DATA STRUCTURES FOR COMPUTER GRAPHICS

Marshall D. Abrams

National Bureau of Standards

ABSTRACT

This paper introduces data structures as applied to computer graphics. Design criteria for computer graphics data structures are discussed, followed by a comparison of general-purpose and tailored graphic data structures. A general graphic data structure is introduced as an example of a structure meeting the preceding criteria. The L^6 language is then examined as a tool for implementing the above data structure, and is compared to a few other language systems.

INTRODUCTION

"Computer graphics" is the general term applied herein to the use of a digital computer to form an internal model representation of an externally perceived graphical entity. The objective of such modeling is to extract information from the graphical entity so that it may be modified, manipulated, or otherwise processed. A short computer graphics reading list is provided at the end of this paper.

After graphical information has been digitized, an organization must be provided for the storage and retrieval of this data within the memory of the computer system. The linear array is the simplest scheme available, but it is not of interest within the scope of this paper even though it enjoys wide use in certain classes of applications. Rather, this report will be directed to a discussion of those organizations which represent the relationships among the components of the graphical entity. The relationships which must be represented within this subset of computer graphics include topology and dependency relationships. It is most convenient to represent such information in a hierarchical data structure which, in some abstract way, models the external graphical entity.

Historically, this subset has been restricted to the study of line drawings, but gray-scale and color representations are currently under investigation. The purpose of the data structure is to facilitate the extraction of intelligence and manipulation of both the image and the information it represents.

A graphical image is a preferred medium of human communication because of the possibilities inherent for maximum information transfer with minimum effort. Graphical communication is often highly stylized, requiring significant training for both the generation and interpretation of images. Conventions are established, which may vary from one application field to another, which enable precise communication with a minimum effort and non-graphical information.

Since graphical communication among humans is such an easy and effective technique, effort has been expended to extend this communication to digital systems. As with other languages which are close to human language and far from machine language, considerable resources must be allocated to store and manipulate the graphical communication within the digital system.

OVERVIEW OF GRAPHICAL DATA STRUCTURES

Both medium and conventions present difficulties associated with the digital representation of graphical information. The significant attribute of the medium problem is dimensionality. Pictures deal with two dimensions, sometimes representing three dimensions. High speed primary memory is usually addressed in a piece-wise linear fashion, therefore a transformation is required to map the graphical structure into a one-dimensional frame. The information describing a graphical entity must be stored in such a way that it can be retrieved, manipulated, edited, and used to produce the desired graphical image.

The pertinent information associated with a graphical entity frequently consists of the geometrical description of the graphical item: scaling, position, and orientation data; relationships and connections to other items; name and identification of the item; graphical constraints on the item itself and on its relationship to other items; and non-display (textual) data intimately connected with the graphical entity.

One of the first decisions to be made in designing a graphical data structure is whether it should be completely general, or tailored to a specific application. The general fixed structure format is usually inefficient of storage since it must provide for unused options[1]. Furthermore, no matter how general the fixed structure was designed to be, there always exists the pathological case which exceeds the capability of the structure. In such a case, one has no choice but to redesign the structure, hopefully maintaining upward compatibility.

The tailored structure meets all of these objections, but necessitates the effort of construction. In fact, the existence of general-purpose structures is extremely helpful to the user interested in tailoring a structure to his application. The intellectual effort implicit in the general structure may simply be transferred to the tailored structure, simultaneously modifying the structure to meet the current objectives. Since most graphical data structures are pointer-type structures, with such pointers being explicit or associatively addressed, the presence of a language designed to work with such pointers greatly facilitates the construction of a data structure.

While it is certainly possible to build the entire graphical data structure up from scratch, the use of a list processing or associative processing language greatly simplifies the work. In fact, most of the literature ostensibly devoted to computer-aided design is in fact concerned with data structure. Particular attention is called to references (1), (2), (3), (9), (11) and (12) where the salient features of new and more established "service" languages are discussed.

The next problem to be considered involves the communication of data to and from the graphical data structure. This data will often be involved in the process of drawing a picture, in modifying an existing picture, or in re-drawing a picture from the data structure. The use of computer graphics in facilitating the convenient use of computers requires that this information transferal process not be a burden on the human user[25]. Delay that is annoying to the user should be avoided; a measure of tolerable delay is the user's concept of the difficulty of the task. While the design of graphical processors is not within the scope of this survey, one cannot ignore the hardware requirements forced by the desire for rapid response.

The mode of operation is for the user to communicate his desires to the digital system, often using graphical input devices such as light pen joystick, and mouse; and for the system to respond by displaying the desired picture on his CRT display. Simultaneously the data structure needs to be updated to reflect the changes resultant from the CRT activity. Rapid response criteria require that at least part of the data structure must be instantaneously accessible to the user at the terminal[5,6]. The size of the local processor and the capacity of the communication line with the main system determine the extent of the local image of the complete data structure.

A final problem is concerned with storage capacity in both the display driving computer and in the central system. The portion of the data structure represented in the local memory is usually less than that stored in the central system and may reasonably be restricted to that portion of the structure being displayed. If additional storage is available, this may be augmented by logically adjacent substructures possibly selected by an anticipatory mode algorithm.

Storage restrictions in the central system require greater attention and careful study. At the time that a structure is created or modified, it is necessary that the storage used not be the limitation upon the process. Thus, high speed core storage is required. Considering the possible extent of

graphical data structures and the core use limitations imposed by the operating system environment, it is quite reasonable to expect and provide for the possibility of insufficient core storage being available. The solution exists in the form of a paging scheme, but careful attention and re-examination must be paid to the design and handling of the paging[1, 2, 3, 13].

GENERAL GRAPHIC DATA STRUCTURE

The concepts and techniques of graphical data structures will be introduced here in the form of an example structure. This structure will purposely be kept at a level comprehendible to human users and modifiable by them. It is not intended that this presentation be exhaustive, but rather typical and hopefully educational. The organization of the data structure presented here is an explicit referencing structure, similar to that of Cotton and Greatorex[5], the GRAPHIC-2 system[4, 7] and GRASP[8]. The structure is certainly not exclusive; additional examples are given the surveys by Gray[9], Dodd[21], and Hamilton[27]. The structure will be presented in the form of a directed graph; the mechanism of representing such a structure in a computer memory will be discussed subsequently.

In an effort to minimize the amount of graphics terminal machine language programming required, especially by users that are not interested in such a level of detail, the commonly used picture-building elements are provided as building blocks called "basic subpictures". A basic subpicture may be a single command to draw (display) a point, line, or conic section; it may also be a sequence of such commands to draw a commonly used geometric entity. Such basic subpictures are often in the form of the "frame" or "skeleton" of an open subroutine, because the essential positioning information must be supplied by each reference to (use of) the basic subpicture. The basic subpicture data block must also contain the identification and relative location of the externally accessible terminals of the subpicture, such terminals being the parts of points of connection to the subpicture by the greater, outside world. If there are to be constraints on the use of the basic subpicture, these constraints must also be contained within a block pointed to by the basic subpicture data block.

Within the context of basic subpictures lie all of the characters and geometric figures directly presentable with a single machine level command, these constitute the "hardware character set". Commonly used graphical entities may also be coded in graphical control language as a short program, and provided as a time-saving service. In addition it is desirable for a user to be able to define his own subpictures which are useful for his application. The definition of a subpicture is usually not within the scope of the graphical language-data structure being herein described, but is at the level of graphical control language.

Thus, a picture is the highest level, encompassing all lower constituent levels. These lower levels being collectively referred to as subpictures. The lowest level is termed a basic subpicture in that it is the only level which references the display.

Using graph terminology, each basic subpicture is a node in the directed graph which is the graphical data structure. Although it is quite possible for the node block to be of variable size, herein pointers will be used to reference variable sized data segment blocks. Under these conventions, a basic subpicture node can appear as in figure 1.

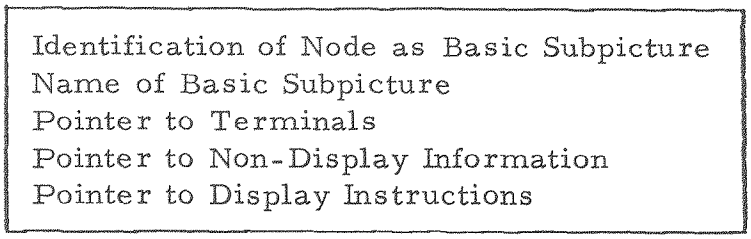


Figure 1. Basic Subpicture Node

Since a basic subpicture does not possess any absolute frame of reference, it cannot in and by itself cause any display. It must be referenced by a higher node to be used as a display item. These higher nodes will in general reference multiple lower nodes, thus building a picture out of previously created components. The terms "higher" and "lower" relate only to position in a directed graph drawn to describe the structure and might instead be termed "subsequent" and "prior" respectively.

The construction of a picture is described by a directed graph wherein the top node is the picture, the intermediate nodes are subpictures, and the terminal nodes are the basic subpicture. A sample graphical data structure is represented in figure 2.

While it is fairly obvious that when expressed in computer code the picture and subpicture nodes need to be represented by data blocks, it may not be immediately clear that the same is true for the branches connecting the nodes. There is of course a trade-off between the information associated with the node block and that associated with the branch block. The following selections are somewhat arbitrary although typical[7]. The subpicture node block will essentially consist of pointers to other information containing blocks. Among these blocks is the branch block, which deserves special mention.

The branch block contains the necessary transformation on the lower nodes to incorporate them into the subpicture defined by the subpicture node. Certain blanks in the skeleton frame of the lower level picture must be completed, and other parameters may require systematic modification. The transformation information consists essentially of displacement, scale, and rotation information.

Since all nodes except the highest "picture" node are intrinsicly referenced to zero, a new reference displacement must be provided for each instance of use of subpicture and basic subpicture node. This displacement

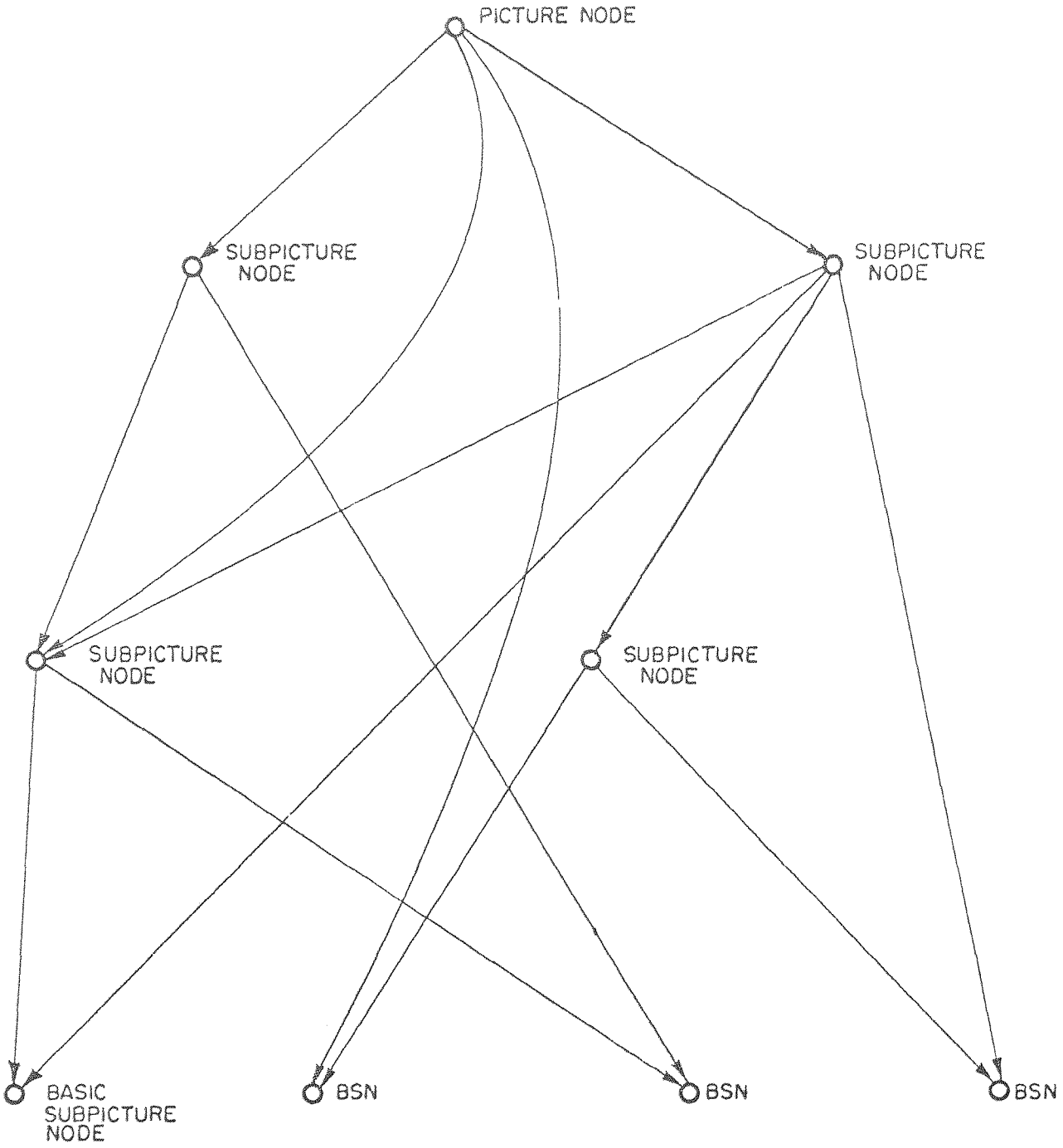


FIGURE 2. SAMPLE GRAPHICAL DATA STRUCTURE

may occur in a virtual display space having no size relationship to the display viewing area; therefore, an additional displacement calculation may be required in the process of display generation when the segment of the structure is selected for display.

Scale and rotation information must also be provided when constructing the present subpicture out of lower level items. In most applications it is unlikely that the rotation information will require changing after picture-generation time, but scale may be a continuously varying parameter. The analogy of photographic enlargement is accomplished by a modification of the scale parameter. In certain applications this operation could be the critical item in operation of the facility.

The connectivity of the subpicture constituting the present level subpicture must be separately treated. It is not sufficient to build a picture by appropriately placing subpictures so that their terminals coincide. Subsequent parameter change in the branch blocks, or minor malfunction in the hardware, could easily destroy such coincidence. Thus, there must be explicit provision for an ordered relationship among the terminals of subpictures associated within a higher level picture.

For this reason, another block is provided, this being the "connector" block. The connector block is a specific example of a constraint. It is presented as a separate block by virtue of its prevalence. The connector block must identify the lower level terminals to be connected, and it must point to the constraints on such connection. These constraints might include a requirement for coincidence. If the terminals being connected were not coincident, the connector block would be required to provide a line to form the connection. This line in turn could have attributes such as intensity and rate of blink.

The blocks described above may be represented as follows:

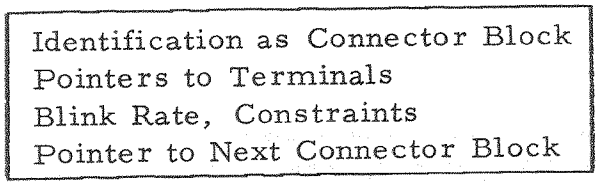


Figure 3. Connector Block

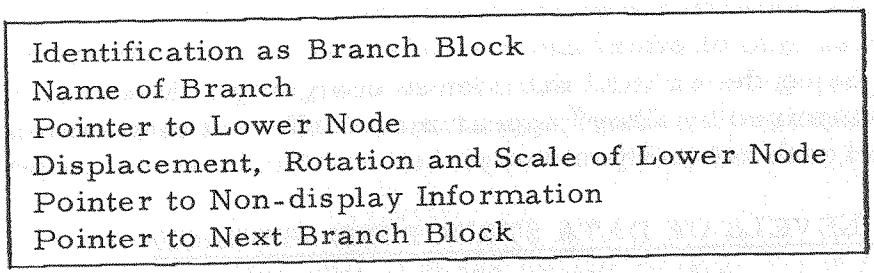


Figure 4. Branch Block

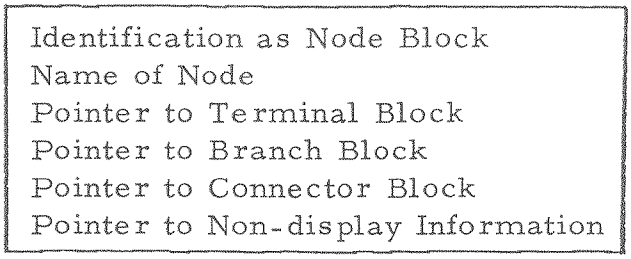


Figure 5. Node Block

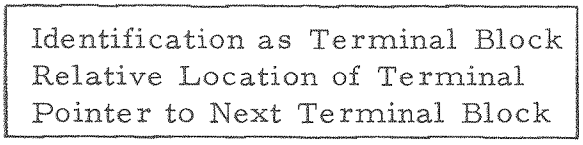


Figure 6. Terminal Block

The design of the pointer scheme is a critical part of any data structure. An excellent discussion is given by Dodd[22], to which the reader is referred. The most simple pointer structure is the single linked list wherein each block contains a pointer to the succeeding block in the list. The pointer field in the terminal block contains a special symbol known as the "null pointer" indicating the termination of the list.

The major shortcoming of the single linked list is the inability to return to the head of the list without having previously saved the location of this head in a well known location to which reference might be made.

The single linked list is rarely employed simply because a ring structure may be obtained by having the end of the list point back to the head. Of course, the head and tail must be suitably flagged to avoid endless ring-chasing. Another way of returning to the head of the list is to use a doubly-linked list possessing a backward as well as a forward pointer, but this involves twice as many pointers. It is, on the average, twice as fast as the single linked list in returning to the head of the list.

One purpose of rings or doubly-linked lists is to be able to return to the head of the list, or the list pointer, when a success or failure has occurred; the second pointer which the doubly-linked list requires is often replaced by a back pointer to the head of the list. To conserve space, the pointer to the head of the list may not occur in every block, but rather in strategically placed blocks.

A great deal of effort has gone into the development of pointer arrangements, this being the critical decision in designing a data structure. The structures examined by Gray[9] appear more different than similar, yet they are all concerned with related problems.

MULTIPLE LEVELS OF DATA STRUCTURE STORAGE

The requirements of fast response to operationally complex requirements and possible large data structures necessitate that the structure be simultaneously maintained in more than one level of storage, at least in part. First, consider the question of storage in the display.

Early graphical displays required the exclusive service of a large computer system. Today the trend is to provide a small computer as the local service to each display and to service this local processor-graphical terminal from the central system only when necessary. There is a whole spectrum of capabilities of local processor attached to graphical processors. We shall not go into the evolution of such dedicated processors here; the situation has been stated elsewhere[17]. Needless to say, however, the extent and kind of representation of the graphical data structure in the local computer is highly dependent upon the kind(s) and amount of storage available, the instruction repertoire, and the speed of the local processor.

The minimum information to be kept in the local processor is the display list which directly controls the picture presented. The display list is extremely machine-dependent, containing the necessary machine instructions to generate the display. If the computing capability of the local processor is non-existent or extremely minimal it may be necessary to construct the display list in the main system for transmission to the graphical display. In such a case the local processor fulfills only the function of refreshing. In this situation it is impossible to reference the graphical data structure via the display image because the display list has been generated only for display purposes.

In systems with minimum local processing ability, or even in more substantial systems, it seems a waste of an expensive resource to store the display list in randomly addressable core storage. It appears a better allocation of resources to use rotating storage for the display list. Not only does this free core storage for programs, but it makes it possible to carry on display refresh as a parallel process. Recent[18, 19, 20] and not so recent[21] systems have used this approach.*

The next step is to provide an association between the display list and the graphical data structure. Such as association requires a referencing technique from the display list back to the data structure. A pointer scheme can be implemented, but difficulty occurs as to the subpicture level to be pointed to. Under various conditions the user at the graphical terminal might be interested in pointing to a picture, a level of subpicture, or a basic subpicture. An automatic safe technique is to have the pointer go to the highest level of subpicture being referenced, with the user being able to initiate pointer chasing under his control to reach the desired lower level.

If memory and speed allow, part or all of the graphic data structure may be contained in the local processor. If sufficiently large and fast, the local processor could contain the entire data structure, generate its own display list, and reference the main system only for archival purpose or for linking to other subsystems. In this form of operation the graphical subsystem can be considered a "sketchpad" on which various trial drawings are made.

---

*But a cycling display carries with it three prices to pay: (1) it is usually slow to access, (2) it is fixed in size (but the size can be very large), and (3) it can be quite difficult to respond to light pen interactions.

When an acceptable one is produced it can be preserved by referring it to the central computer.

In general, the data structure will be too vast to be contained completely in the graphical terminal. A compromise is then effected wherein part of the data structure might be transported as needed between the central system and the graphical subsystem. The degree of compromise is a function of the processing capability of the graphical terminal, a subject well discussed by Myer and Sutherland[17]. For convenient operation the transmission must occur within the user's wait tolerance. When only part of the data structure is resident in the graphical subsystem, extreme care must be taken with the pointer to the non-resident parts of the structure. There must be a mechanism for flagging references to non-resident items; there must be a mechanism for enlarging or contracting the size of the portion of the data structure available. These problems are quite akin to the multiple-level storage problem in the central system, which shall be discussed next.

The adage of "a picture being worth a thousand words" is magnified in computer representation. It can easily require many thousand words to store a moderately complex picture. Such storage requirements can easily consume available high-speed primary memory.

It is certainly possible to design the driving and service program, the "resident system", into minimally-interacting modules. These modules can be brought into core as pages[1, 12] or overlays[13], thus reducing the core storage which must be devoted to the system.

For user-created programs and data structures the situation is different. It is desirable that as few restrictions as necessary be placed on the programmer. Therefore, systems are written which automatically assign program and data to storage pages[1, 3, 13]. However, there is not total rigidity in these page assignments; variable-sized pages[1] and partial user control[3, 12] helps to adapt the system to its current use. For the storage of the graphical data structures it is even more important that the system be given as much information as is available. With complete information it is possible to implement valid anticipation of program needs[3].

SAMPLE USE OF GENERAL GRAPHIC DATA STRUCTURE

Representing a graphical data structure on paper is an awkward necessity. Awkward because the confines of standard paper size makes it an exercise in topological ingenuity on the part of the writer and parallax error elimination on the part of the reader. Necessary because the expository approach alone generally produces an incomplete information transferal.

The first illustration, in figure 7 is of the structure representing a triangle. This is a trivially simple structure involving only one node and one basic subpicture. The one node, which is automatically the top (picture) node points to three rings: branch, terminal, and connector. Each block in the

branch ring contains a pointer to the basic subpicture used, namely "point", the X, Y coordinates of the instance of that point, and a pointer to the next branch block and back to the node block. Each terminal block contains the coordinates of the terminal relative to the origin of the subpicture (which in this case are selected to be all the verticies), and the forward ring pointer. Each connector block contains a pair of pointers to the terminals being connected, and the forward ring pointer. All of the remaining fields contain zeroes interpreted as null pointers. The organization of the blocks is in conformance with figures 1, 3, 4, 5, and 6.

Let us now use this triangle as a subpicture in building a larger picture. As an example, consider the hexagon shown in figure 8(a). The triangle used of the subpicture is assumed to have been drawn in the position shown in figure 8(b). Note that external terminals are denoted by small circles in these figures.

The data structure of the hexagon is drawn in figure 9. Included is the terminal block ring of the triangle data structure which is necessarily referenced by the connector ring of the hexagon. Note the dotted lines representing pointers from the connector ring of the hexagon to the terminal ring of the triangle.

These dotted lines from connector blocks to terminal blocks associated with another node block are representations of an amazingly complex pointer chasing mechanism required. The connector block must point to the branch ring which it accesses by pointing to the branch pointer in the node block. From the appropriate branch block it obtains a pointer to the node block of the subpicture references. Also from the branch block it obtains the displacement, rotation, and scale data which is necessary for the calculation of location of the desired terminal in the particular instance of use. From the node block pointed to by the branch block it obtains the pointer to the terminal ring associated with that level of subpicture. Finally, that terminal ring is traversed until the particular terminal desired has been acquired.

Since such pointer chasing is not an abnormality in graphic data structures, there must be a mechanism easing such constructions. The pointer concatenation facility of L$^{6\ 23}$, recently implemented by R. A. Siegler in conversational form as CL6^{24}, is one technique which facilitates such pointer chasing.

THE TAILORED GRAPHICAL DATA STRUCTURE

As discussed in the overview, it is frequently convenient to construct a data structure which is tailored to the graphical image to be modeled. The tailored structure can eliminate those features of the general graphic data structure which do not apply to the problem at hand. The structure of the graphical entity may be taken into account in designing the tailored structure; conditions which were provided for in the general case may not occur. Therefore, the space reserved for the eventualities in the general structure could be released for other use in a tailored structure.

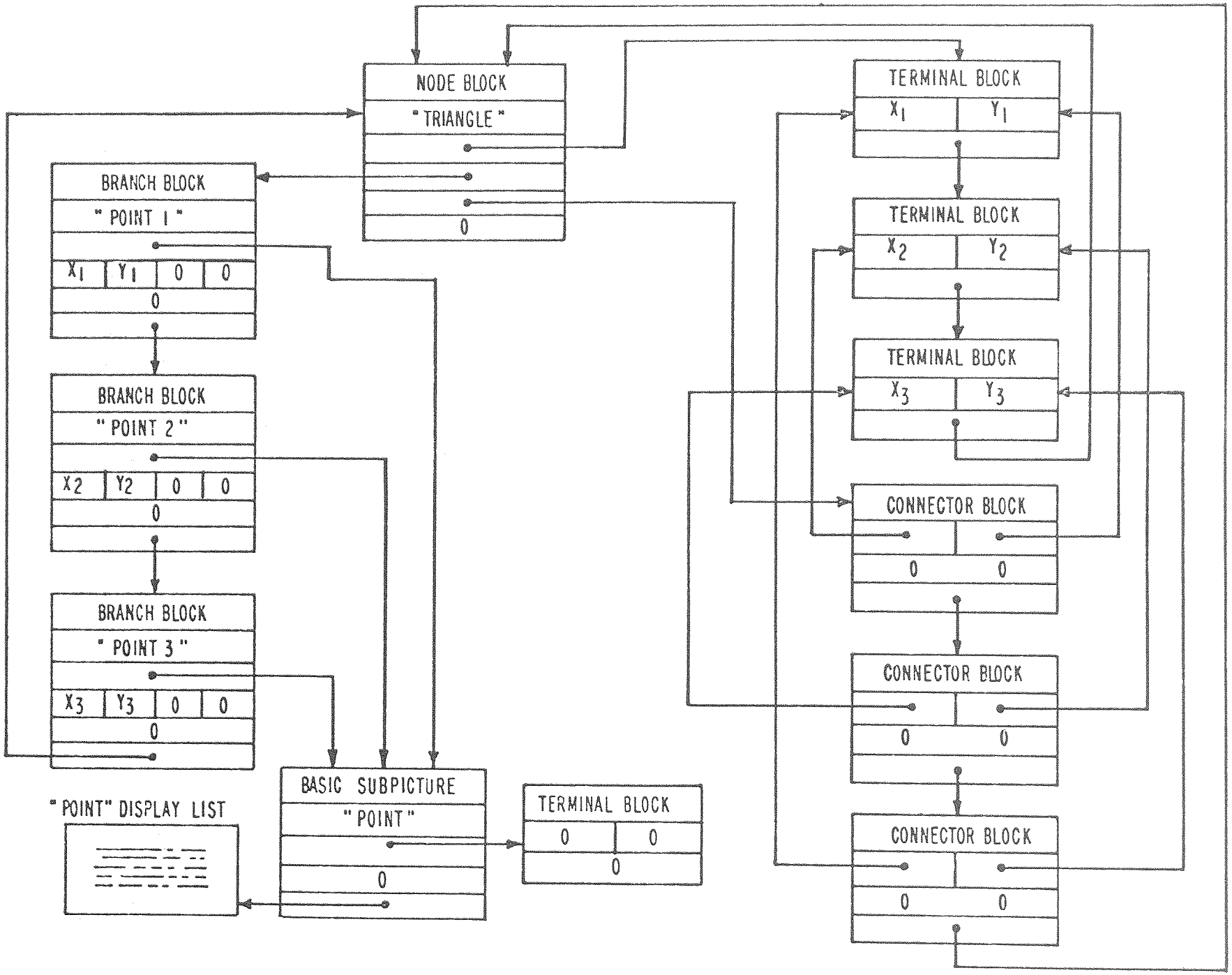
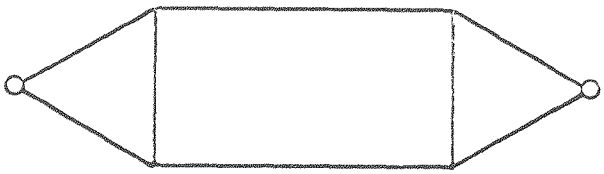


FIGURE 7 DATA STRUCTURE OF A TRIANGLE

(a.)



(b.)

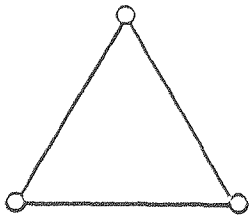


FIGURE 8 HEXAGON. (a) COMPOSED FROM TRIANGLE. (b) TERMINALS DENOTED THUS: (—o)

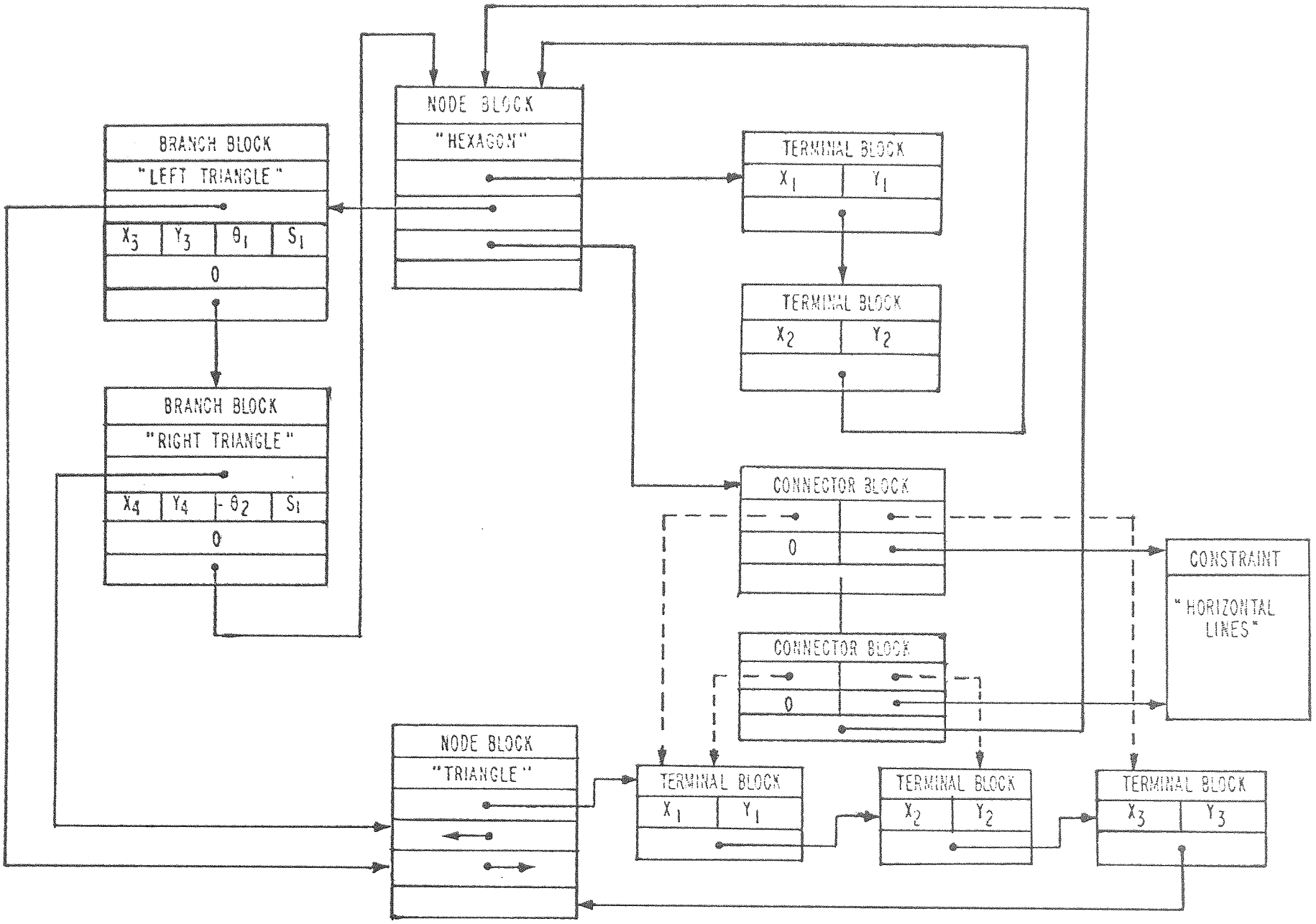


FIGURE 9 DATA STRUCTURE OF HEXAGON

Fixed format data structures and the language systems supporting them[5, 15, 26, 28, 29] have been first developed as the solution mechanism to specific problems. Indeed, since these languages were designed with a class of problem applications in mind, the user might be unaware of the potential data structure restrictions.

Looking back, it is possible to contrast these fixed structure systems to produce a generalized set of criteria as has been done herein. For new applications, however, it appears that a data structure language is the better approach. Using a language designed to manipulate data structure gives the user increased freedom and power.

DATA STRUCTURE LANGUAGES

Several groups have developed languages for the manipulation of graphic data structures (among other applications). One family, which will be explored herein, is that derived from L^6 [23], including DSPS [3, 12]; CL6 [24]; and XL6. XL6 is currently under development by the author, being a compiler, written in FORTRAN and therefore to a certain extent portable, currently producing relocatable binary code for the UNIVAC 1108 which has compatible subroutine linkage with FORTRAN. It is this author's opinion that use of a variety of specialized languages offers advantages over one all-purpose language.

L^6 is a low level language in that the programmer must be aware of words and bits, but it possesses the higher level characteristic of a set of run time subroutines which support its operation. Being low level, L^6 expresses data structures very well for programmer visualization.

One useful feature in L^6 is the ability to define the location of a "field" within a "block" of consecutive computer words. A field, once defined, may contain a pointer to another block, an arithmetic value, or anything else the programmer desires. A field is designated by a single letter name. The complete specification of a field includes the word in the block in which the field is to exist, the name of the field, and the inclusive bit boundaries constituting the field within the computer word.

The format of the field definition command is

(<word>, D <field name>, <bit bound 1>, <bit bound 2>)

where <word>, <bit bound 1>, and <bit bound 2> are all integers indicating the relative word in the block, and the inclusive bit boundaries within that word. <field name> is the single letter name by which the field is symbolically referenced. XL6, like DSPS, permits multi-character names, thus permitting association of name with function for an increase in readability and visualization.

Like most programming languages, L^6 provides the programmer with a means for inserting comment lines for internal documentation. In L^6 such comment lines must contain an asterisk in column one and are ignored by the L^6 translator.

The ability to define those fields appropriate to the specific application is only one advantage of the tailored data structure. Another advantage of this tailored data structure is the ability to define variable size blocks, the size being determined during program execution. This feature permits optimum memory utilization. The meaning attributed to field contents is the programmer's responsibility. Note also that field definitions are nonunique; it is likewise the programmer's responsibility to use the field definition he requires from the set created.

CONCLUSION

Having presented design criteria for graphic data structures, a sample data structure meeting these criteria, and a language for implementing the designed data structure, the picture is not complete. Completion requires an implementation, with interactive modification of the data structure and perhaps extension to the language. The author hopes to enjoy the pleasure of this interaction in the near future.

REFERENCES

1. Feldman, J.A. and Rovner, P.D., An ALGOL-Based Associative Language, Comm. ACM 12, No. 8, 439-449 (Aug. 1969).
2. Rovner, P.D., and Feldman, J.A., The Leap Language and Data Structure, Proc. IFIP Congress 1968, C73-C77.
3. Evans, D. and Van Dam, A., Data Structure Programming System, op. cit., C67-C72.
4. Ninke, W.H., A Satellite Display Console System for a Multi-Access Central Computer, op. cit., E65-E71.
5. Cotton, I.W. and Greatorex, F.S., Data Structures and Techniques for Remote Computer Graphics, Fall Joint Computer Conference, 1968, 533-544.
6. Kulsrud, H.D., A General Purpose Graphic Language, Comm. ACM, 11, No. 4, 247-254 (April 1968).
7. Christensen, C., and Pinson, E.N., Multi-Function Graphics for a Large Computer System, Fall Joint Computer Conference, 1967, 697-711.
8. Thomas, E.M., GRASP--A Graphic Service Program, Proc. ACM National Meeting, 1967, 395-402.
9. Gray, J.C., Compound Data Structure for Computer Aided Design: A Survey, op. cit., 355-365.
10. Wexeblat, R.L., and Freedman, H.A., The MULTILANG On-line Programming System, Spring Joint Computer Conference, 1967, 559-569.
11. Van Dam, A., and Evans, D., A Compact Data Structure for Storing, Retrieving, and Manipulating Line Drawings, op. cit., 601-610.
12. Evans, D. and Van Dam, A., Data Structure Programming System, IBM document number 360 D 06.8.003.

13. Bobrow, D. G., and Murphy, D. L., Structure of a LISP System Using Two-Level Storage, Comm. ACM, 10/, No. 3, 155-159 (March 1967).
14. Sutherland, W. R., The On-Line Graphical Specification of Computer Procedures, Ph. D. Dissertation, M. I. T. (Jan. 1966).
15. Kantrowitz, W., CORAL Macros--Reference Guide, Lincoln Labs. (1965).
16. Feldman, J. A., Aspects of Associative Processing, CFSTI AD 614-634 (April 1965).
17. Myer, T. H., and Sutherland, I. E., On the Design of Display Processors, Comm. ACM, 11/, No. 6, 410-414.
18. Rippy, D. E., MAGIC II - Graphical Display Terminal Interfaced to a Digital Computer, Computer/Display Interface Study, Final Report, AD 699366 (April 1969).
19. Gear, C. W., An Interactive Graphic Modeling System, Dept. of Computer Science, Univ. of Ill. Report No. 318 (April 1969).
20. Hostovsky, R., Design of a Display Processing Unit in a Multi-Terminal Environment, op. cit., Report No. 343 (July 1969).
21. Rippy, D. E. and Humphries, D. E., MAGIC -- A Machine for Automatic Graphics Interface to a Computer, Fall Joint Computer Conference, 1965, 819.
22. Dodd, G. G., Elements of Data Management Systems, Computing Surveys, 1, No. 2, 117-133 (July 1969).
23. Knowlton, K. C., A Programmer's Description of L^6, Comm. ACM, 9, No. 8 (Aug. 1966).
24. Siegler, R. A., The CL6 Conversational List Processing System, Computer/Display Interface Study, Final Report, AD 699366 (April 1969).
25. Miller, R. B., Response Time in Man-Computer Conversational Transactions, Fall Joint Computer Conference, 1968, 267-277.
26. Sutherland, I. E., SKETCHPAD: A Man-Machine Graphical Communication System, Spring Joint Computer Conference, 1963.
27. Hamilton, J. A., A Survey of Data Structures for Interactive Graphics, CFSTI, AD 706 706.
28. Sutherland, I. E., "The CORAL Language and Data Structure," Adams Associates Computer Display Review, Vol. 1, Section II.
29. Ellis, T. O., Heafner, J. F., and Sibley, W. L., The Grail Ring Structure and Primitives, CFSTI, AD 706 715.

A SHORT COMPUTER GRAPHICS READING LIST

Bowman, S., and Lickhalter, R. A., "Graphical Data Management in a Time-Shared Environment," Proc. 1968 SJCC, 353-362.

Kennedy, J. R., "A System for Time-Sharing Graphic Consoles," Proc. 1966 FJCC, 211-222.

Prince, M. D., "Man-Computer Graphics for Computer-Aided Design," Proc. IEEE, 54, 12, 1968-1708 (Dec. 1966).

Hobbs, L. C., "Display Applications and Technology," Proc. IEEE, 54, 12, 1870-1884 (Dec. 1966).

Machover, C., "Graphic CRT Terminals -- Characteristics of Commercially Available Equipment," Proc. 1967 FJCC, 149-159.

Abzub, I., "Graphic Data Processing," Datamation, January 1965, 35-37.

Wylie, C., Romney, G., Evans, D. and Erdahl, A., "Half-tone Perspective Drawings by Computer," Proc. 1967 FJCC, 49-58.

Licklider, J. C. R., "A Picture is Worth A Thousand Words -- and it Costs . . . ," Proc. 1969 SJCC, 617-621.

Davis, S., "Display Processing Subsystems for Computer Data Displays," Computer Design, Vol. 8, Number 5, May 1969, pp 50-55.

Johnson, C. I., "Principles of Interactive Systems," IBM Systems Journal, Vol. 7, numbers 3 & 4, 1968.

Parker, D. C., "Solving Design Problems in Graphical Dialogue," On-line Computer Graphics, 1966, McGral-Hill.

Van Dam, A., A Survey of Pictorial Data Processing Techniques and Equipment, distributed by Clearinghouse for Federal Scientific and Technical Information, Number AD 626 155.

Davis, M. R. and Ellis, T. O., "The RAND Tablet: A Man-Machine Graphical Communication Device," Proc. 1964 FJCC, 325-350.

Stotz, R. H., "A New Display Terminal," Computer Design (April 1968), 80-86.

Hargreaves, B. et. al., "Image Processing Hardware for a Man-Machine Graphical Communications System," Proc. 1964 FJCC, 363-386.

Ophir, D. et. al., "BRAB: The Brookhaven Raster Display," CACM, 11, No. 6, 415-416 (June 1968).

Ahuja, D. V., "An Algorithm for Generating Spline-like Curves," IBM Systems Journal, Vol. 7, Numbers 3 & 4, 1968.

Ahuja, D. V. and Coons, S. A., "Geometry for Construction and Display," IBM Systems Journal, Vol. 7, Numbers 3 & 4, 1968.

Appel, A., Dankowski, T. P., Dougherty, R. L., "Aspects of Display Technology," IBM Systems Journal, Vol. 7, Numbers 3 & 4, 1968.

Charfaris, G. J., "Display Technology-Today and Tomorrow," Proceedings of the Society for Information Display, Vol. 10, Number 1, Winter 1969, pp 3-29.

Sutherland, I. E., "Computer Graphics - Ten Unsolved Problems," Datamation, May 1966, pp 22-27.

Teixeira, J. F., and Sallen, R. P., "The Sylvania Data Tablet: A New Approach to Graphic Data Input," AFIPS Conference Proceedings, 1968 Spring Joint Computer Conference, pp 315-321.

Auerbach Corp. & Auerbach Information, Inc., "Special Report-Design and Application of Automated Display System," Auerbach Standard EDP Reports, Vol. 1, 23:120.001, 1969.

Johnson, T. E., "Sketchpad III: A Computer Program for Drawing In Three-Dimensions," AFIPS Conference Proceedings, Vol. 23, 1963.

Desens, R. B., Computer Processing for Display of Three-Dimensional Structures, CFSTI, AD 706 010.