A single data-display structure: A new view on interactive computer graphics in CAD

N. Marovac

Department of Electrical Engineering, Imperial College of Science and Technology, Exhibition Road, London, SW7 2BT

The data-display structure (DDS) is a single structure developed for a suite of programs for general network analysis using computer interactive graphics. Its purpose is to describe a network built on the screen in full pictorial, constitutive and topological details, and thus to enable the display of the network and the building of various network matrices required by different network analysis programs.

Because of the specific topology of networks, there is little extra information needed for network analysis which is not already present for the purpose of graphical (pictorial) description of the network, i.e. for its display.

This fact is used as the basic idea in the development of this structure.

(Received June 1971)

1. Introduction

This paper reviews the basic concepts of display files and data and display structures with particular reference to the design of an interactive graphical system for general network design and analysis. A description is given of the considerations which led to the development of a new single display-data structure for this general application.

The new structure was implemented in a program for electrical network analysis on a configuration consisting of a DEC PDP-9 computer and a DEC 340 display. At present it is being re-implemented in a software system for electrical network analysis on a configuration comprising a DEC PDP-15 computer and a VT 15 display.

The paper is divided into five sections. Section 2 sets out basic concepts and gives definitions and explanations of terms used in the further text. Illustrations are given in Section 3 of ways in which basic concepts of display files and data and display structures have been applied by previous workers, with an analysis of each application. In Section 4 a general specification is derived for a version of an efficient and optimal data and display structure to be used in a software system for general network analysis by interactive computer graphics.

The single data-display structure designed and implemented on the basis of this specification is presented in Section 5.

2. Basic concepts and definitions

Programs for computer aided design using interactive computer graphics apply a number of concepts closely related to these types of programs.

These concepts will first be described in the sense in which they will be referred to in subsequent sections. Terms which are defined in the text are in italics when mentioned for the first time.

The term *picture* is used to describe a graphical representation of a network, or a system of networks, in short a *system*, the description of which has been built inside the computer, up to that particular moment. This description is to be used either to analyse the system or to apply to it any of the used design techniques.

The part of the picture which is currently displayed on the screen (windowed) will be called a *displayed picture*. As an example, let us consider a large network, the description of which has been generated inside the computer by drawing it on the screen. If the network is reasonably large, only a part of it can be currently displayed on the screen (the displayed picture) but the description of the complete up-to-date network is kept in the computer memory, which includes the information necessary to obtain a pictorial (graphical) representation of the network as a whole (the picture).

An *object* represents a physical or an abstract fundamental part of a system, as for example a line, an element or a node in a network.

A *display file* is a set of all instructions for a display (display instructions) which enable it to display the picture or a part of it at one time. These instructions are interpreted and displayed in the order in which they occur in the display file, except for jumps to various *display subroutines*. These subroutines are used to display 'instances' of various fundamental generic types of objects. (An instance of a display subroutine is a separate appearance of the corresponding object somewhere in the displayed picture. An example of a display subroutine would be a symbol for an element in an electrical network, such as a resistor.)

After exiting from any display subroutine the display continues from the place at which the last jump occurred.

A *data structure* (or an intermediate file) is a file which contains the complete description of a network or a system of networks with regard to all relevant aspects, i.e. display, analysis, etc. This description must include information about graphical properties (position on the screen, visibility, sensitivity to a light pen hit and so on) and semantic properties of all objects in the system as well as all the topological properties of the system.

The data structure is used to generate

(*a*) The desired displayed picture. For this purpose the data structure is 'down-compiled', i.e. it is analysed and all information concerning the pictorial representation of the part of the picture to be displayed is extracted and compiled into the display instructions, to produce the display file, and

(*b*) All the matrices which are necessary for analysis or design using the information about the semantic and topological properties of the network and the parts of the network.

In some program systems a *temporary display file* is used. This is a short file serving as an extension of the display file. In it, a limited number of display instructions are created, while a limited edition of the displayed picture is being shown. After the current amendment of the picture has been completed (i.e. a command is given to the effect that the current amendment has been completed) the temporary display file is 'up-compiled'. This means that the display instructions in the temporary display file, created during the last amendment of the displayed picture, are interpreted. The resulting information about the change in the displayed picture is used to up-date the data structure to include the recent changes in the picture.

After this the data structure is down-compiled to produce the

new display file and the temporary display file is ready for the next amendment of the displayed picture.

A *display structure* is a segmented form of a display file, in which the display instructions necessary to display any object participating in the displayed picture are collected together in separate blocks. These blocks are linked together by *display pointers* (display jump instructions) to form either a hierarchical tree structure or a ring structure, or a combination of both. The instructions in the structure are no longer interpreted and displayed in the order in which they are stored in the computer memory, but in a sequence governed by the display pointers.

The display structure offers greater flexibility during the editing operations. When adding or deleting new objects the entire structure need not be re-created as in the case of the display file. It is only a question of adding or deleting blocks corresponding to these objects and amending a few display pointers. Amendments of the displayed picture can be of any size, and are performed directly on the structure itself, so that a temporary file is not required.

The display structure requires slightly more space for itself than the display file (because of the presence of display pointers which link the structure blocks as a whole). On the other hand, there is an overall saving of space since there is no need to keep the pictorial information in the data structure. The small additional space needed for the display structure is more than balanced by the saving in space for the data structure.

3. Some examples of the use of the display file, the data structure and the display structure, in previous work

The process of up-compiling the temporary display file to amend the data structure and the down-compilation of the data structure to produce the display file is used in the program PIXIE built by N. E. Wiseman and his colleagues (1969). This up- and down-compilation is performed after every change of the displayed picture which includes deleting or moving one or a set of objects and adding a new portion to the displayed picture (which can consist of a number of objects).

This normally takes a few seconds, during which time the permanent display file is not displayed. This leads to a flick of the displayed picture at every up- and down-compilation. As both the display file and the data structure store the graphical properties of the objects in the displayed picture, this information is duplicated thus reducing the allowable size of the data structure because of the limited memory space. This implies a limitation in size of the displayed picture.

The generated permanent display file is compact and its copy can be easily reproduced. The necessity to have a temporary display file, which is relatively small, limits the amount of amendments of the picture at each step.

The data structure and the display file are linked by various non-display pointers. This assists in identification of the object, currently displayed, when a light pen hit or any other graphical type interrupt occurs from the displayed picture.

In another version of the display file, used in the SLAP program written by A. J. Drew, a short file contains displayable code for only one object at a time. To display a whole picture, the data structure is analysed in cycles, and after one object has been displayed the display code for the next object is generated in the display file from its description in the data structure. In the time which elapses after one object is displayed and the displayable code for the next object to be displayed is generated, the display is off; this introduces a short dark interval which increases the flicker of the picture.

To keep the display going, the computer re-generates, in cycles, the display instructions for each object participating in the displayed picture. This process engages the computer processor. Thus the processor is free to do other useful work only in the intervals when the displayable code for each object has been completed and is being displayed. As display speed increases, the computer processor is more heavily utilised in re-generating display instructions and available time for other tasks is diminished.

To store a copy of the picture, a copy of the data structure is stored and can be reproduced easily. Sometimes in this last version (Drew) there are two short display files and while one object is being displayed, the displayable code for the next is being generated in the second display file. When the display reaches the end of the displayable code for the first object and the display code for the second object has been completed, the display switches to the second file and the display code for the following object is being generated into the first display file.

The graphic program SELMA built by J. H. Jackson (1969) for the queueing network analysis program QAS developed by V. L. Wallace and K. Irani (1970), uses a display structure in a PDP-9 computer for the display and building of queueing networks. The data structure which stores, for analysis, the semantic and topological description of the networks built up on the screen is in an IBM 360/67, linked via a dataphone line with the PDP-9. The link to the 360/67 is in a conversational mode. By the user's actions the display structure in the PDP-9 and the data structure in the IBM 360/67 are amended together, so that up-to-date descriptions of the networks on the screen are kept in both machines.

In SELMA there are no one-to-one linkages to create correspondence links between blocks in two structures belonging to the same object; the user's actions are accordingly restricted. In the case of two or more networks on the screen a newly-built object can only be inserted in the network which was created last. The updated or newly-created connection line can only be inserted into the last connection block, and the up-dated or created parameter is assigned to the last created element. The user is thus forced to a specific sequence of actions while building the picture on the screen; this is a severe restriction.

An advantage in SELMA, resulting from the absence of data structure in the PDP-9, is that the memory portion dedicated to the display structure is bigger, thus allowing bigger networks to be drawn on the screen.

An interesting example of the use of data and display structures is in the multi-purpose graphics system built by C. Christensen, E. N. Pinson and colleagues at Bell Telephone Laboratories (1967). This system consists of a central computer, a set of devices for rapid hard-copy generation called STARE, a cathode ray tube output display called GLANCE and a set of interactive computer graphics terminals called GRAPHIC-2. Each of the GRAPHIC-2 terminals comprises a DEC PDP-9 computer and a special version of DEC-340 display. The central computer may be one of several available in the different BTL locations. The main structure called the Graphics Data Structure is in the central computer. As the purpose of this structure is to serve all three types of graphics devices, it is device independent. It comprises all information required for any of the three types of graphics devices and the data necessary for various analysis or design programs. It is structured in blocks, which are organised in a tree-hierarchical type of organisation. To obtain a copy of the structure for one of the three types of graphical devices the main structure is compiled to suit the corresponding device. The compilation is performed in the central computer and the resulting device-dependent structure is sent to the device. There is one translator for each type of device.

The structure in the PDP-9 is a display structure. It contains only display information and its organisation is identical to the structure in the central computer, with the difference that due to the small memory of the PDP-9 (8K), only blocks required for the current displayed picture are in the memory. To keep track of the blocks present in the display structure and to establish the correspondence between the blocks in the two structures, there is a block reference table in the PDP-9 memory. The

entries in this table link identifiers of the blocks in the central data structure to the addresses of the corresponding blocks in the display structure. Because of the slow transfer speed of the line between the central computer and the PDP-9 (2000 baud voice-grade), editing of the display structure, and so the displayed picture, is done in real-time by the PDP-9, and the information concerning the amendments is collected and sent in bulk, periodically, to the central computer. This information received by the central machine is then used to up-date the central data structure.

It is not possible to analyse this version of the data and display structures in the way done previously for other systems, because in the BTL system the central data structure is not dedicated to serve a single type graphic device but a group of different types of graphic devices. This obviously has a major effect on the organisation and the contents of the structure.

4. Requirements for an efficient data and display structure for general network analysis

Consideration of previous work and analysis of the requirements for an efficient combination of (*a*) a data structure and (*b*) a display file, or structure, led to the following conclusions

1. Picture display should be performed independently of a computer processor. This means that the display file or structure should be a self-contained entity which does not require cyclical generation in the computer processor.
2. Editing, i.e. adding new objects to or deleting objects from the picture, should be carried out directly on the display file or structure and in unlimited portions at any one time.
3. The nature of the display file or structure, and the relationship between it and the data structure, should be such as to ensure quick identification of any picture object which is the subject of a light pen hit flag or any other graphical type interrupt. The relationship should also enable quick location of a block in the data structure corresponding to the block in the display file or structure (and vice versa).
4. Duplication of the information in the display file or structure and that in the data structure should be minimal.
5. Both the display file or structure, and the data structure, should enable the storing and restoring of the network they represent.

The first point rules out the short type of display file consisting of the display data for only one object at a time, as described in paragraph 3.2.

The second point rules out the permanent and temporary display file and the up- and down-compilation, described in paragraph 3.1.

Considering points 1 and 2 together, the author came to the conclusion that any type of display file would be inadequate for the application; the design was, therefore, based on a display structure, as described in the next section.

5. The data-display structure

The design adopted, as a result of the foregoing considerations, is a single data and display structure, the *data-display structure*. It is a combined structure which serves both for the display and building of networks on the screen and also to store semantic and topological descriptions of networks used for analysis.

Because of the specific nature and topology of networks, a display structure with little additional information about non-graphical properties of objects in networks can satisfactorily describe networks for interactive graphics purposes, as well as for network analysis.

Let us look closer into this statement. Consider a network drawn on a sheet of paper or displayed on the screen. The diagram comprises a set of symbols representing network elements, a set of character strings, and a set of symbols representing diverse relationships between various network parts.

The set of symbols of network elements provides qualitative and quantitative information about elements incorporated in the network. The set of character strings depicts values of network elements and labels (names) for various network parts which have to be separately identified (labels for nodes for example). The set of symbols representing different relationships between network parts gives graphical indications as to which parts of the network are relationally linked. These are, for example, connectivity relations, where the fact that two element terminals are connected is represented by a connectivity symbol (a line) or a constitutive relation between a voltage-controlled current source and a voltage across an element. The latter can be represented by a dotted line from the control element to the controlled current source, and so on.

The diagram of a network on a sheet of paper or on a screen possessing all the aforementioned information contains all that is needed for a theoretical design or analysis of the network. To display a network diagram on the screen, depicting this information, the information must be stored inside a computer in displayable form. Therefore, we can state that all the information needed for the display of a network, as well as for its design and analysis is included in the display code. The task now is to organise this display code into a form which

(*a*) Satisfies the specifications listed in Section 4, and

(*b*) Is suitable for quick and efficient parsing, to extract parts of the information for diverse purposes, e.g. for design and/or analysis, for identification of objects involved in graphical interrupts, and so on.

The display data in the data-display structure is organised to form a display structure. Every object in a network displayed on the screen, where an object can be *physical* (an element or a connection) or *abstract* (a node), and also the *network* itself, possesses a corresponding block in the structure. These blocks are called *object blocks*, and they store all the data which is needed to display their associated objects.

Besides the display code, in the data-display structure there are included two types of non-display entities; these are *identifiers* and *relational-linkage pointers*. An identifier is associated with every block in the structure. It stores the *semantic value* of the object corresponding to the block and is the first word in the block. Its purpose is to describe the block, so to speed up the identification of blocks, both when interrupts are handled in the process of building a network on the screen and in processing the data-display structure to extract information about the network builtup on the screen, for design or analysis. As the identifier is a non-displayable entity, the display entry to every block is immediately after the identifier.

The second type of non-displayable entity in the data-display structure is the *relational-linkage pointer*. These pointers link blocks whose corresponding objects are participating in diverse types of relationship. Although these relationships are represented pictorially and so the description of their existence, nature and extent is included in the display data, the information regarding their topology (i.e. stating which objects are participating in which relationships) is comprised in the display data only indirectly. For example, this problem would arise if there is a requirement to determine which element is connected to a specified element terminal, given only the information about absolute co-ordinates of the element terminals and the ends of lines. (This type of information is actually all that is available to a computer from the display data representing a network on the screen, as the computer cannot 'visually' inspect a diagram on the screen.) To solve this problem one uses the criteria of coincidence of points, i.e. one would have to scan all lines and elements to find one whose co-ordinates of an end-point or a terminal respectively coincide (or do so closely enough) with the co-ordinates of the terminal in question. If the

found object is an element, then the search is finished, but if it is a line the search is restarted from the other end of the found line, and so on.

To speed up the process of defining relationships between objects in a network and to have always available a complete indication of the topology and the nature of the relations present in the network on the screen, the relational-linkage pointers were introduced*. These pointers exist, and are attached to a block, only when the corresponding object participates in a relationship. There are as many pointers attached to a block as there are relationships in which the corresponding object participates. These pointers precede the identifier of the block.

Fig. 1(*a*) shows an element in an electrical network. It represents a voltage-controlled current source. This object participates in three relationships; two of connectivity and one of controllability. The corresponding block is shown on the **Fig. 1**(*b*). A part of an electric network comprising the element and the corresponding part of the data-display structure are shown on **Figs. 2**(*a*) and **2**(*b*) respectively.

With T1, T2 and cntrl. in Figs. 1(*b*) and 2(*b*) are denoted relational-linkage pointers, corresponding to two connectivity relationships (via terminals T1 and T2) and one controllability relationship respectively, in which the corresponding element participates. ID denotes the identifier of the block. The dashed line in Fig. 2(*a*) depicts the controllability relationship between the voltage controlled current source and the voltage across the controlling element El 1. The dashed lines in Fig. 2(*b*) show the flow of relational-linkage pointers.

The data-display structure described above possesses all the advantages over a display file, derived in Section 4 from analysis of the requirements in general network design. Furthermore, the data-display structure, as a single, compact, and self-contained structure, which contains complete geometrical, topological and constitutive description of a network, offers greater simplicity and flexibility, in the process of building a network on the screen and in building a model of the network in the computer, over a tandem consisting of a display structure and a data structure.

It was mentioned before that in the case of a tandem of structures the sequence of users actions in the process of building a network on the screen by drawing it is restricted, or else it is necessary to include in the software system a set of complex software devices. The purpose of these software devices is to establish correspondence between blocks in the two structures belonging to the same objects and to enable extraction of relevant information about an object, when this information is kept in one structure, while a block, corresponding to the object and being processed, belongs to the other structure.

It is also the author's conclusion that a single data-display structure, as described, is more natural and more closely related to the system it represents (a network) than is a tandem of a display file/structure and a data structure.

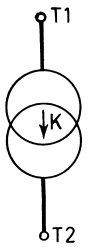

Fig. 1(*a*)

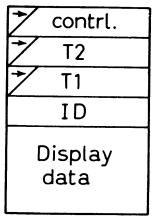

Fig. 1(*b*)

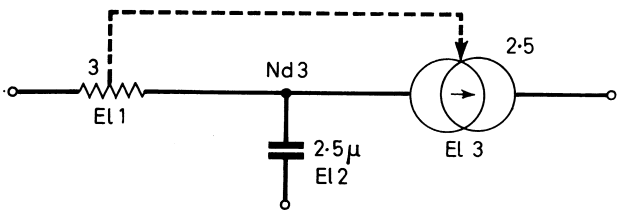

Fig. 2(*a*)

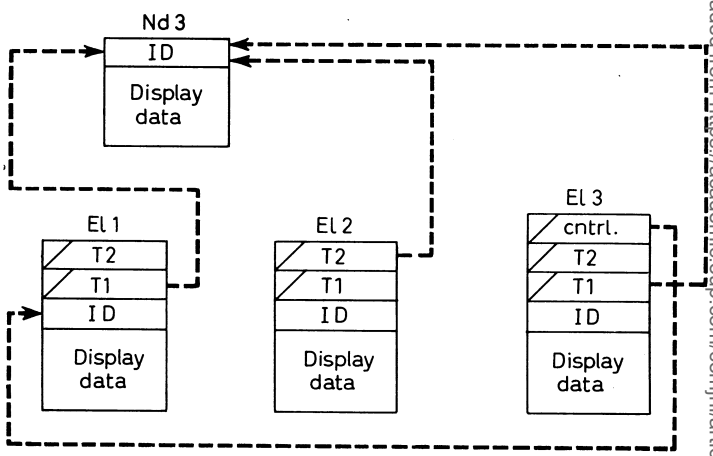

Fig. 2(*b*)

6. A graphics satellite computer/main backing computer and structures

After the completion of a network on the screen, parsing of the data-display structure provides the information necessary to build various network matrices. To perform the analysis, these matrices can either be sent by a link to a larger computer or used directly in the display computer (the computer associated with the display, which is a satellite to a large main computer). The larger computer is required if analysis involves extensive and complex arithmetic operations, which would take too much time to carry out on the satellite computer.

It is the author's strong belief that in a system consisting of a small computer with interactive graphics linked to a large main computer, it is unnecessary to have a fully conversational mode between the two computers in the phase of building the networks on the screen, or even to have a complex data structure in the large computer to represent the picture.

The designer, using the program to draw networks on the screen, makes errors in drawing and often changes his decisions (i.e. he deletes a part of a network, or he changes the geometry of one part of the network); also he takes his time in thinking. If the program, in responding to his actions, up-dates a data structure in the large computer after every designer's action, much unnecessary core and CPU time is used. It is far more economical to draw a network, using only the small display computer, and when the network is completed and an instruction for analysis is given, the small computer sends all the prepared data to the large machine. While the analysis is being carried out in the large computer the designer can either wait for the results to be displayed, if the analysis is expected to be a

*As previously pointed out, the information which explicitly identifies objects participating in diverse relationships is not directly included in the display data; presence of this information in the form of relation-linkage pointers does not, therefore, mean duplication of information. Also, the introduction of relational-linkage pointers is justified from the point of view of efficiency of storage management. Each pointer occupies one word of memory. However, the software required to search the data-display structure to determine the topology of relationships is very much smaller and faster when the pointers exist.

short one, or he can start drawing another network on the screen.

When the small computer receives a message that the analysis in the large machine has been completed, a choice is required from the designer.

The designer can either proceed further with the drawing already started or he can give an instruction to save the created portion of the new picture (on disc or magnetic tape) and ask for the result of the analysis to be displayed. After examining the results he can continue to draw the new network on the screen.

Using this mode of link between the two computers, continuity between the drawing and analysis stages of a network is provided. It is more economical than having a fully conversational mode with the large computer during the drawing stage, but more expensive than normal batch mode, because the link to the large computer must be kept open continuously to deal with a request for analysis.

7. Conclusion

This paper describes a single structure, the data-display structure, which has been implemented for general network analysis using interactive computer graphics.

The data-display structure serves both to describe the network in full pictorial details for display, and also to build all the necessary mathematical description of the network. This leads to the omission of a separate data structure, which was necessary in the case of the display file or structure. The underlying idea is simple. Just as a picture of a network, on a sheet of paper, carries all the information necessary for the designer to analyse the network, so should the picture of the network on the screen possess all the information needed for the computer program to analyse the network. The data-display structure is the image of the picture on the screen, and so should be the starting point for the network analysis. This leads to the conclusion that separation of the display file or structure, the means to display the network, from the data structure storing the description of the network for analysis (and building the display file where a display file is used instead of a display structure) is unnatural.

Acknowledgements

I wish to thank Professor V. L. Wallace, now at the University of North Carolina, Chapel Hill, for the helpful discussions we held together during his time as an Academic Visitor to Imperial College, at the initial stages in the development of the structure. His experience as one of the co-authors of the program for queueing networks analysis in Michigan (Jackson, 1969; Irani, Wallace and Jackson, 1970) and his ideas enriched the basis of the structure and helped to channel my research in the current direction.

Thanks are also due to Professor W. S. Elliott who read the manuscript of the paper and made many useful suggestions and to the SRC who provided a grant to enable the research work to be carried out.

References

CHRISTENSEN, C. and PINSON, E. N. (1967). Multi-function graphics for a large computer system, AFIPS Conference Proceedings, *Fall Joint Comp. Conference*, Vol. 31, pp. 697-711.

DREW, A. J. SLAP—Small Linear Analysis Program, program for AC steady state analysis using computer interactive graphics, Imperial College, London—to be published.

IRANI, K. B., WALLACE, V. L., and JACKSON, J. H. (1970). Conversational design of stoacastic service systems from a graphic terminal, International Symposium *Computer Graphics 70*, Brunel University, Uxbridge, Middlesex, England.

JACKSON, J. H. (1969). SELMA—A conversational system for the graphical specification of Markovian Queueing Networks, Report of Department of Electrical Engineering, System Eng. Lab., the University of Michigan, Ann Arbor, October.

MAROVAC, N. *The data-display structure elements*, internally circulated.

WISEMAN, N. E., LEMKE, H. U., and HILES, J. O. (1969). PIXIE—A new approach to graphical man-machine communication, International Conference on *CAD*, 15-18 April, 1969. The University of Southampton (conference publication).