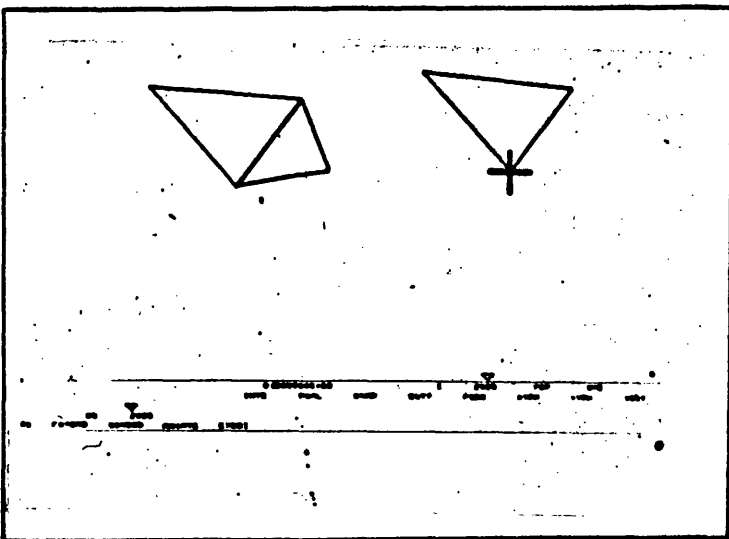
FIGURE 5.5



(a)



(b)



(c)

FIGURE 5.6

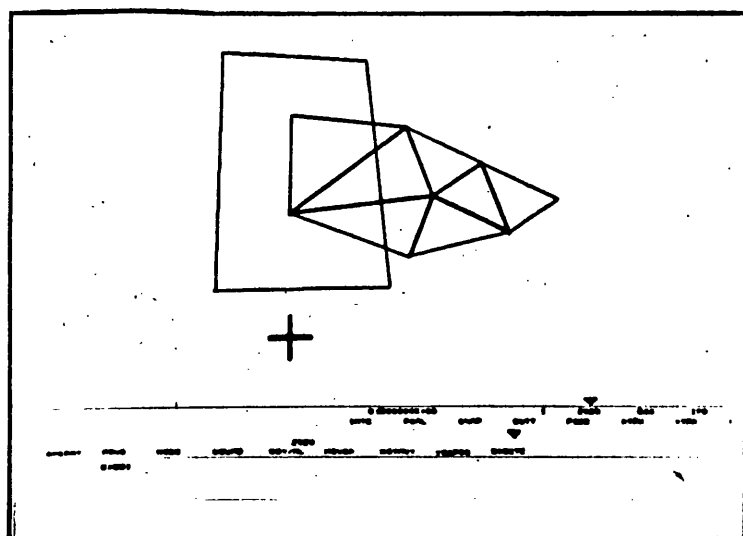
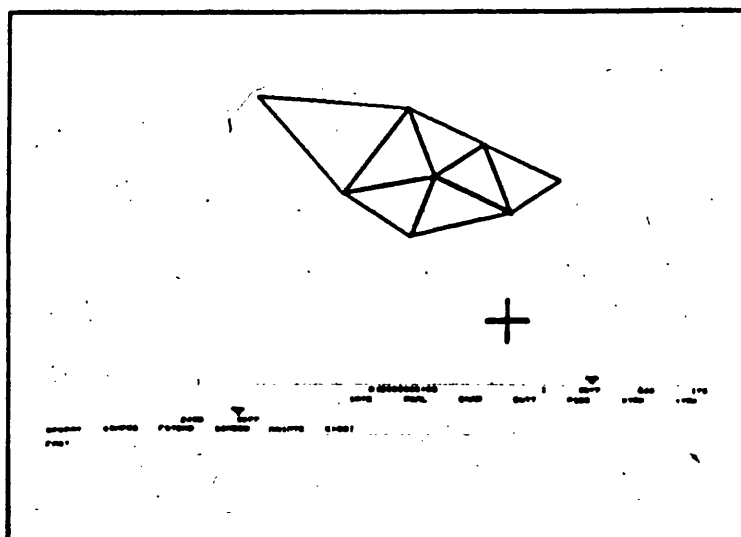
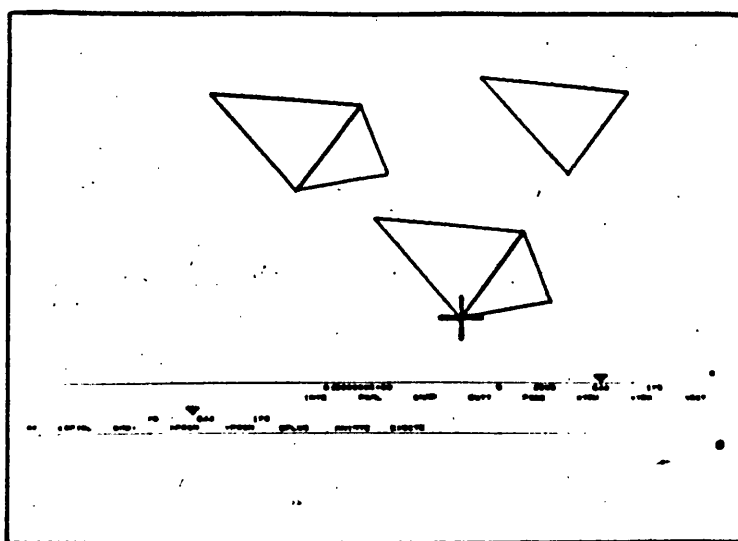
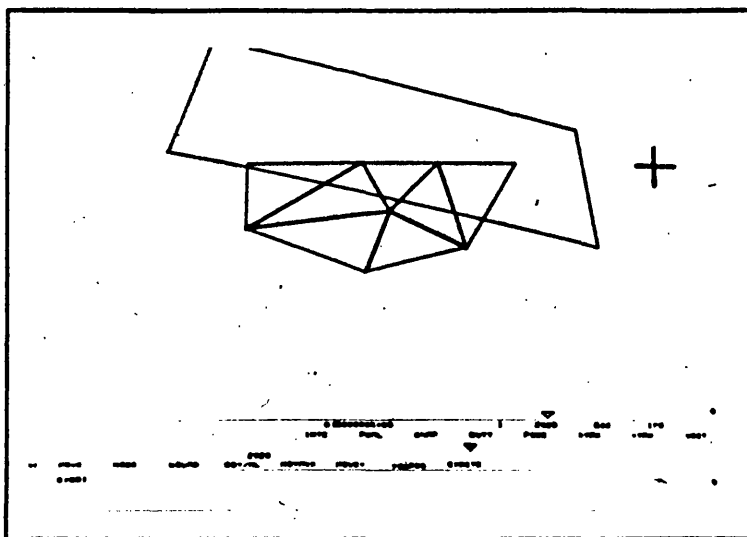
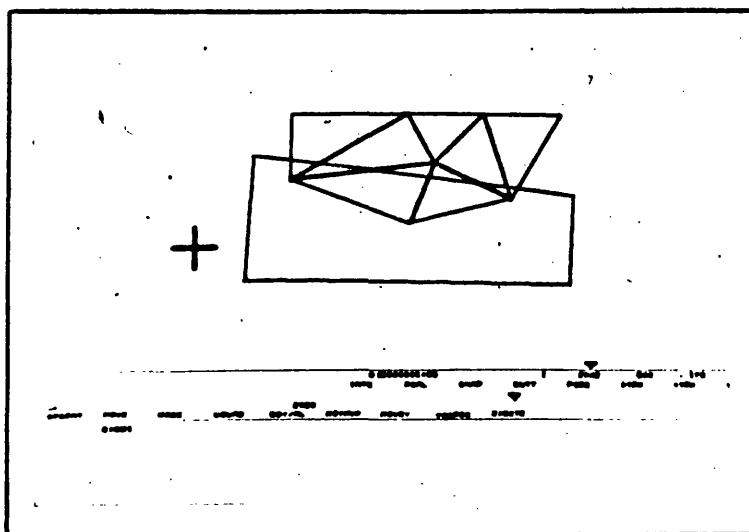


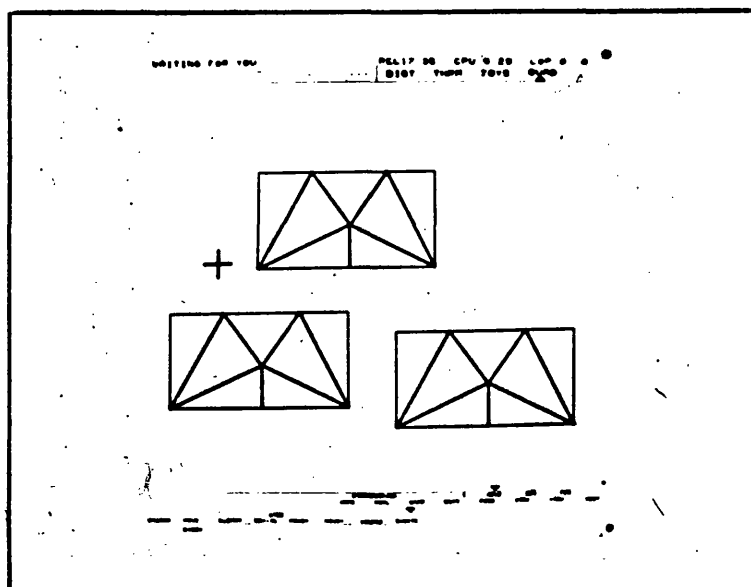
FIGURE 5.6



(g)

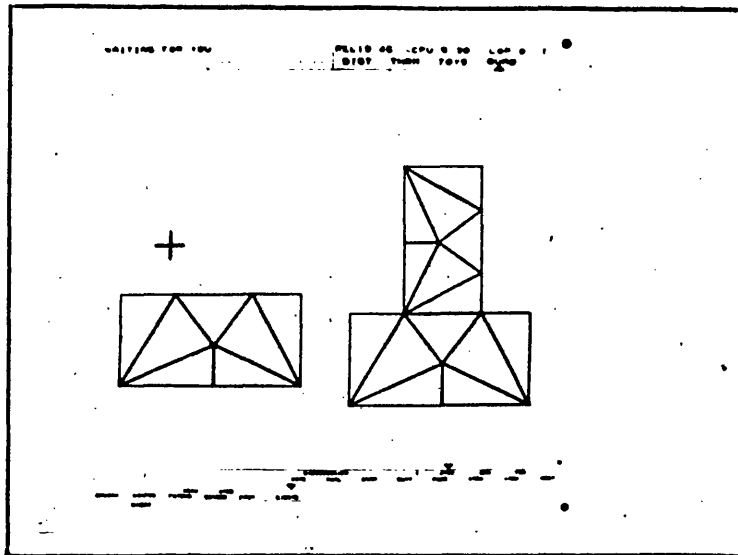


(h)

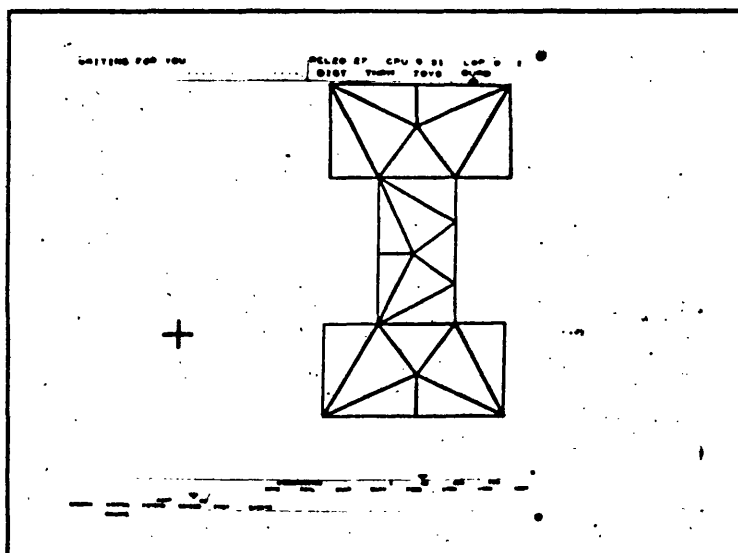


(i)

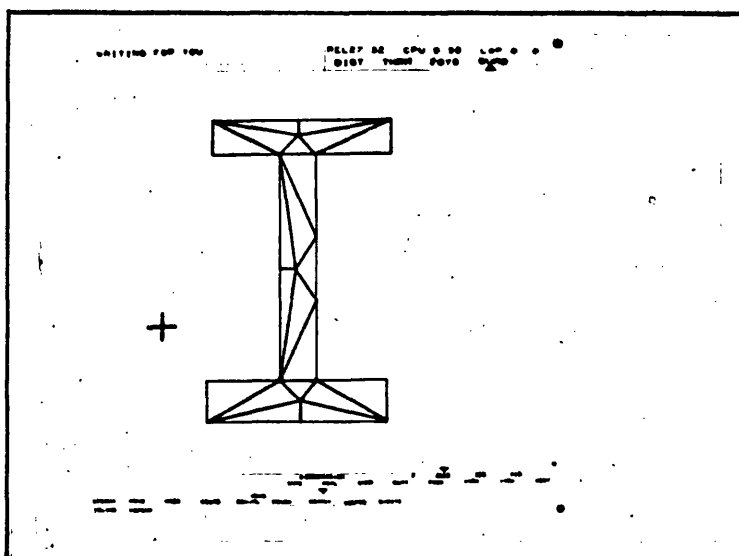
FIGURE 5.6



(j)



(k)



(l)

FIGURE 5.6

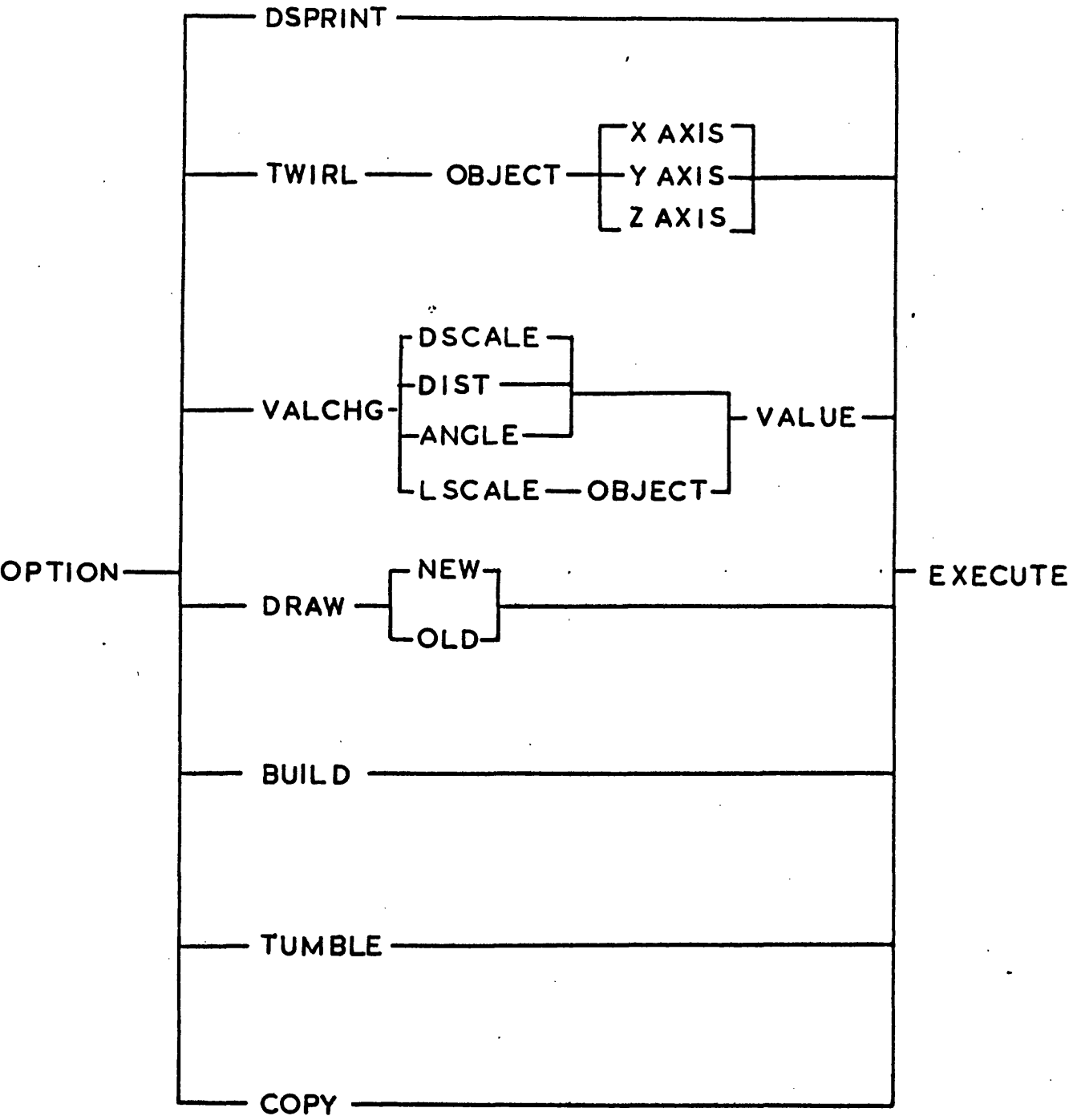
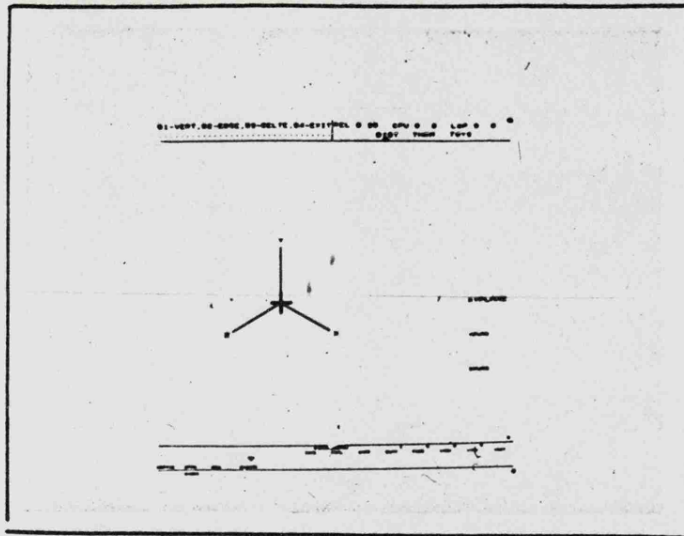
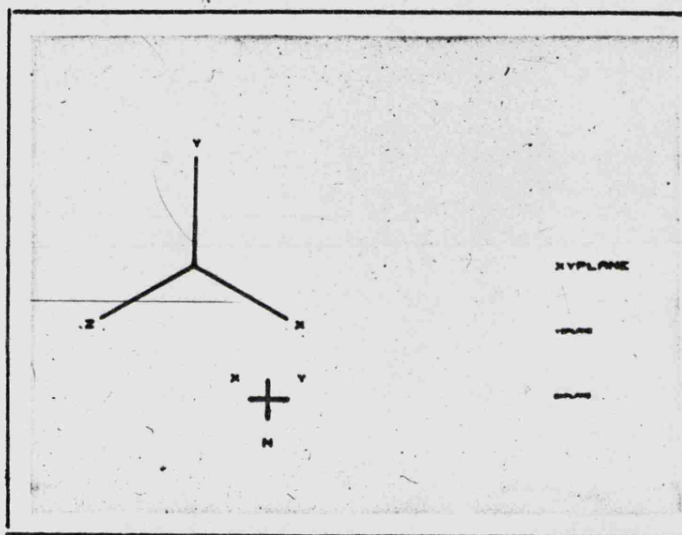


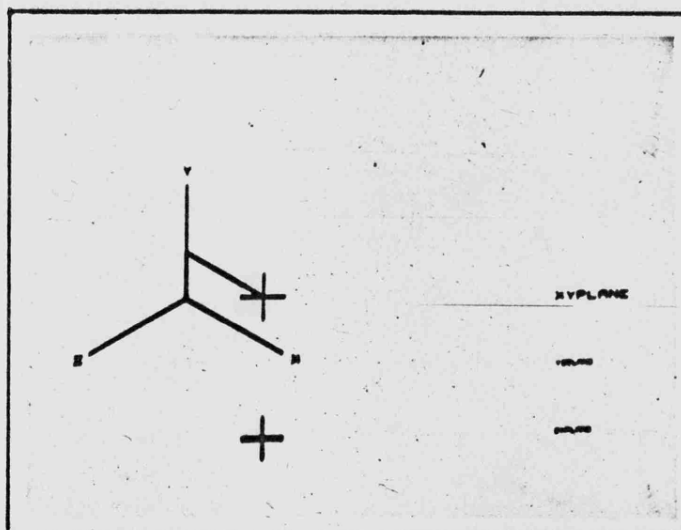
FIGURE 6.1



(a)

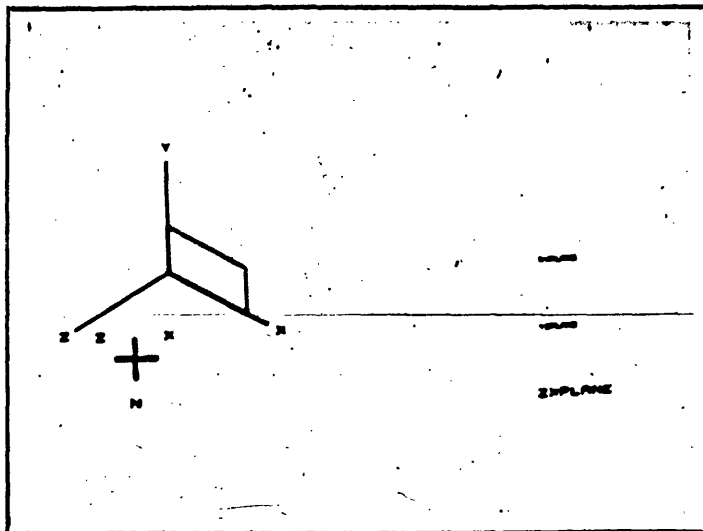


(b)

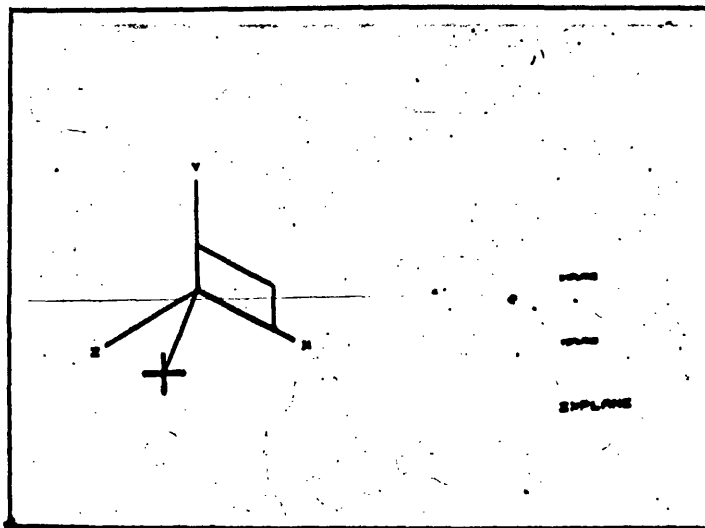


(c)

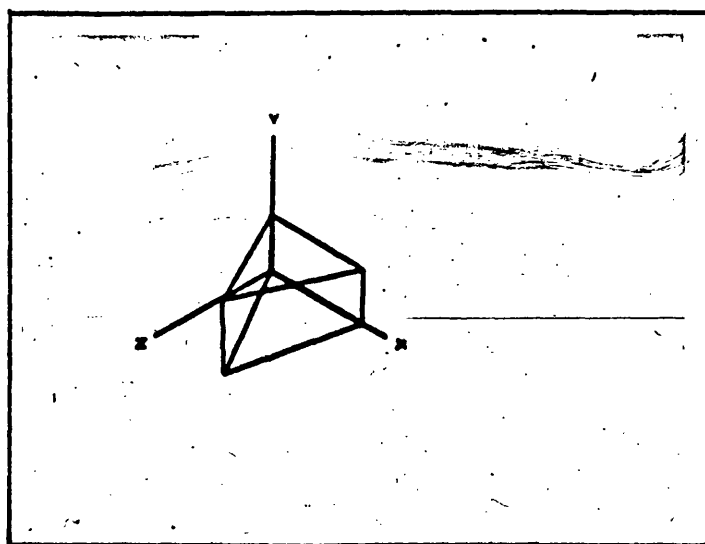
FIGURE 6.2



(d)



(e)



(f)

FIGURE 6.2

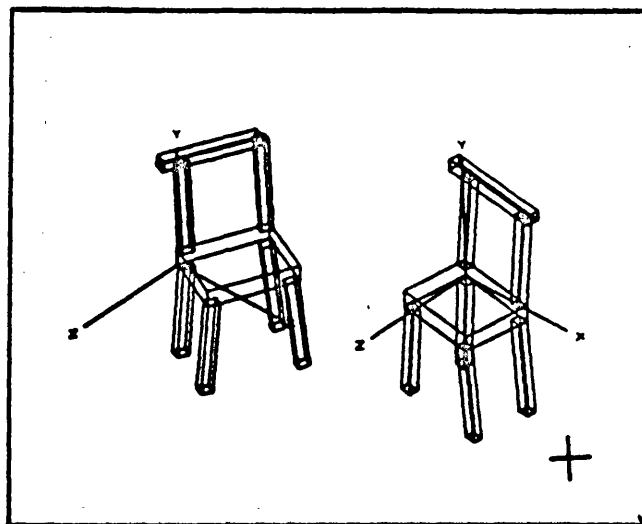
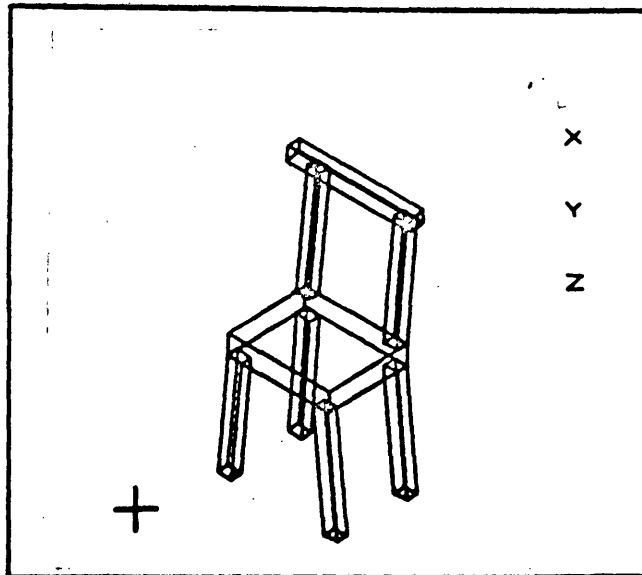
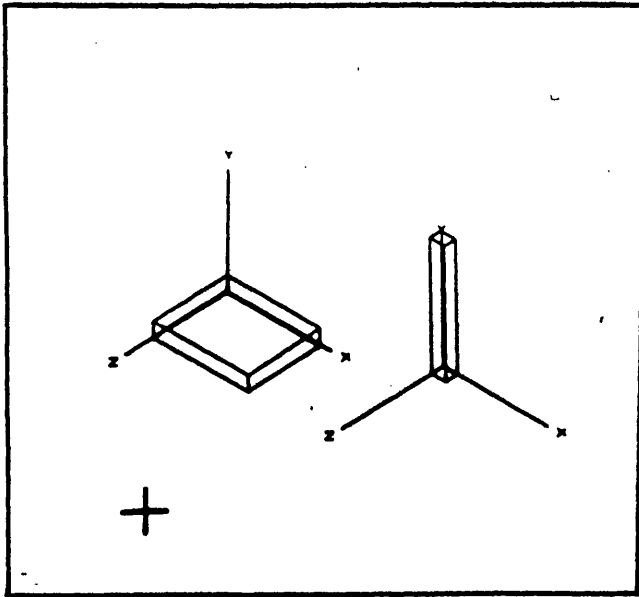


FIGURE 6.3

OBJ

NEXT	Pointer to next bead
XORIG	X } global coordinates of origin of local axes Y } Z }
YORIG	
ZORIG	
ALPHA	} Orientations of local axes w.r.t. global axes BETA } GAMMA }
BETA	
GAMMA	
SCALE	Local scale for this object
NVERTS	Number of vertices
NEDGES	Number of edges
FVERT	Pointer to ring of vertices
LVERT	Pointer to last vertex bead on ring
FEDGE	Pointer to ring of edges
LEDGE	Pointer to last bead on edge ring

VERT

NEXT	Pointer to next bead
OBJ	Pointer to object
LAST	Pointer to preceding bead on ring
X	} Local coordinates of vertex Y } Z }
Y	
Z	
GX	} Global coordinates of vertex GY } GZ }
GY	
GZ	
DX	} Display coordinates of vertex DY }
DY	

EDGE

NEXT	Pointer to next bead
OBJ	Pointer to object
LAST	Pointer to preceding bead
VERT1	Pointer to vertex at end of this edge
VERT2	Pointer to vertex at other end of this edge

SCENE

NOBJ	Number of objects
FOBJ	Pointer to ring of objects
LOBJ	Pointer to last object bead on ring

FIGURE 6.4

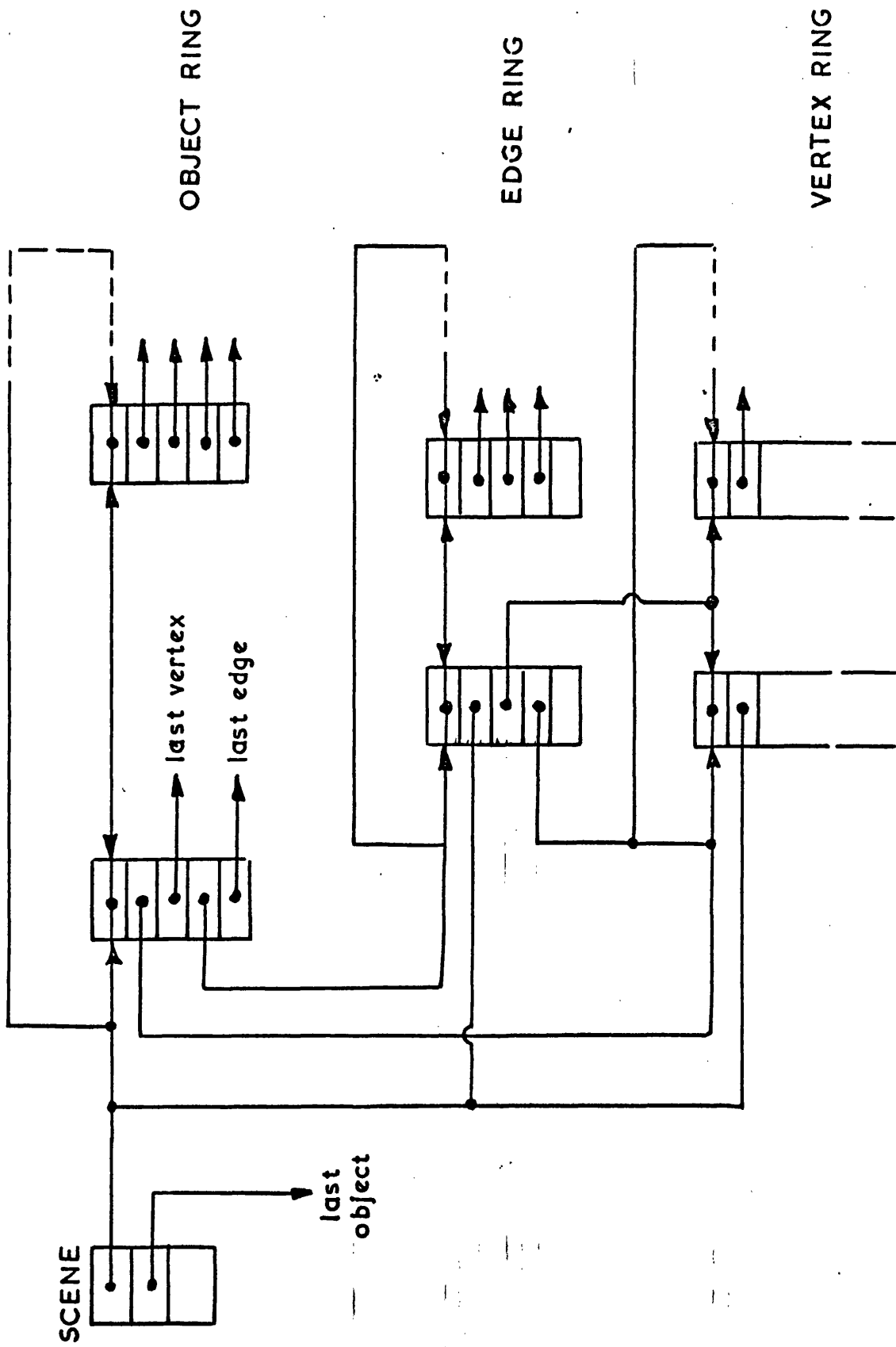


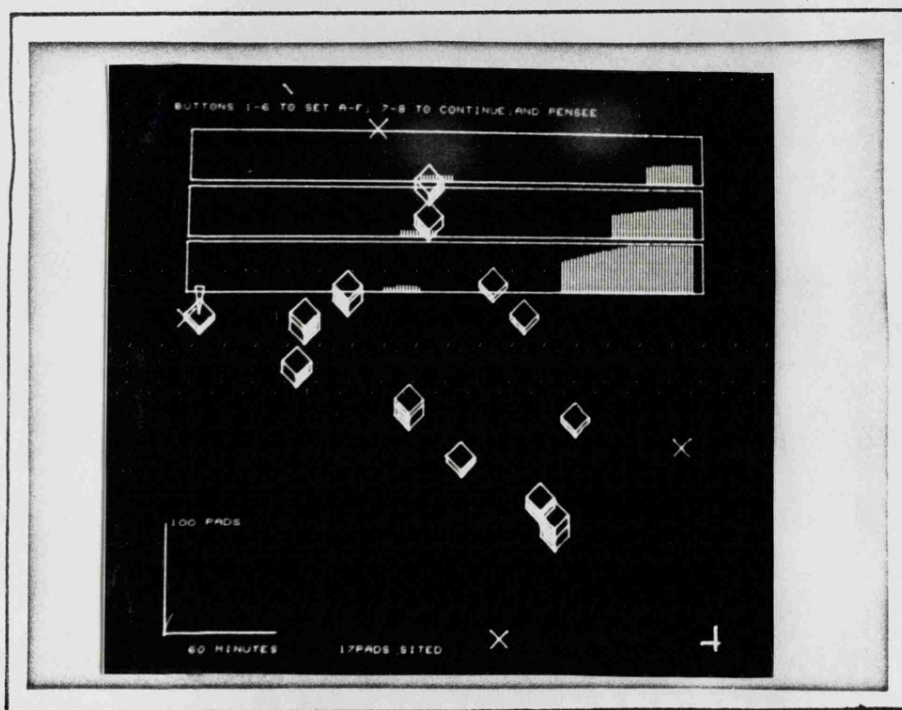
FIGURE 6.5

COMPONENT

REPRESENTING

KPADAL	Pointer to next bead on list of all pads
KPADGR	" " " " " " " ground "
KPADRF	" " " " " " " roof "
KBNTHP	Number of floors below this pad
KORENT	Orientation of pad
KSTOYS	Number of storeys (flat or maisonette)
KCHECK	Code used by checking algorithms
KX	X-coordinate of pad
KY	Y " " "
KZ	Z-coordinate of pad (absolute)
KSKYLN	Pointer to skyline data for pad (if a ground pad)

FIGURE 7.1



This figure shows an axonometric display of a housing site developing under control of the BAID program. Three skyline views are displayed representing three viewpoints from the window of one dwelling.

FIGURE.7 2

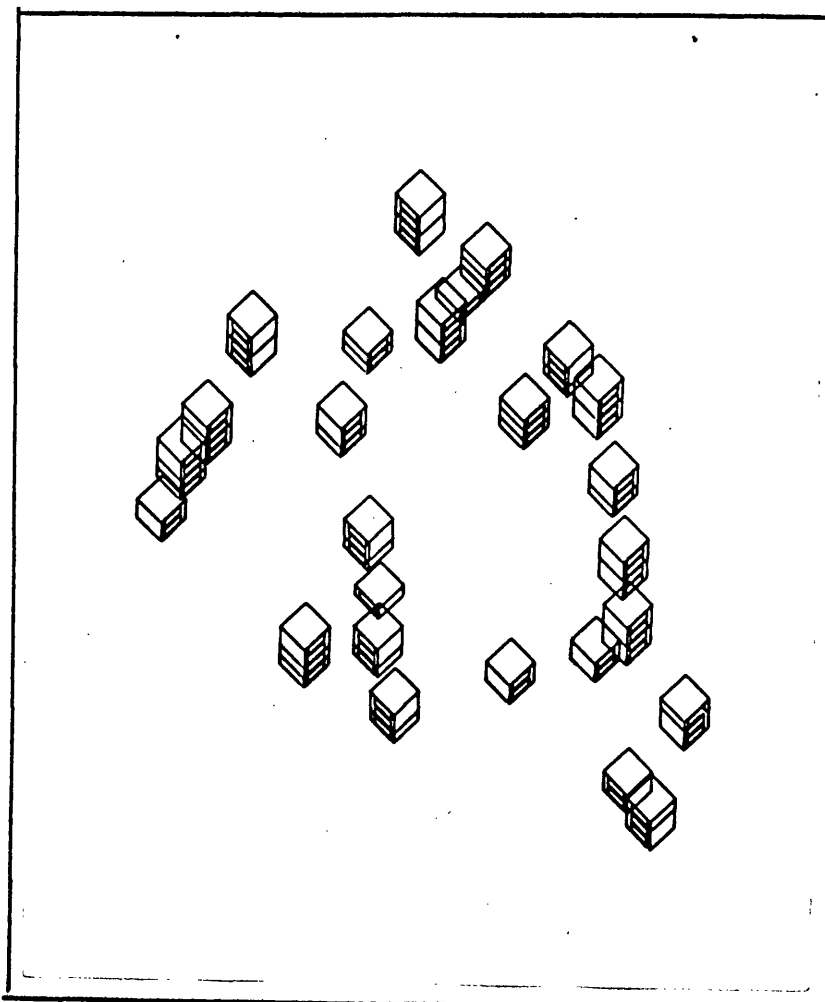
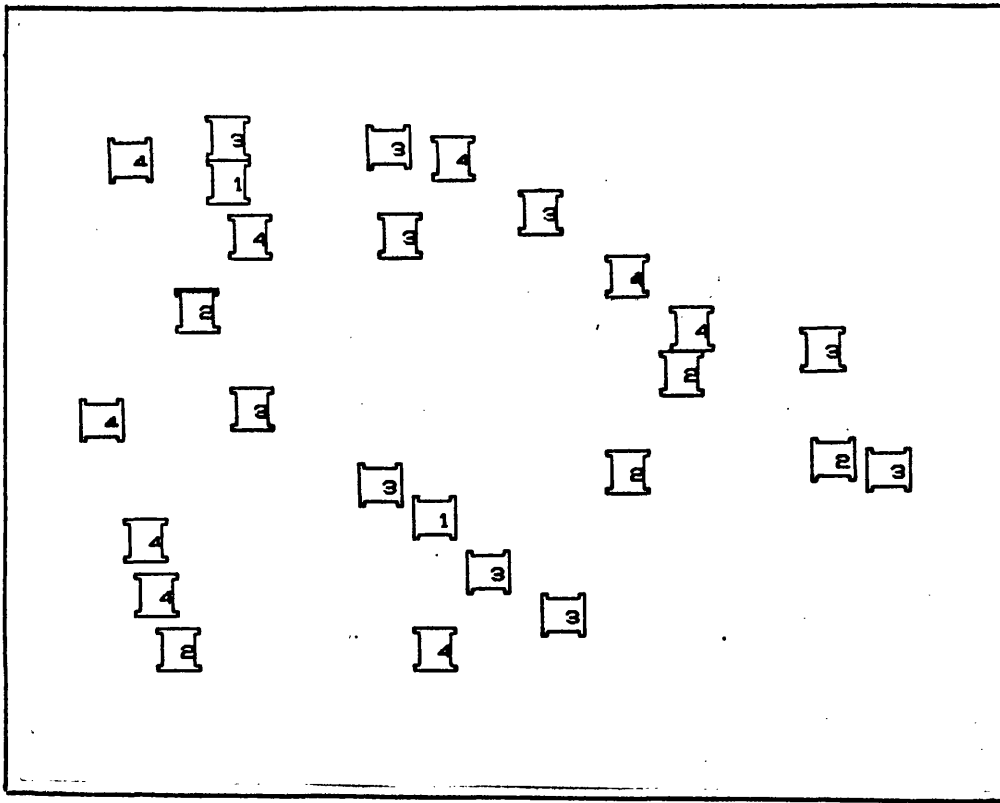


FIGURE 7.3

PAGE SIZE (WORDS)	TIME TO SWAP PAGE (MSECS)	TIME TO SWAP A 5K SUBSTRUCTURE (SECS)
64	425	34
128	425	17
256	425	8.5
640	425	3.4
1K	450	2.25
2K	550	1.65
3K	550	1.1
5K	600	0.6

FIGURE 8.1

TIME TO SWAP A PAGE AND

TIME TO SWAP A 5K SUBSTRUCTURE

Time (seconds)

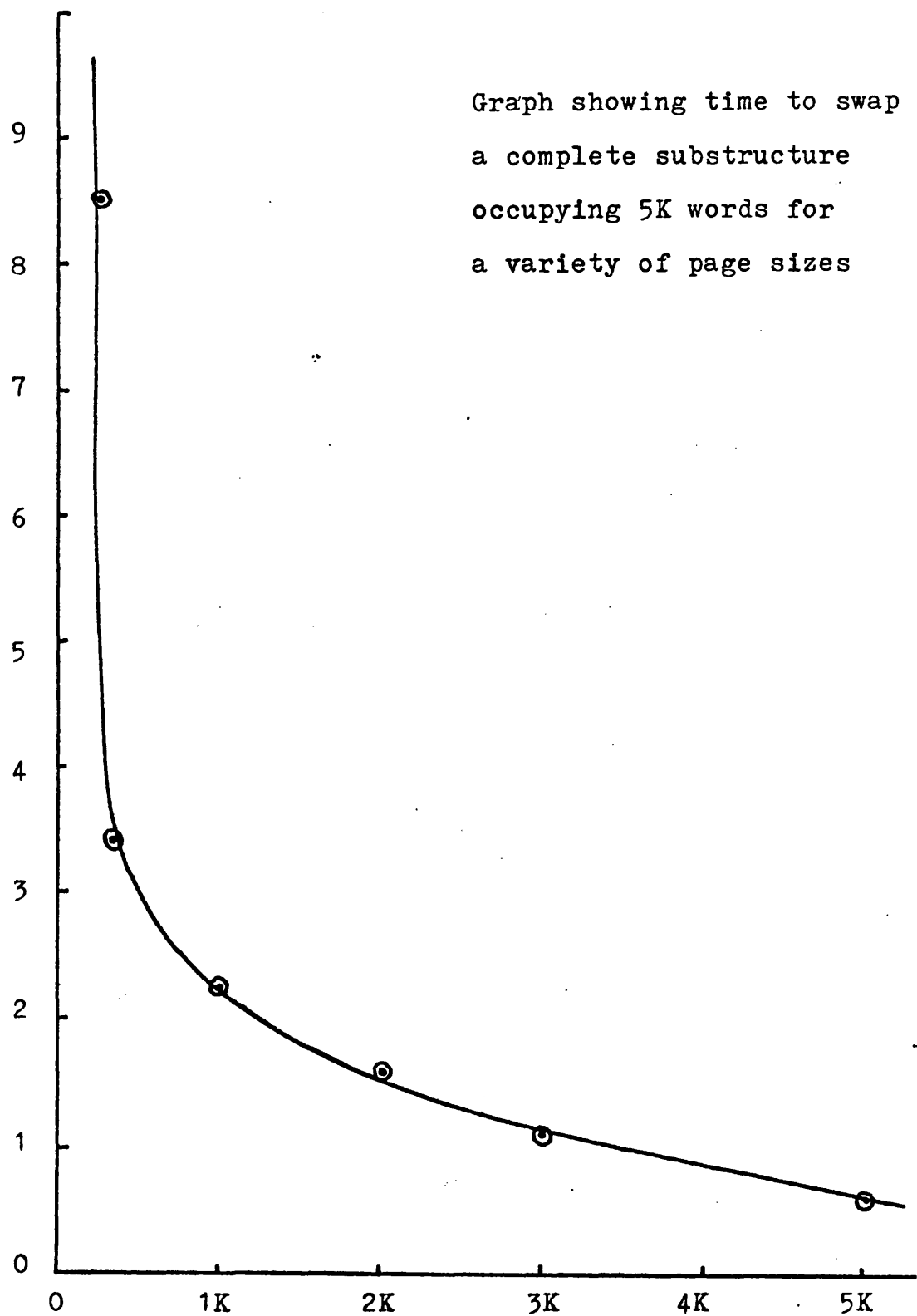


FIG 8.2

Page size (words)

Pointer to next bead		ELEMENT BEAD
Bead type	Display status	
Number of nodes	Number of elements	
Pointer to nodes		
Pointer to boundaries		
Pointer to transformation matrix		
Pointer to stiffness matrix		

Pointer to next bead		NODE BEAD
Bead type	Global node number	
Local X coordinate		
Local Y coordinate		
Local Z coordinate		
Pointer to parent element		
Pointer on linked node list		

Pointer to next bead		BOUNDARY BEAD
Type of bead		
Pointer to end of boundary		
Pointer to other end of boundary		

FIGURE 8.3 DEFINITIONS OF BEAD CONTENTS

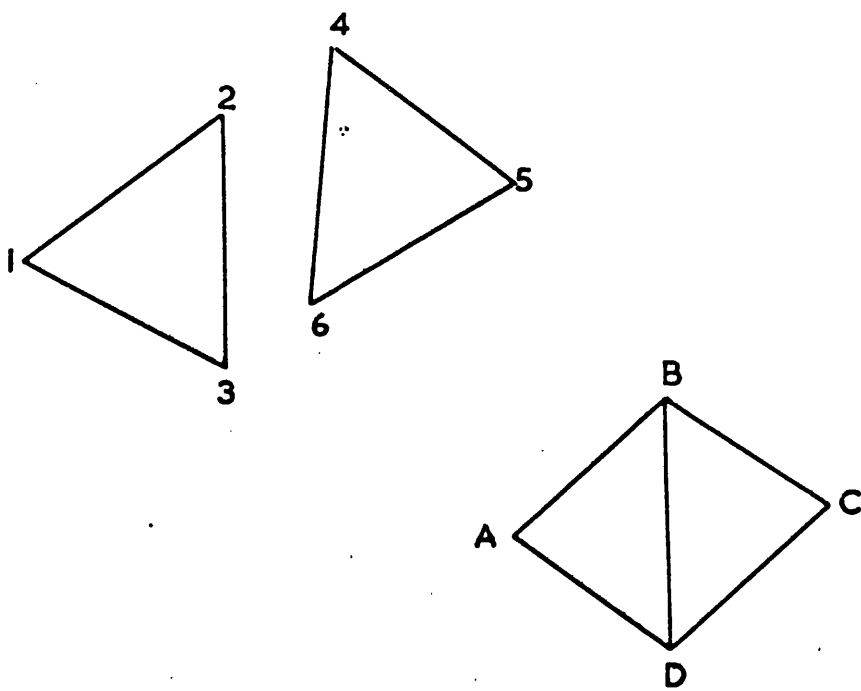


FIGURE B.I

APPENDIX A

Matrix algebra for LUISA system

The general matrix equation relating the vector of nodal force parameters $\{f\}$ to nodal displacement parameters $\{\delta\}$ is:

$$\{f\} = [K] \{\delta\} \quad (1)$$

If forces f_1 are specified and f_o are those remaining unfixed the matrices may be partitioned to give

$$\begin{Bmatrix} f_o \\ f_1 \end{Bmatrix} = \begin{bmatrix} K_{oo} & K_{o1} \\ K_{1o} & K_{11} \end{bmatrix} \begin{Bmatrix} \delta_o \\ \delta_1 \end{Bmatrix} \quad (2)$$

From which $\delta_1 = K_{11}^{-1} f_1 - K_{11}^{-1} K_{1o} \delta_o$

$$= \Delta\delta_1 - T \delta_o \quad (3)$$

and $f_o = (K_{oo} - K_{o1} K_{11}^{-1} K_{1o}) \delta_o$

$$+ K_{o1} K_{11}^{-1} f_1$$

$$= K^1 \delta_o + \Delta f_1 \quad (4)$$

The first term in equation (3), $\Delta\delta_1$, represents the effects on displacements at those nodes where forces f_1 are applied, due to their application. The second term, $T\delta_o$, expresses the relationship between the displacements δ_1 and the displacements

at free nodes where neither force or displacement has been fixed. Equation (3) is the displacement-force relationship for the fixed nodes.

Equation (4) is the force-displacement relationship for the nodes remaining free. The second term, Δf_1 , represents the force increments stored in the force increment vector FINCR.

The connection matrix referred to relates the parameters at one level of assembly to those at the level below. A simple connection matrix to represent one-dimensional displacements for the assembly shown in Figure B.1 would be

$$\begin{pmatrix} \delta_1 \\ \delta_2 \\ \delta_3 \\ \delta_4 \\ \delta_5 \\ \delta_6 \end{pmatrix} = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \begin{pmatrix} \delta_A \\ \delta_B \\ \delta_C \\ \delta_D \end{pmatrix}$$

APPENDIX B

Basic software, and display organisation

In order to implement the system in Fortran a package of programs known as FRED (Fortran Routines for the Elliott Display) was used. The FRED package includes all the necessary routines for the three major aspects of display usage: basic administration operations, code generation routines for creating pictures, and control routines. It should be emphasised that these routines are for basic operations only, and interactive aspects of an application system are programmed using the FRED package as a tool.

Objects are drawn on the screen by creating code in a buffer area, then inserting this code into the display file, which is an area of core store devoted to the display. The code contains formatted information about the picture to be drawn and is decoded and converted to analogue form by the display controller. The display file is scanned, and the picture redrawn, ten times per second. A block of code is termed an item, and has an identifying number to allow subsequent referencing. It is usual practice for items to correspond to complete picture elements, but the definition of a picture element rests with the programmer and is dependent on the application. Using the control routines, the machine can be programmed to await some action on the part of the user, including pressing of function keys and indicating items on the screen with the lightpen. If an item is seen by the lightpen (the lightpen having been enabled for this purpose), a

program returns the identifying number of the item. This can then be used to perform display file editing or to access other data, associated with the item, stored in the application data structure.

Three Fortran subprograms are used to perform data structuring operations and the administration of free storage zones. The three routines enable various free storage zones to be defined, areas to be assigned within these zones for storing data (i.e. allocation of beads), and the returning of these areas to free store. The routines were provided by I.C.L. and are called OPEN, IALLOC, RETURN.

REFERENCES

- (1) Ross, D.T., "The A.E.D. Approach to Generalized Computer Aided Design", Proceedings A.C.M. National Meeting, 1967.
- (2) Leckie, F.A. and M.P. Ranaweera, "Interactive Use of the Lightpen in Finite Element Limit Load Analysis", Proceedings of Computer Graphics 70 Symposium, Brunel University, April 1970.
- (3) Hubbard, R.J., "A Preliminary Report on the Automatic Input and Output of Data for a Triangular Plate Bending Finite Element Program Used for Analysing Bridgedecks", University of Leicester Engineering Department Report 68-13.
- (4) Greator, F.S., Jr., and D. Cohen, "Producing Dynamic Perspective Views for Vehicle Simulation", Data Processing Magazine, April 1968.
- (5) Sutherland, I.E., "A Head Mounted Three Dimensional Display", A.F.I.P.S. Conference Proceedings, Volume 33 Part 1, 1968 Fall Joint Computer Conference. Thompson Book Co.
- (6) Sutherland, I.E. and R.F. Sproull, "A Clipping Divider", A.F.I.P.S. Conference Proceedings, Volume 33 Part 1, 1968 Fall Joint Computer Conference. Thompson Book Co.
- (7) Sutherland, I.E., "Computer Displays", Scientific American, June 1970.
- (8) Forrest, A.R., "A Survey of Techniques for Removing Hidden Lines", C.A.D. Group Document No. 19. Cambridge University Mathematical Laboratory.

- (9) Feeser, L.J. and J.D. Cutrell, "Perspective Views and Computer Animation in Highway Engineering", Proceedings of Computer Graphics 70 Symposium. Brunel University, April 1970.
- (10) Gray, J.C., "Compound Data Structure for Computer Aided Design: A Survey", Proceedings of 22nd National Conference, Association for Computing Machinery. Thompson Book Co., 1967.
- (11) Gray, J.C., "ASP Programming Manual", C.A.D. Group Document No. 15, Cambridge University Mathematical Laboratory.
- (12) Johnson, T.E. "A Mass Storage Relational Data Structure For Computer Graphics and other Arbitrary Data Stores", Department of Architecture Report, M.I.T., October 1967.
- (13) Programmers Guide to Genesys.
- (14) Roos, D., ICES System Design. M.I.T. Press, 1966.
- (15) Foster, J.M., List Processing. MacDonald/Elsevier, 1967.
- (16) Butlin, G.A. and R.J. Hubbard, "A Scheme for Man-Machine Interactive Structural Analysis". Proceedings I.E.E. International Conference on Computer Aided Design, April 1969. I.E.E. Conference Publication No. 51.
- (17) Young, A.G., "Structural Design Using Interactive Graphics". Proceedings of Computer Graphics 70 Symposium. Brunel University, April 1970.
- (18) Hubbard, R.J., "TDD - An Interactive Three Dimensional Drawing Program for Graphical Display and Lightpen". Proceedings of Computer Graphics 70 Symposium. Brunel University, April 1970.

- (19) Wiseman, N.E. et.al. "PIXIE - A New Approach to Graphical Man-Machine Communication". Proceedings I.E.E. International Conference on Computer Aided Design, April 1969. I.E.E. Conference Publication No. 51.
- (20) Butlin, G.A., "A Fortran Package for Interactive Graphics". Proceedings of Computer Graphics 70 Symposium. Brunel University, April 1970.
- (21) Sutherland, I.E. "Sketchpad: A Man-Machine Graphical Communication System". A.F.I.P.S. Conference Proceedings, 1963 Spring Joint Computer Conference.
- (22) Johnson, T.E., "Sketchpad III: A Computer Program for Drawing in Three Dimensions". A.F.I.P.S. Conference Proceedings, 1963 Spring Joint Computer Conference.
- (23) Butlin, G.A., R.J. Hubbard, C.K. Grafton, Fortran Programs for the 4280 Display. Leicester University Engineering Department Report No. 70-26.
- (24) AED-O Programmer's Guide. Electronic Systems Laboratory, Massachusetts Institute of Technology.
- (25) Lang, C.A. et. al. GINO - Graphical Input/Output. Cambridge University C.A.D. Group.
- (26) I.I.T. Research Institute. APT Part Programming. McGraw-Hill, 1967.
- (27) Throsby, P.W., "A Finite Element Approach to Surface Definition". Computer Journal Vol. 12 p.385, 1969.

- (28) Kuehner, C.J., and B. Randell: "Demand paging in perspective",
AFIPS Fall Joint Computer Conference proceedings 1968.
- (29) Joseph, M.: "An analysis of paging and program behaviour",
The Computer Journal, Vol. 13. Number 1, February 1970.
- (30) Bobrow, D.G. and Murphy, D.I.: "Structure of a LISP system
using two-level storage", Communications of the A.C.M. Vol.10.,
number 3, March 1967.
- (31) Cohen, J.: "A use of fast and slow memories in list-processing
languages", Communications of the A.C.M. Vol.10., number 2,
February 1967.
- (32) Wilkes, M.V.: Time Sharing Computer Systems. MacDonald/Elsevier
Computer Monographs No.5. 1968.
- (33) Coons, S.A.: "Surfaces for Computer-Aided Design of Space Forms",
MAC-TR-41. MIT. 1967.
- (34) Armit, A.P.: "A multipatch design system for Coons' patches."
I.E.E. Conference on Computer-Aided Design, Southampton, 1969.
- (35) Zienkiewicz, O.C., and Cheung, Y.K.: "The Finite Element Method
in Structural and Continuum Mechanics". McGraw Hill Publishing Co.

TECHNIQUES FOR INTERACTIVE COMPUTER GRAPHICSPh.D. THESIS SUMMARY - R.J. HUBBOLD

The work presented is concerned with investigations into the use of a refreshable graphical display for the solution of design and analysis problems in engineering. Effort has been devoted to the study and implementation of a variety of applications in an attempt to identify the most suitable techniques, form of program organization and hardware configuration for this type of equipment. General topics pertinent to these investigations, including data structuring, graphical communication and some general principles of software design, are discussed.

The applications which are presented are:

- (a) LUISA, a system for finite element analysis of two-dimensional engineering structures,
- (b) TDD, a set of programs for three-dimensional drawing, written to investigate a number of methods of communicating with a three-dimensional model,
- (c) BAID, a program for aiding the architect with the design of high density housing layouts,
- (d) An outline of a system for generating the data input for three-dimensional finite element analysis of solid and shell structures. A detailed description is included of a paging scheme used to segment the data structure.

The data structure employed and scope of facilities provided are described for each of the applications. Discrete representation of engineering components is shown to be ideally suited to the organization of data in structured form both in the computer core store and on secondary storage.

A description is given of some devices, provided by software and making use of the lightpen, which allow development of dynamic

techniques for graphical communication with a digital model. A modular approach to software design is advocated, with advantage being taken of general packages, wherever possible, for administration of interaction handling and data organization.

Proposals are made about the re-arrangement of the applications dealt with in order to implement them on a remote satellite computer configuration. Suggestions are made about the size of such a configuration and the organization of software in the two machines.

March 1971