An experimental on-line network design system is proposed. Called DESIGNPAD, *it consists of a small computer with graphic display equipment connected to a time-sharing computer and includes the necessary programming support for the equipment.*

The system is designed to accept problems covering a broad spectrum of applications in the form of labeled block diagrams. The input/output medium, the man-machine interface, and the supporting data structures, particularly the cellular structure, are discussed.

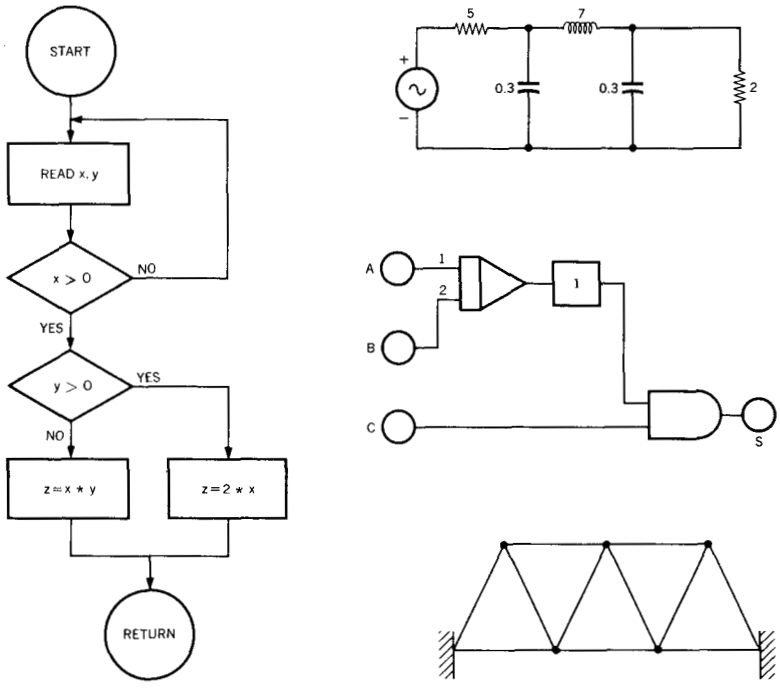
A computer graphics system for block diagram problems

by L. A. Belady, M. W. Blasgen, C. J. Evangelisti, and R. D. Tennison

Generally it is easier to express complex ideas symbolically by sketching them on paper rather than by using other means of communication. When such two-dimensional representations were applied to computer output, they resulted in the development of computer graphics.[1] But they are only now being applied to the input. We anticipate an evolution in input from one-dimensional character strings to two-dimensional methods occurring similarly to the one in which programming languages evolved from machine code and assembly language to such higher level languages as PL/1. To further this evolution, the construction of an experimental system called DESIGNPAD was started with the major single objective to explore the potential of two-dimensional man-machine interfaces for both computer input and output.

Frequent need for total reprogramming to reflect even slight variations in functional specifications has been an unfortunate attribute of computer graphics. For example, the case of a program for graphical electrical network analysis[2] being incompatible with a system for graphical programming using flowcharts[3] is a manifestation of this difficulty, although there are commonalities between these two problem-solving areas. DESIGNPAD is a single, integrated graphics system that can handle both these applications and similar ones. First outlined by Baskin and Belady,[4] it is an extension of the work by Baskin and Morse[5] which developed from work by Sutherland.[6]

Figure 1 Examples of labeled block diagrams



DESIGNPAD applications range over all problems that can be represented by *labeled block diagrams*. A labeled block diagram consists of interconnections or *lines* between *attacher points* of blocks, with text possibly associated with these elements. Examples are shown in Figure 1. Potential applications include any discipline that uses block diagrams for problem specification, even conventional programming techniques, since a degenerate case of a labeled block diagram is text alone. A precise definition of a labeled block diagram appears in Appendix A.

The DESIGNPAD system contains facilities for structuring these diagrams in an interactive fashion. Because of the choice of application domain, this input phase is common to all users, permitting the entire graphics section of the system to be application independent. Thus, at the same time, we gain extensive commonality in programming and a large scope of applications.[7] We feel that this range of applications is an important feature of the DESIGNPAD system.

A new concept developed in the system employs a data structure called a cellular structure to implement the unbounded two-dimensional drawing media. Other concepts such as the hierarchical library structure, the use of two computing systems (a system-subsystem) with the associated function allocation problem, the drawing package, and the data structures used in the system are not

necessarily novel but are combined in DESIGNPAD in an original configuration.

In this paper, we first describe the approach used in organizing the system. Then the various functions performed by different parts of the system are presented. Finally, the data structures necessary for the operation are discussed.

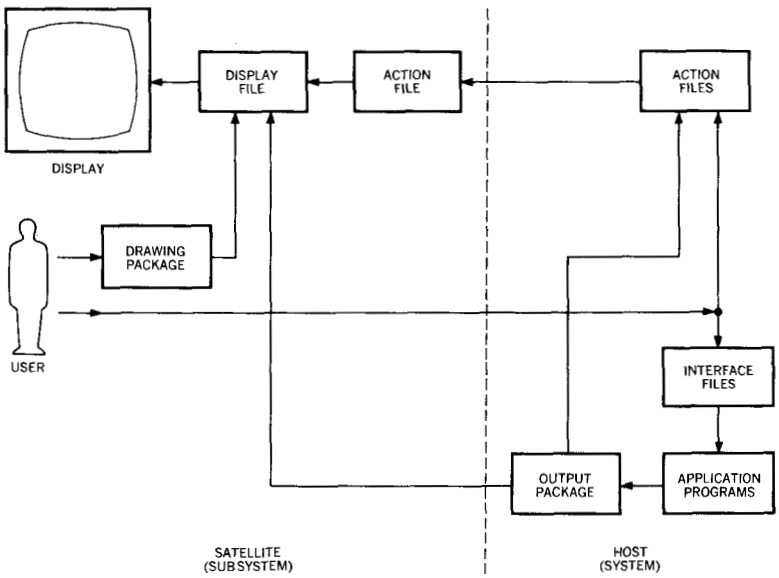
The system-subsystem approach

DESIGNPAD was designed as an experimental extension to a time-sharing machine. To be economically effective in operation, the system was required to make use of existing time-sharing programs without extensive reprogramming. Correspondingly, the experimental vehicle is a system-subsystem arrangement where the *host computer* is a conventional, large data-base, time-sharing machine with a small *satellite computer* attached to it. This organization has been used in several other projects.[8] The host, a time-shared System/360, Model 67 running under the time-sharing system TSS/360, is the central analysis machine, whereas the satellite, an IBM 1130 combined with an IBM 2250 display unit (Model 4), is a high-capability user terminal. The satellite communicates with the host via standard telephone service at 2000 baud. The hardware in the system is standard,[9,10] and DESIGNPAD programs and data residing in the host are stored and accessed using the standard TSS facilities.

The system-subsystem arrangement shown in Figure 2 offers advantages as well as some problems. The structuring of ideas, or modeling, becomes restricted to the satellite with little or no work load presented to the host until some analysis of the model is desired. This results in relatively infrequent access to the powerful host with its large library, and intimate man-machine communication is concentrated in the satellite. Restricting the communication in this manner permits the satellite to function as a transformer of graphic information into strings and vice versa such that the string format is compatible with the conventional requirements of programs in the host library. For example, existing packages like GPSS,[11] ECAP,[12] and CSMP[13] or even LISP,[14] can be imbedded in the DESIGNPAD framework.

By organizing the system in this manner, the DESIGNPAD effort was directed toward the development of the *modeling subsystem* on the satellite (i.e., the programs run on the satellite computer), facilitating the creation of graphic input and providing for visual feedback and the design of data structures for both machines. The data structures should respond efficiently to manipulations on, and the display of, block diagrams and serve as conventional input/output files for existing program packages.

Figure 2 DESIGNPAD organization



function allocation

Among the many functions of the system, some must be provided by the satellite and some by the host, but there is a gray area in between. For this reason, DESIGNPAD has considerable flexibility to permit experimentation in function allocation. For example, because of the low-speed intermachine communication, certain groups of data (specifically the action file discussed later) are stored almost identically in both machines. Experience may show that such redundancy is unnecessary, or it may show that the low-speed communication is intolerable. Another example of an experiment is the determination of whether the package of output subroutines should reside in the host, the satellite, or both. The point here is that DESIGNPAD is at least as much a system study as it is a graphics interface experiment. What follows is thus a description for the initial configuration of the DESIGNPAD system.

The satellite processor supports the input/output tasks for the designer, providing immediate visual feedback to him of the system's interpretation of his actions. In this way, the demands for rapid response are handled locally at the graphics terminal. Once the user has confirmed his action (e.g., "draw line"), the result is sent to the host system for subsequent analysis or recall. Since only observed and confirmed actions are sent to the host, the transmission rate of 2000 baud is not seen to be a bottleneck for transmission from satellite to host. If a large data transfer is requested by the satellite from the host data structures, however, some noticeable delay is expected. For example, to transmit a file of a thousand 32-bit words (corresponding to a very large action file), the transmission delay would be approximately 30 seconds.

The host processor manages those tasks not requiring quick response but rather large complex supporting programs. Thus the host processor manages the major data structures, carries out whatever analysis of data is necessary, and generates output. This support includes the recording of all user actions, the extraction of the relevant geometric and topological information from the model as represented by a list of user actions, and the execution of analytical programs.

modeling media

To provide a smooth transition for DESIGNPAD users from both conventional time-sharing operations and from the use of drafting boards, the concept of a *modeling sheet* was introduced. The sheet acts like a large piece of paper and is the basic two-dimensional medium, the device-independent carrier of all input or output information from user or host whether graphic, textual, or both. DESIGNPAD is equipped to handle a large number of sheets for each user.

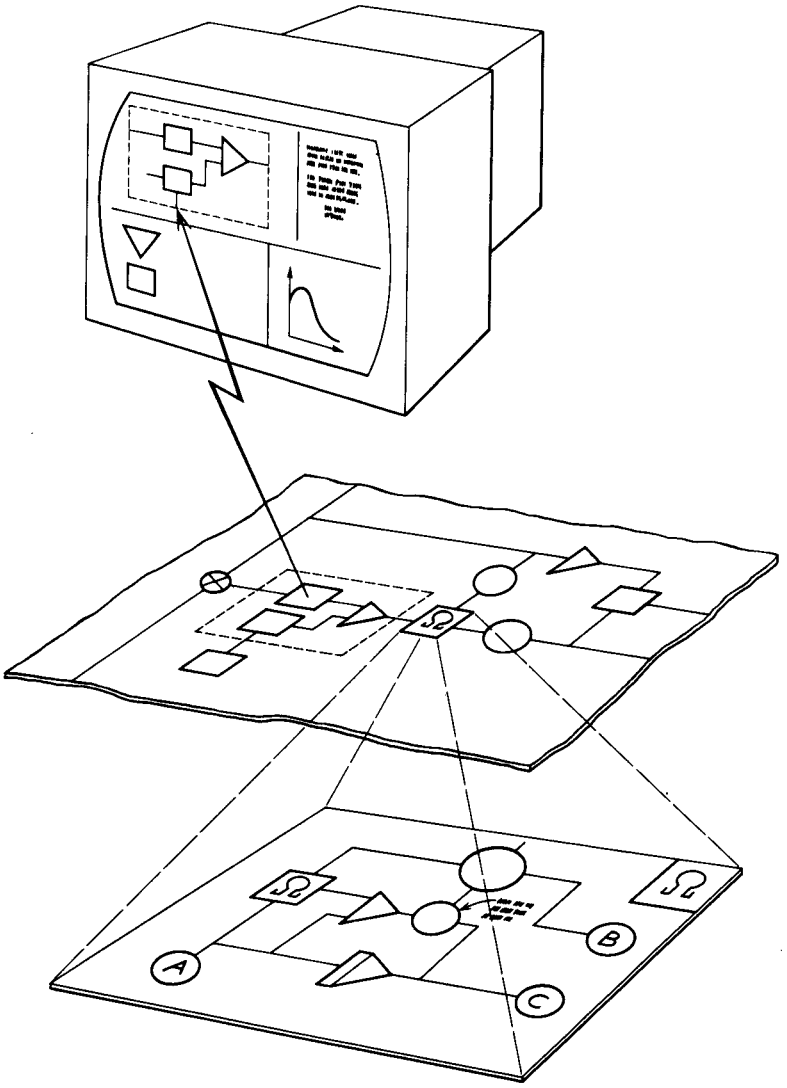
The sheets are equivalent to a sheet of paper sixty feet square. To utilize this large sheet, a user first begins work in the portion of the sheet visible on the display unit. If more area is needed to draw the model, the area in the display can be moved to access an unused portion of the sheet.

At the satellite, a sheet may be displayed on the screen, a model constructed on it, and then the entire contents of the sheet transmitted to the host for analysis. The host similarly formats graphic output and transmits it to the satellite. Each sheet has a unique name which is generally symbolic (like a picture of a block) and which may be specified by the user. The symbols corresponding to all the sheets available to the user, plus the additional symbols to be used as blocks in the block diagrams, are gathered together on a *library sheet.* Models are constructed graphically by using built-in system functions such as "draw line" and by referring to another sheet containing blocks in order to obtain a copy of a particular block. Reference for the user is ultimately made possible by his assigned library sheet containing the blocks he may use in his model. Some of these blocks may represent sheets he has created (or will create) thus permitting both hierarchical and recursive use of blocks in the system. The concepts of hierarchy and recursion in DESIGNPAD are illustrated in Figure 3.

The modeling subsystem

This section discusses the modeling subsystem of DESIGNPAD, which provides the support at the satellite display system to construct models, transmit requests for analysis to the host machine, and examine output. The subsystem provides a program technique for displaying the contents of sheets, a drawing package and a text

Figure 3 Viewports, sheets, and windows showing hierarchy and recursion



editor to modify the contents of a sheet, and an output package to receive output from the host after analysis has been requested. Sheets can be filed and retrieved, and there is a block specification facility where special user-defined blocks can be constructed. Appendix B contains a more detailed description of the operational facilities and control commands.

viewports and windows

To examine graphic input and output, viewports and windows are provided. Each *viewport* is a rectangular area, the size of which is adjustable. Up to four viewpoints can be obtained by dividing the display screen so that the user can view parts of four different sheets simultaneously. Using this capability, for example, one viewport could contain the portion of the model that is under construction

or being analyzed, while another viewport could be used to display the blocks that may be copied into the model. A third viewport might be used to display a sheet containing light keys, and the fourth could be used to examine the results of the analysis of the network.

The boundary of the displayed portion of the sheet in the viewport is called a *window*.[1] A procedure known as "windowing" allows continuous change of the window position on the sheet to display in a viewport a portion of a sheet. In this way, a large model can be constructed by creating a portion of the model and then windowing over a few inches in order to continue. Also, many light keys can be placed on a single sheet, a circumstance that may prevent all the keys from being simultaneously displayed, but one in which windowing can be used to gain access to them. Figure 3 illustrates the use of four viewports and the windows.

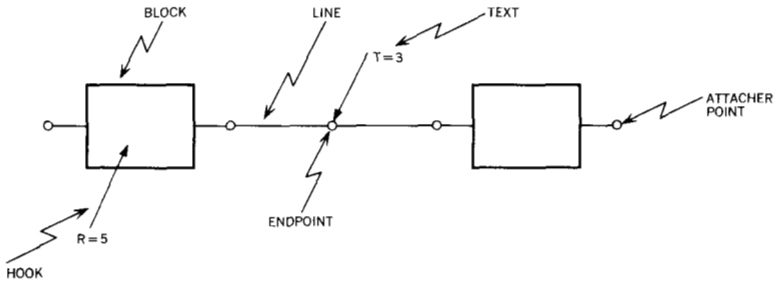
drawing package

A user wishing to construct a block diagram on a sheet uses a set of facilities called the *drawing package*. With the drawing package, he may copy an individual block onto a sheet. (A block is a symbol that is designed by the user with special points called *attacher points*.) He may draw a line or a series of lines between attacher points. The end of a line not connected to an attacher point is called an *endpoint*, and endpoints may also be connected with lines. A user may also erase blocks and lines from a sheet.

Although the drawing package permits the construction of block diagrams made up of blocks and interconnections, text is usually required to completely specify the network. Thus DESIGNPAD has a text-editing facility incorporated into the drawing package for the creation, modification, and elimination of textual entities. A *textual entity* (or simply text) is a group of alphanumeric characters that are logically treated as a single unit, much like a block. The text-editing commands permit creation of text at a specified location, character-by-character editing, and deletion of the entire entity. It is well to point out that the contents of a sheet may be entirely text, as in the case of a FORTRAN program, for example. If text is mixed with blocks and lines, it becomes necessary to have a facility to associate text with particular blocks (as in the case of program flowcharts), or with particular attacher points (as in the case of integrators in an analog computer simulation). This is also provided by the drawing package and permits the creation or deletion of *hooks* that associate text with endpoints, blocks, or attacher points. These terms are illustrated in Figure 4.

In the modeling mode, two of the viewports are designated the *modeling viewport* and *reference viewport*. The user constructs the desired block diagram on the sheet displayed in the modeling viewport. This construction process requires the use of the reference viewport, in which is displayed a sheet containing blocks.

Figure 4 An example of blocks, texts, attacher points, lines, endpoints, and hooks



The existence of the two viewports and the variety of operands—blocks, attacher points, endpoints, lines, texts, and hooks—provide for a drawing package without explicit commands. In a conventional drawing package, a command is given along with a set of operands. For example, the command to copy a block is followed by two operands: which block to copy and where the block is to be copied. The DESIGNPAD drawing package eliminates the commands for operations used to construct block diagrams. A pair of operands alone determine which operation is to be executed. For example, a block from the reference viewport and a point in the modeling viewport determine that a block is to be copied. The drawing functions are described more fully in Appendix B, and a complete description is available elsewhere.[15]

block specification

In the block specification mode, the user may create a new block by specifying its shape and function. Thus there are two phases, the first being a drawing phase where the appearance of the block (shape, number of attacher points, etc.) is determined by the user, and the second a specification of the function of the block. In the drawing phase, the system provides the user with a set of graphic functions, not unlike the implied drawing package, such that the block can be designed quickly. In the specification phase, the user may determine the function of the block on its associated sheet. One way to specify the function of a block is to construct a network on this sheet (possibly including the block being defined, thus permitting recursion). This introduces the possibility of defining hierarchical networks which, combined with the windowing facility, permit the construction of large and complex networks.

file storage and retrieval

DESIGNPAD provides a facility for the filing and subsequent retrieval of sheets. Thus a model at any stage of its construction may be filed for later use. This storage and retrieval is done in one of two ways. As the model is constructed, a *display file* that is used to refresh the display unit is produced in the main memory of the satellite and stored on its disk, and simultaneously an *action file* is created in the host. The action file is a chronological list of all the actions the user took in constructing a model (for example: draw line, copy block, draw line, etc.). Thus at any time there are two

files in the system capable of reconstructing the sheet for viewing on the display. At retrieval time, the system first attempts to locate the display file on the disk of the satellite. Because of the size of this file, however, the system may have replaced the desired sheet with the display file of a more recently accessed sheet. In this case, the system requests that the host transmit the appropriate action file to the satellite which then dynamically recreates the display file. An important point here is that a display file can be created either by a user drawing on the display unit or by an action file transmitted from the host. There is only one set of programs that create the display file; however, its input may be taken from either source.

analysis

Once the block diagram is complete and all the necessary blocks have been specified, the analysis mode is entered, and the application program (normally supplied by the user) analyzes the network. Input to the application program is taken from the *interface file*, so named because it provides the interface between the DESIGNPAD system and the user's application program. This file is discussed in detail later in this paper. The system has been designed so that all application programs accepting any representation of a labeled block diagram as input can be incorporated.

An output package that permits displaying results of the analysis is provided in the host. The package includes subroutines for plotting conventional x-y graphs, bar graphs, projections of three-dimensional surfaces for functions of two variables, and other graphs on the satellite display unit. In addition, application programs may create an action file in order to produce block diagrams as output. This permits the display of text or of complex shapes made up of lines produced with the drawing package. The output produced with these routines is placed on a sheet, thus allowing the drawing of numerous plots and use of the windowing facility to move from one to another. The output, being simply a sheet, may be filed and later retrieved for further study in any viewport. Besides providing static output in the form of graphs and models, DESIGNPAD provides dynamic output (animation).

Data structures

This section describes the data structures (or files) found in DESIGNPAD. These structures include the action file, the display file, and the interface file. The overall organization and data flow are shown in Figure 2.

action file

As stated previously, the satellite computer acts as a terminal and issues a series of transactions representing user actions. These actions determine the graphical elements that make up the contents of a sheet and are simultaneously transmitted to the host and recorded there in a file called the action file. New actions taken on

a sheet update this file so that a sheet generated at the satellite is completely described. The contents of all sheets therefore reside in the host. Since storage in the satellite is limited, the action file in the host is used as secondary storage.

In addition to using the action file for storage, DESIGNPAD also uses the file to generate animated output. An action file describing a series of changes to a sheet can serve as a script for animated output. Animation is accomplished by having an application program in the host produce a sequence of commands that describe the initial frame and subsequent changes to that frame on an action file. A special transaction called "mark frame" acts as a delimiter by marking the beginning of each frame of animated output. All transactions between the special mark frame transactions constitute the modifications to the frame of animated output. A moving line, for example, is represented by the following transactions: draw A, mark frame, erase A, draw B, mark frame, erase B, draw C, mark frame, etc. An action file describing animated output is then transmitted to the satellite where it is processed for display. Since the file is now associated with the satellite and no further host-satellite transmission is required, the animation can be presented at normal speed. Such an action file may be stored in the satellite, and a user may run an animation many times.

display file

As the designer uses the drawing package to create a model on a sheet being displayed, his action builds a data structure, the display file, in the satellite. The display file is a set of *graphic orders* that are commands to the display unit to generate an image. The display file for a sheet thus contains a large number (perhaps thousands) of commands taken from a small set of graphic orders (DESIGNPAD uses only eight with any frequency). If this file is presented to the display unit, the sheet will be displayed. Unfortunately this makes no sense in DESIGNPAD since the sheet is very large compared to the screen size. Most display units will display unintelligible information when presented with a display file that requests off-screen images. To overcome this problem, two solutions—scaling and windowing[1]—were considered. Scaling reduces an image to fit into a given area, whereas windowing (or scissoring) "cuts out" and displays a portion of the image. To provide the user with flexibility, either technique must be done dynamically so that the windowing or scaling can be quickly changed under user control.

cellular structure

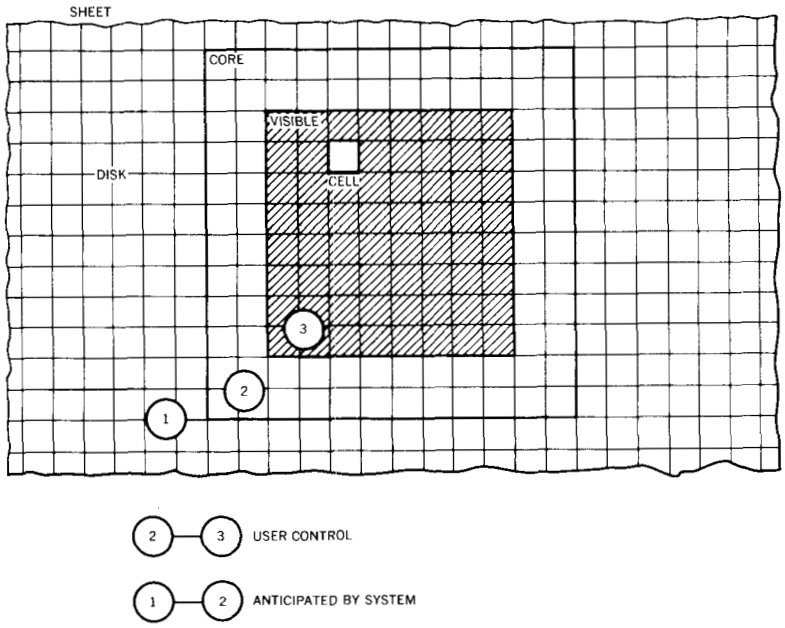
DESIGNPAD incorporates only windowing, since dynamic scaling requires special hardware and is not by itself sufficient; it is not possible to construct a model on a sheet whose scale has been significantly reduced. Dynamic windowing can be carried out in two ways—by special hardware or by an appropriately organized display file. It was felt that special hardware would make the resulting system less transferable to a new location, and in addition, hardware windowing implies certain data management problems

discussed later in this paper. Therefore, DESIGNPAD uses the second alternative, employing a *cellular structure* for the display file. A cellular structure is implemented by conceptually partitioning the sheet into cells one and a half inches square. (This is considered to be the most convenient size; however, this is another example of a parameter to be determined by experiment.) The display file associated with the sheet is then divided into groups of graphical data. The orders which make up this graphical data are incremental orders (graphic orders which, independent of the absolute beam position, displace the electron beam a specified amount) and together form a graphic subroutine. A cell is displayed on the screen by executing its graphic subroutine. All graphic objects crossing cell boundaries are divided into subparts and placed in the appropriate subroutines. For example, a line which passes through four cells is broken up into sublines, and these are placed into four subroutines. A textual entity that spans several cells is partitioned into subentities that are again placed in appropriate subroutines. The subroutines are assigned names corresponding to their coordinates on the sheet, stored on the satellite's disk, and later accessed by using their names.

When a portion of a sheet is to be displayed, the cells (or more accurately, the subroutines corresponding to the cells) for that portion are retrieved from disk and displayed as shown in Figure 5. Each cell is preceded by a positioning vector, a graphic order which moves the beam to a specified position but does not cause a line to appear on the screen. When a user "windows" using one of the viewports, the positioning vectors for the cells under that viewport are modified causing the entire sheet to move under the viewport. When a column or row of cells passes over a viewport boundary, the entire column or row of cells is removed from the graphic orders being displayed and a new column or row of cells is displayed. New cells do not appear until there is room for them. Cells surrounding the ones being displayed in a viewport are kept in a main-memory buffer so that a user will not experience delays during windowing. The buffer will be replenished with cells from disk depending on the direction and speed of windowing. At the completion of windowing (caused by confirming the present position), the sheet registers itself at the cell boundaries automatically.

As previously stated, the sheets are divided into cells to allow a continuous view on the sheet as it moves under the viewport. This "prescissoring" eliminates the need for a scissoring operation while windowing; thus the windowing is rapid and requires no special-purpose clipping hardware. Another advantage of cellular sheets is that searching display files on the x, y position of the input device is more efficient. Such searching is necessary if, for example, the user points at a line on the screen and requests that the line be erased. Since DESIGNPAD uses the pointing device (a light pen, but it could be the stylus of a tablet) only to obtain x, y positional information, the display file must be searched to locate all graphical

Figure 5 Cellular structure and window

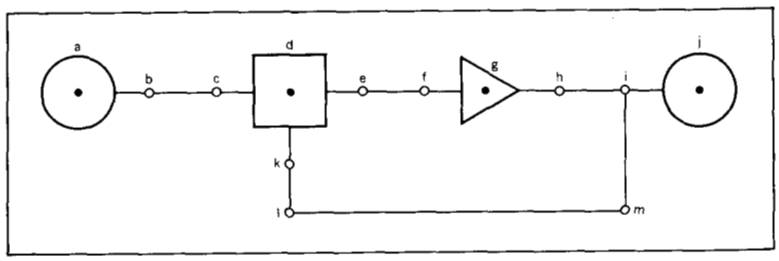


entities at that position. The entire display list must be searched in the usual organization, but only a single cell need be searched when the display list has a cellular structure.

Scissoring "on the fly", which is an alternative to cells, solves the problem of separating the visible information from the invisible provided both are kept in high-speed storage. Unfortunately a memory management problem arises since main memory can overflow in an attempt to accommodate large sheets. The cellular structure, however, solves this problem by prescissoring, and the cell becomes the single fundamental unit of display information for transition between visible and invisible and between main memory and disk. Figure 5 illustrates this management technique.

There are some disadvantages to the cellular structure. Since some graphic entities are broken up into subparts and placed into cells, it is necessary to reconstruct these entities for such graphic functions as erase. This operation can be time-consuming. The problem can be avoided by storing more information. For example, the relation between line L and sublines a, b, c could be entered into a special data structure and later retrieved. Another approach is to search the cells for the desired information. Suppose all the lines that share point A as endpoints are to be found. A special graphic order is inserted to mark endpoints of lines. The cell on which point A lies is searched. If an endpoint is found, all sublines sharing point A are placed on a list. For each subline on the list, a search will be made for additional sublines in other cells until complete lines have been found. Either approach, the supplementary data structure or the

Figure 6 A sheet generated at the satellite



extra orders, requires approximately the same amount of storage; the latter, however, is attractive because it places all information in a single data structure.

Another disadvantage of the cellular structure manifests itself in the process of windowing. When a user windows, a column or row of cells drops from view suddenly at the edge of a viewport. This would not happen if special clipping hardware were employed. However, this phenomenon occurs at the edge of the viewport only, whereas the motion at the center is smooth.

interface file

Each sheet generated at the satellite has associated with it an interface file at the host. The file, which is the input to application programs, is a compound data structure[16] built from transactions received from the satellite. The file contains all the relevant geometric and topological information and is updated by any new transactions. An example will be used to illustrate the organization of this file.

Figure 6 shows a sheet that has been generated at the satellite. The sheet contains four blocks of different types. Each of the four blocks has a unique name, its (x, y) coordinates on the sheet (e.g., a, d, g, j). The type of block is represented by the coordinates of that block on the library sheet. Thus a block's type is its name on the library sheet. The sheet also contains six lines, and the name of a line is determined by the coordinates of its two endpoints.

Figure 7 shows the transactions sent to the host for this sheet. The first transaction identifies the sheet to which the succeeding transactions refer. A transaction to copy a block has two parts of data: where the block is to be copied on the sheet and the type of block. A transaction for drawing a line requires the coordinates of the endpoints of the line.

These transactions are inputs to the program that produces the interface file. The interface file consists of *entities*, *attributes*, and *data*. These notions are also used in the LEAP language and elsewhere.[16–18] The entities are the basic elements of the file, and data is attached to them. Examples of entities are instances of blocks, lines, hooks, and texts. To associate some data with an entity, the

Figure 7 Transactions from satellite to host

TRANSACTION*	COMMENTS
ON SHEET A	THE FOLLOWING TRANSACTIONS ARE ON THE SHEET WHOSE BLOCK IS AT A ON LIBRARY SHEET.
COPY a,B	COPY AT LOCATION a ON THE ABOVE SHEET AN INSTANCE OF THE BLOCK THAT IS AT B ON THE LIBRARY SHEET.
DRAW b,d	DRAW A LINE FROM b TO d.
COPY d,C	
DRAW e,f	
COPY g,D	
DRAW h,i	
COPY j,E	
DRAW k,l	
DRAW l,m	
DRAW m,i	

*UPPER CASE LETTERS ARE COORDINATES ON LIBRARY SHEET,
LOWER CASE ARE COORDINATES ON THE MODELING SHEET.

Figure 8 A representation of the initial entries in the interface file

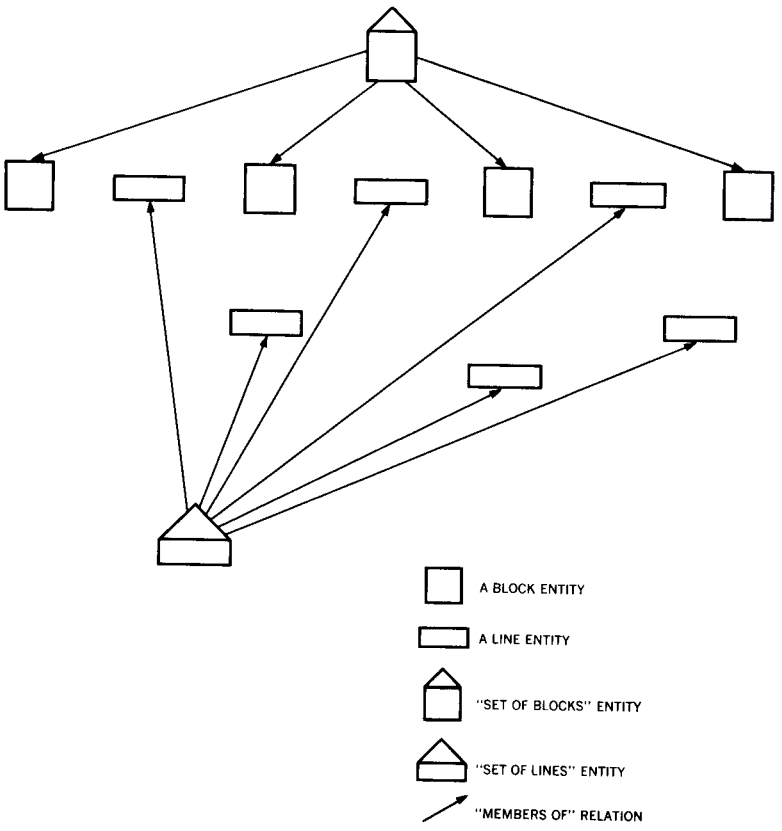
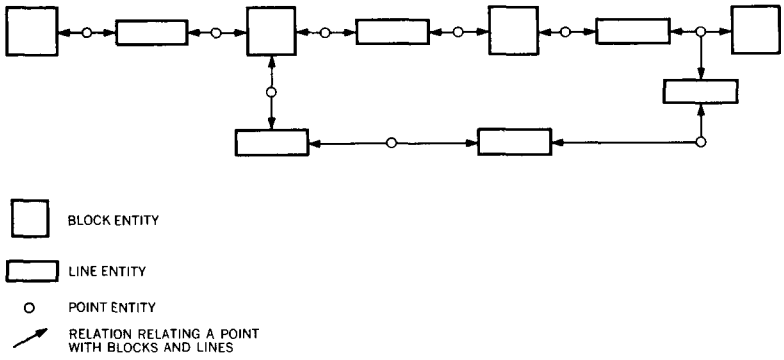


Figure 9 Interface file with new entity and new relation



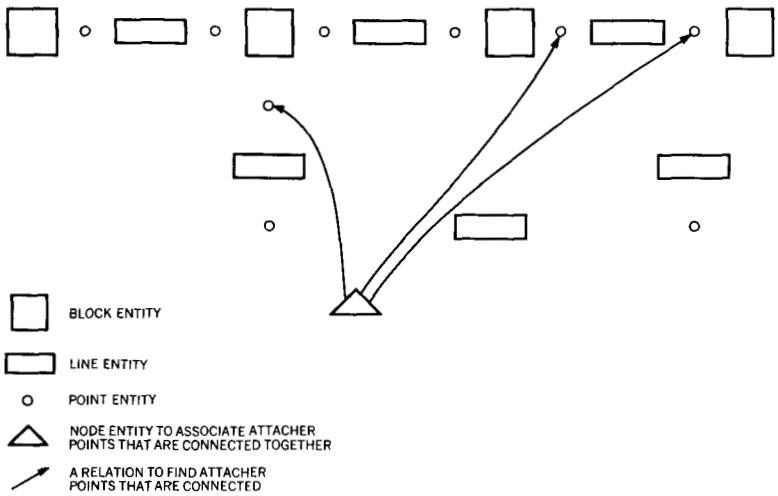
data is filed with that entity under an attribute. For example, if the number of attacher points for Block A is five, an entity with the name BLOCK A is created, and the attribute NUMBER OF ATTACHER POINTS OF is used to file the data "5". Later when the query "NUMBER OF ATTACHER POINTS OF (BLOCK A) is ?" is presented to the data structure, the answer "5" is returned.

In addition to associating entities with data, attributes can associate entities with other entities. Such an attribute is called *relation*. To illustrate the concept of relation, suppose that POINT A is an entity representing point a, LINE AB is an entity representing line a, b and that the attribute LINES BEGINNING AT applies to the entity POINT A. The answer to the query, "LINES BEGINNING AT (POINT A) are ?" would be LINE AB. In this case, the attribute returns another entity as a value.

Figure 8 shows a representation of the interface file at a point where the basic data from transactions have been added to it. The square symbols represent instances of blocks and the rectangular symbols represent lines. The symbols in the figure have been placed so that their positions correspond to the positions of the lines and blocks in Figure 6. As shown in Figure 8 two new entities have been added to the interface file. One represents the set of lines and is called SET OF LINES; the other represents the set of blocks and is called SET OF BLOCKS. The relation MEMBERS OF is assigned to these new entities. Thus we have a means of getting all the blocks in a sheet simply by asking "MEMBERS OF (SET OF BLOCKS) are ?". Similarly, the set of lines in the sheet may be accessed.

In addition to the line and block entities, Figure 9 shows a new type of entity that has been added to the file. The new entity represents a point on the sheet, either an attacher point of a block or the endpoint of a line or both. A new relation is also added to the file to relate points with blocks and lines. Thus lines that end on attacher points of a block are related to the block through the new relation and the new entity.

Figure 10 Interface file with new entity and relation giving topology of a diagram



Finally, Figure 10 shows another new entity and new relation added to the file. The entity represented by a triangle is used to collect the set of attacher points that are wired together. Thus the topology of the diagram has been extracted. The attacher points of connected blocks are known independent of the number and location of the lines connecting them.

In this example, the manner in which lines were used to connect attacher points was not material. In other applications, the arrangement of lines may carry information for the application program. In fact, a diagram may consist simply of lines, texts, and hooks. The flexibility of a compound data structure is required to handle a variety of applications. Many such data structures have been implemented.[16]

Summary

This paper has described the design of an experimental system intended for solving complex problems. Facilities are provided for stating a problem as a labeled block diagram of arbitrary complexity. A block diagram can be drawn on a representative sheet that is many times larger than the screen of the cathode-ray tube. The block diagram may be structured into a hierarchy of sheets allowing blocks to be designed and specified by the user. The modeling subsystem provides a number of viewports so that more than one sheet can be displayed. To make entering of block diagrams easier, a drawing package that does not require commands for creating the diagram but uses only the operands to determine the command is a part of the system. Finally, a data structure in the host computer is provided to allow for easy access to data by any application program.

ACKNOWLEDGMENTS

The help of A. J. Stein and V. Watson of the IBM Thomas J. Watson Research Center in programming the modeling subsystem is greatly appreciated.

CITED REFERENCES

1. C. I. Johnson, "Interactive graphics in data processing: Principles of interactive systems," *IBM Systems Journal* **7,** Nos. 3 & 4, 147–173 (1968).
2. F. F. Kuo, W. G. Magnuson Jr., and W. J. Walsh, "Computer graphics in electronic design," *Datamation* **15,** No. 3, 71–79 (March 1969).
3. W. R. Sutherland, *On-Line Graphical Specification of Computer Procedures,* Technical Report 405, Lincoln Laboratory, Massachusetts Institute of Technology, Cambridge, Massachusetts (May 1966).
4. H. B. Baskin and L. A. Belady, "DESIGNPAD: A graphic design/problem-solving facility," *Proceedings of the Third Annual Princeton Conference on Information Sciences and Systems,* 173 (1969).
5. H. B. Baskin and S. P. Morse, "Interactive graphics in data processing: A multilevel modeling structure for interactive graphic design," *IBM Systems Journal* **7,** Nos. 3 & 4, 218–228 (1968).
6. I. E. Sutherland, "SKETCHPAD—A man-machine graphical communication system," *AFIPS Conference Proceedings, Spring Joint Computer Conference* **23,** 329–346 (1963).
7. H. B. Baskin, "A comprehensive applications methodology for symbolic computer graphics," *Pertinent Concepts in Computer Graphics,* University of Illinois Press, Urbana, Illinois (1969).
8. W. H. Ninke, "Graphic I—A remote graphical display console system," *AFIPS Conference Proceedings, Fall Joint Computer Conference* **27,** Part I, 839–846 (1965).
9. A. Appel, T. P. Dankowski, and R. L. Dougherty, "Interactive graphics in data processing: Aspects of display technology," *IBM Systems Journal* **7,** Nos. 3 & 4, 176–187 (1968).
10. *TSS/360 Concepts and Facilities,* System/360 Reference Library, C28-2003, International Business Machines Corporation, Data Processing Division, White Plains, New York.
11. G. Gordon, *System Simulation,* Prentice-Hall, Inc., Englewood Cliffs, New Jersey (1969).
12. R. W. Jensen and M. D. Lieberman, *IBM Electronic Circuit Analysis Program; Techniques and Applications,* Prentice-Hall, Inc., Englewood Cliffs, New Jersey (1968).
13. *1130 Continuous Systems Modeling Program,* H20-0209-1, International Business Machines Corporation, Data Processing Division, White Plains, New York.
14. J. McCarthy, et al., *LISP 1.5 Programmer's Manual,* The MIT Press, Cambridge, Massachusetts (1966).
15. C. J. Evangelisti and S. P. Morse, *Graphical Modeling Using Contextually Implied Functions,* RC 2280, IBM Thomas J. Watson Research Center, Yorktown Heights, New York (1970). Accepted for publication in the *Computer Journal.*
16. J. C. Gray, "Compound data structures for computer-aided design," *Proceedings of the 22nd National Conference of the Association for Computing Machinery* **P-67,** 355–365 (1967).
17. J. A. Feldman and P. D. Rovner, "An Algol-based associative language," *Communications of the ACM* **12,** No. 8, 439–449 (1969).
18. A. J. Symonds, "Interactive graphics in data processing: Auxiliary-storage associative data structure for PL/1," *IBM Systems Journal* **7,** Nos. 3 & 4, 229–245 (1968).

Appendix A: Block diagram definitions

DESIGNPAD is equipped to handle a large class of block diagrams:

Definition: A block is a pair (b, A) where b is called the block name and A is an ordered set called the attacher points.

Definition: A labeled block diagram is a quadruple $(B, \mathfrak{A}, E, T)$ of sets and two relations L and H. The elements of B are blocks and $\mathfrak{A}$ is the set of all attacher points of the blocks in B. E is called the set of endpoints, and the elements of T are called text. The relation $L : \mathfrak{A} \cup E \rightarrow \mathfrak{A} \cup E$ yields a set of ordered pairs called the *lines*, and $H : T \rightarrow E \cup \mathfrak{A} \cup B$ yields the hooks.

These definitions introduce more concepts than absolutely necessary, but they correspond with the notions discussed in the paper. Blocks, block names, endpoints, and attacher points correspond exactly. The attacher points are ordered simply to distinguish them. The two relations indicate connections—if a1 and c4 are attacher points, and L(a1) = c4, then a1 and c4 are connected by a line. Similarly if ttt is a text, b is a block, and H(ttt) = b, then ttt is related to b by a hook.

Appendix B: Modeling subsystem facilities

Listed here is a more detailed description of the facilities provided by the modeling subsystem. These facilities include the drawing package and the control commands necessary to construct and analyze a model.

Facility	*Control Command*	*Description*
Start-up	Cold start	System initializes with four predefined viewports, displaying the library sheet, the light-key sheet, and two blank sheets.
	Warm start	System starts up in same condition as left by this user.
Control	Viewport size	Upon selection of the appropriate light key, any selected viewport boundary will move with the light pen.
	Display sheet	Select a block on the library sheet and a point in a viewport. The sheet is displayed in that viewport.
	Window	Select viewport and appropriate light key, and a special windowing control appears on the screen. Using the light pen on this control moves the sheet continuously in the viewport.
	Clear sheet	Selecting a light key and a viewport clears the specified sheet.
	Analyze	Selecting appropriate light key and entering an identifier through the alphanumeric keyboard causes that application program to operate on the model in the modeling viewport.
	Display/blank hooks	A light key suppresses all the hooks. A similar command redisplays them.
	Move	Selecting a light key and pointing at a block in the modeling space moves the block along with all lines or blocks attached to it under the pen. The lines stretch or contract as needed.

Facility	*Control Command*	*Description*
Drawing	Draw line	Placing the pen at a point in the modeling viewport and confirming causes a flashing point to appear at this position. Moving the pen away causes a line to be drawn from that point to the pen position, which can then be moved to the desired location and confirmed. In the "rubber-band" mode in which the line follows the pen, the line on the screen is a single, long stroke. Upon confirmation, the line is broken up and entered into the cellular display file.
	Copy block	Pointing at a block in the reference viewport and confirming causes the block to flash. Moving the pen in the modeling viewport causes a copy of the block to follow the pen, and as in "draw line", the block may be placed as desired.
	Erase line/block	Selecting an isolated point in the reference viewport and a line or block in the modeling viewport erases that line or block. An unconfirmed erase blanks out the beam (the display file is essentially unchanged, but instead of actually drawing the line, the electron beam is simply moved without drawing), whereas when the erase is confirmed, the cellular display file is modified. In addition to single line/block erase, selecting a point in common between several lines or blocks erases all of them.
Text editing	Begin text	Selecting a point in the modeling space and entering any alphanumeric character displays that character at the point; text mode is entered (thus all characters entered are displayed) and a left margin is established automatically.
	Edit text	The usual text-editing facilities are provided.
	Delete text	Selecting any isolated point in the reference space and specifying a character in a textual entity erases the entity.
	Relate text	Selecting a textual entity and a line or block or attacher point creates a hook from the point to the text.
	Erase hook	Selecting an isolated point in the reference viewport and a hook erases the hook.
Block definition	Specify shape	Selecting a light key causes the shape specification mode to be entered. A new picture is brought on the screen, consisting of a large area for drawing a block, an area in the corner where the block is reproduced as it is drawn in actual size (i.e., it is scaled down), and several light keys for use in drawing.
	Draw	This mode is entered automatically permitting lines to be drawn between any two points in the drawing area. In addition, attacher points can be specified with the use of the light keys.
	Label block	Touching the appropriate light key permits a block to be labeled with an alphanumeric character string for use in the application program.
	Specify function	After shape specification, a light key returns the user to the modeling phase, with the sheet corresponding to the newly defined block in the modeling viewport. The implied drawing package may now be used to draw a model that describes the function of the new block. The new block appears on the library sheet in the reference viewport, and thus recursive definitions are permitted.
	Specification complete	A light key completes this phase.
Sign off	Save	A light key causes all models and blocks defined at this session to be saved under the user's name.
	Quit	Shuts down the system.