*Present considerations in interactive display activity are brought into perspective.*

*Emphasizing programming aspects, varying approaches are considered with respect to system concepts and peripheral graphics processors, as well as complex data structures, high-level languages, and image-generation techniques. Many of the problems discussed are not unique to graphics systems but are common to interactive systems in general.*

*The requirements for a conversational system to support programmers and application users are listed in the Appendix. An extensive bibliography has been added as basic reference for this issue.*

**INTERACTIVE GRAPHICS IN DATA PROCESSING**

# Principles of interactive systems

**by C. I. Johnson**

The man-computer relationship has been considerably improved during the past few years by new developments in graphic display technology. Nonspecialists can now interact directly with large data processing systems via computer-generated displays. Complex solutions can be compared, adjusted, or improved directly on the graphic display screen. Seen in perspective, however, we are still in the early stages of computer graphics: after years of research, interactive graphics techniques are just now becoming useful in industrial computing applications. The programming support, the main topic of this paper, is no exception; in spite of the emphasis recently placed on it, graphics programming will undoubtedly go through many extensive changes in future years.

definitions

The term *computer graphics*, through use, has become an accepted term to denote that set of computer techniques and applications wherein data is either presented or accepted by a computer in the form of line drawings or graphs. Although the meaning of the word graphics has been stretched, we shall use this popular term as just defined. *Computer-aided design* is another popular bit of "computerese" that denotes the use of a computer as an assistant in the design of some entity. It is generally accepted that this term applies to a give-and-take, or interactive, use of the computer; the designer describes part of his design, performs some analysis procedures, and then, based on the results of the analysis, changes the design and reanalyzes it. We use the term computer-aided design to describe this iterative mode of operation and intend that it imply an interactive or conversational approach to design.

To use computer graphics in computer-aided design in the conversational mode described, it is obvious that the device used for interpretation of graphic input or for the production of graphic output must be reasonably fast. It must be able to present or accept data quickly to maintain an efficient interplay between designer and computer.

As this paper concentrates rather heavily on the relationship of graphics to computer-aided design (or other interactive applications), the cathode-ray tube display with its attendant function keys, light pen (or stylus and tablet[25,85]), and typewriter keyboard is assumed as the device for the transfer of information between user and computer. This device satisfies the stated speed requirements. The techniques pertaining solely to the problem of picture description and structuring of data, as presented in this issue on interactive computer graphics, apply equally well to programs for the use of such noninteractive devices as plotters and drafting machines.

## History of interactive display

For several years the cathode-ray tube (CRT) display console, or terminal, has been used as a computer output device, primarily for the display of curves and graphs. The early Whirlwind computer at Massachusetts Institute of Technology had such terminals for program debugging in 1956. The CRT display console has also been given moderate input capabilities as in the SAGE (Semi-Automatic Ground Environment) air defense system through the use of light guns or light cannons (the forerunners of the current light pen) which were used to allow the operator to direct the computer program to select specific displayed objects as targets.

The first significant demonstration of the use of a display console as an interactive computer input/output device for applications related to computer-aided design was the SKETCHPAD program[82] designed and implemented by I. E. Sutherland. With the SKETCHPAD program, Sutherland demonstrated the use of a CRT display and a light pen to draw pictures while the computer monitored the operator's motions and built a structured set of data representing the pictures being drawn. This data structure was used to represent the topological properties of the drawing. The elements of the drawing displayed on the CRT served as names or labels of elements in the data structure which could be chosen by pointing the light pen at them. Through use of this labeling mechanism, the program allowed the user to extend the data structure to include numerical or other nonpictorial attributes by pointing at the desired element and typing in the additional information desired.

Similarly, relations between elements of the system represented by the drawing could be specified by selecting the elements with the light pen and indicating the relation. These relations could be as simple as (1) the implicit relations involved in the definition of a line in terms of its end points or (2) geometrical constraints such as

limitations of size, parallelism of two lines, or perpendicularity. It was through the use of such constraints that Sutherland provided the user with the ability to draw accurate pictures despite the lack of precision inherent in the use of the light pen.

T. E. Johnson extended this concept with SKETCHPAD III,[47] which provided the user with the capability to describe three-dimensional objects by drawing them in any one of the three orthogonal views or in perspective projection. Both systems provided the ability to modify the drawings by the addition or deletion of elements and allowed the user to scale, rotate, and translate the whole picture or any sets of elements which had been designated as subpictures.

display console function

The series of SKETCHPAD programs demonstrated the two functions of the interactive display console. First, it was used as an input/output device that could accept or exhibit data in both pictorial and alphanumeric forms. Second, it was used to control the sequence of the program. This second function is not peculiar to graphics programs but is common to on-line (or conversational) programs in general; however, the graphics console provides a more efficient means for directing the progress of the program than does the typical on-line typewriter. For example, at any given point in the program, the possible actions that can be requested of the program may be shown as a list of words on the display (referred to as "light buttons" or "light keys")[81] providing some direction to the user of the program. (The list is sometimes referred to as a "menu.") By pointing at the desired action, the user may, simply and quickly, notify the controlling computer program to perform that operation. Thus, drawing or sketching is not the only use of the light pen and CRT combination.

applications

Much attention has been focused on the sketching function in interactive graphics, emphasizing the drafting or computer-aided-design type of application. The range of applications implemented with use of the graphic display console is far wider than this, however, and includes such disparate functions as text-editing,[45,91] conversational mathematics programs,[23,24,27,48,71,72,80] file updating and retrieval,[9] execution traces of programs as dynamic flowcharts,[78] input to system simulators (such as the General Purpose Simulation System and the IBM 1130 Continuous System Modeling Program[7]), the monitoring of the actions of multiprogrammed and time-shared operating systems (such as done at IBM, at Massachusetts Institute of Technology in Project MAC, and at the Control Data Corporation[14]), lens design (ITEK), and displays of the structures of molecules[55] and crystal structures,[61] as well as the more predictable applications such as circuit layout,[32,51,87] mechanical design,[42] part description for numerical control,[11] cartography, and structural analysis.[28,62]

resource sharing

For some time, the major deterrent to the widespread acceptance of interactive graphics as a standard tool in "production," or industrial, computing installations has been cost. The combination of terminal cost and the necessity of dedicating a rather powerful

computer to control a single display prevented computer graphics from being economically justifiable. Because of the nature of the interactive operation, the computer had to spend much of its time waiting for the user of the display to designate what was to be done next. To make the operation economically feasible, it is necessary to share the computer's power and its auxiliary data storage and program storage resources among multiple-display consoles. The alternative, of course, is to have a small, inexpensive computer dedicated to a single user. Unfortunately, many of the programs for which the display console is being used as an input/output device require more computational power than is provided by small processors. Moreover, the programming support available with small computers does not generally provide the more powerful language processors and extensive data file management routines required.

With the introduction of third-generation equipment by computer manufacturers, a new class of operating systems has become available, which allows the resources of a large computer to be shared by more than one user concurrently.[18,22,31,35,38,49,50,56] These operating systems have provided the means for more economical use of display consoles, and a number of industrial computer installations have begun to experiment with graphics data processing techniques in a production environment. Thus, interactive graphics is no longer to be found only in the research laboratory. This has taken a certain amount of courage and vision on the part of those who are now evaluating graphics in the "fly now—pay now" world of industrial computer installations.

Although graphic displays have become more versatile and less costly than were the experimental systems, the equipment still seems to be expensive. Also, trained programmers with graphics experience are not easy to find. But, similar statements were made in relation to computers themselves in the early days. It is being proved that when programmers, engineers, and management learn which applications can actually benefit from the graphic techniques, and when they avoid those applications that are merely glamorous versions of noninteractive problems, then the cost of graphics data processing can be justified.

The movement of graphics from the university and laboratory environment into the industrial computing environment has merely started. The equipment will go through many improvements including display size and precision, the use of video techniques, and color and three-dimensional displays. The programming support, which is the major concern of this paper, will also go through many changes. There are a number of problems that are the subject of current research and development, the adequate solutions of which should improve the cost effectiveness and ease of implementation of applications using computer graphics. Among areas in which these problems arise are systems, data structures, and language.

Much work remains to be done on current operating systems to provide both local and remote interactive display capabilities and

to reduce the overhead (system loading) for each display enough to allow a significant number of displays to be driven by the large "host" computer.

An extremely sophisticated data structure facility is needed for the application of a SKETCHPAD-like approach to very large design problems.[33] Generally, this should provide the ability to maintain this structure on an auxiliary storage device.

Programming of graphic displays is difficult unless a proper set of support programs[1,19] is made available in an easy-to-use form (hopefully, as part of—or accessible from—a high-level language).

This paper explores these problems and some of the possible solutions.

## System concepts

Many of the problems encountered in the creation of a proper support system in which interactive graphics systems may operate are not unique to graphics applications. Rather, these problems are common to all interactive (conversational) systems. They are, however, accentuated by the graphic display console which, because of its ability to display much information rapidly, causes the user to become impatient with any system that cannot provide a response commensurate with its speed. He wants to be assured that the system has not "lost" his request (attention interrupt). At least, the user expects some immediate feedback which tells him that, although the desired result is not yet available, the system has indeed noticed and recorded his request.

multi-access operation

For this reason, the time-slicing[18] approach to time-sharing is not by itself a sufficient solution. Although the ideal graphics system must be time-sliced to ensure all users reasonably equitable and apparently concurrent service, it must also be attention-driven—at least to the degree that no attentions are lost and an immediate acknowledgment is returned on the user's display to indicate that the system has received his request. To avoid the loss of interrupts, the system must be able to record as many interrupts as can possibly occur during the time a user is waiting for his next "turn" or "slice." The application program may then process the signals when the machine resources are again available to the user.

In addition, if at any time the program (user) currently in control of a sharable resource (e.g., CPU) cannot use the resource due to dependency on the completion of some requested service (e.g., I/O), then the system must be able to shift control to some other program (user) contending for the sharable resource. Most current time-sharing and multiprogramming systems are deficient in the implementation of at least one of the above facets of resource-sharing.

Response time is usually stressed as a major measure of the suitability of a conversational time-sharing system. However, there are many other criteria for judging such systems. These criteria are not discussed in detail in this paper as they are well-covered in

the literature concerning time-sharing systems.[18,22,31,49,50,56] The Appendix lists some of the general requirements for a conversational system in which a graphic display console is used for communication between the user (either an application programmer or a nonprogrammer such as an engineer) and the system.

multiple processor approach

Since we are discussing systems concepts *per se*, it would be useful to consider one of the more recent approaches that provides reasonable response to an individual display console user while avoiding the sort of machine loading that degrades the response to other users. As stated previously, an alternative to time-sharing might be the use of a smaller, less expensive, dedicated, general-purpose computer to drive the CRT display. A compromise solution has been proposed[59,67,92] in which such a small computer controls the display console for drawing and other fast response functions. This computer is connected via communication links (or possibly connected directly through an I/O channel) to a larger host system which can access large data structures and community files and execute powerful analysis programs. The system problems involved now become more complex in that the problems of multiple-processor computing must be solved.

A special "graphics problem" occurs with use of this configuration. A massive transfer between processors may be required for radical changes of the picture on the display although the response to requests must remain rapid. To satisfy these requirements, transmission rates must be high or some exceptionally clever program partitioning between processors must be done to minimize the amount of data transferred. Of course, problems also arise with processor-to-processor protocol[10] and machine-independent data formats.

We shall return to the multiple-processor problem later in the paper. It might be stated at this point that the author has programmed an IBM 2250 Model 1 display for interactive graphics in a time-sliced, time-sharing system. The experiment indicated that a small, general-purpose processor or "intelligent-terminal" approach may be the only practicable way of supporting a *large number* of display consoles if complex computer-aided design applications are to be performed in such a system. Otherwise, with only a small number of graphic displays, system overhead for such functions as three-dimensional pen tracking, dynamic rotation, and translation of three-dimensional images becomes prohibitive. It should also be stated that there are some who disagree with this conclusion.

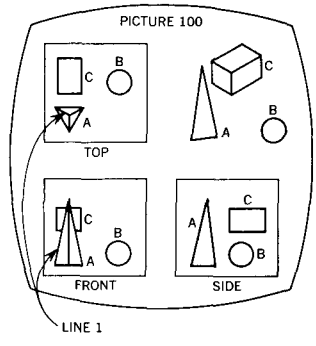
## Data structures

In many of the simpler applications of graphic display terminals with minimal man-machine interaction, the few important entities in the display which the user may detect may be kept in a simple form, such as a two-dimensional array. The purpose of such an array is to correlate the name of the entity with its representation

on the CRT. As the picture is created from a string of display orders in the computer memory or on an auxiliary buffer, the normal means of identification of an image is the location of the order being executed at the instant when the light pen detects the CRT beam. Thus the array correlates some arbitrary name given to the image by the user with the location of the display orders that create the image. The correlation list or table and the list of display orders constitute the display data structure. This simple structure provides a means by which the user may refer to images by name and by which the name of an item detected with the light pen may be retrieved.

In more complex applications of the SKETCHPAD variety, where the user is allowed to define subpictures, modify (delete or add to) the picture, and manipulate (rotate or translate, etc.) its elements or subpictures, there is a need for a more powerful data structure.[1,19,33,86] To use such a structure, the programmer must have access to a mechanism for the previously described correlation function. In addition to the identification of entities (low-level elements of pictures) as provided by the correlation mechanism, the structure must provide for the specification of hierarchical classes. An example of such qualification is "line 1 in image A in the front view of picture 100" as shown in Figure 1, where each qualification except "line 1" indicates class membership. This is similar in concept to a PL/I structure declaration:[37]

Figure 1

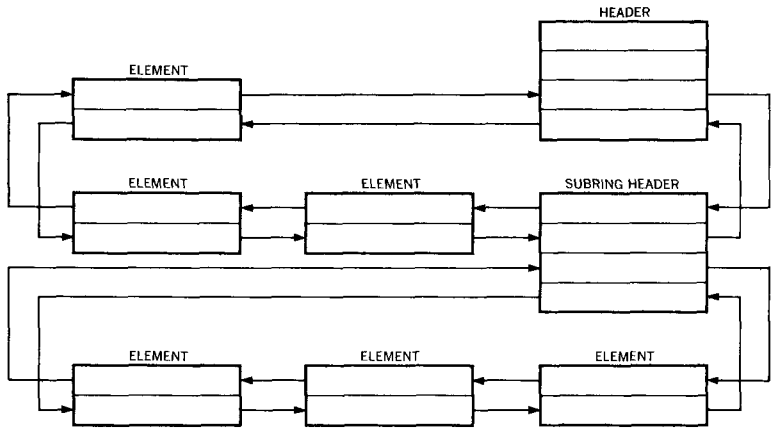



```
1 PICTURE_100,
   2 FRONT_VIEW,
      3 IMAGE_A,
         4 LINE_1;
```

except that it is dynamically declared (in the graphics system) during execution, a facility not currently defined in the PL/I language. Thus, a picture on the display screen and the structure describing it can be thought of as a dynamic tree structure. In some display data structures and in problem data structures, the relationships are not expressed as simply, and the data structures must be represented as a directed graph (of which the tree structure is a special case).

In addition to facilities for the correlation and classification of graphic entities, a means for describing the topology of images was provided in the data structure of the SKETCHPAD and SKETCHPAD III systems. This feature has historically been implemented via list structure techniques, the most popular of which use a special case of circular linked lists that are called *rings*—originally popularized by I. E. Sutherland and D. T. Ross.[68,82,83] A ring is entered at any element of the ring, whereas a list must be entered at the list head. The use of lists also allows the collection of data into sets. Membership in a list denotes membership in the respective set. In this fashion, the use of lists allows the programmer to specify very powerful functions (such as the delete or erase functions) which are performed on all the members of a set—e.g., the set of all ele-

Figure 2 List concept




ments in the front-view subpicture or all resistor symbols in the picture. The same concept may be used in the analysis programs once the facilities for the manipulation of lists are provided. The list concept is illustrated in Figure 2.

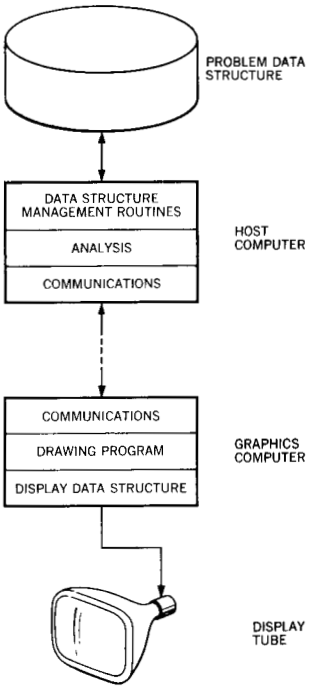
Historically, there has been a tendency to use the same data structure for the previously described display information and for problem data. The ability to collect data into sets which can be dynamically expanded is a very convenient facility in many application programs. As an example, in a circuit analysis program, where the user is allowed to operate an interactive display console to define the circuit, the list facility allows the collection of all resistors and all capacitors into lists as the circuit is defined. In this way, these elements can be easily modified by the user during the process of the analysis program. Unfortunately this approach requires a rather complex list structure system for the requirements of both the display and analysis programs. Generally, it requires the use of variable-length data blocks as elements of the lists. Many analysis programs require a complete representation of sets—intersecting sets, etc.—whereas the display function can be performed using a simple tree representation.

**display data structure**

Recently, it has been stressed that two models really exist in such a system—one for the displayed representation and another for the actual problem or system being represented. Several systems, SYSTEM/360 SKETCHPAD III (currently being implemented by T. E. Johnson at Massachusetts Institute of Technology), GPL 1,[44] LEAP,[70] provide two separate data structures in recognition of this fact. These systems use a simplified tree structure to represent the image on the display, with the problem data stored in a separate data structure. This simplification allows a much more efficient manipulation of the display such as blank delete, add, and the correlate function. The problem data structure can then be specially constructed for the particular application, or a more sophisticated generalized model structure may be used.

In the implementation of the double data structure, the identifiers used for access to elements should be symbolic, independent of the mechanisms of the data structure utilities, and should not be addresses (pointers) used within either structure. If this rule is followed, either structure may be rearranged (as in the garbage collection operations) without necessitating any change in its companion data structure. This condition is especially important if the two structures reside in separate machines, i.e., the display structure in the graphics computer and the problem data structure in the "host" computer, as shown in Figure 3.

Figure 3 Two data structures




Evans and van Dam[29] argue that for efficiency, most applications should use a data structure specifically designed for the application. An example of such a "tailored" data structure is used by Negroponte and Groisser[57] in their implementation of URBAN5 (a conversational graphic design aid for the urban designer). Their data structure records the attributes of urban space modules such as privacy, access, purpose, etc. Thus, the modules may belong to sets of modules, the name of the set indicating that the modules within it have a certain property. These are recorded in an extremely simple data structure consisting of a string of Boolean variables (bits) whose value if true indicates membership in the set represented by this variable. The processing of these Boolean variables is extremely rapid; their system is simple, and the response of URBAN5 is quite pleasing.

Evans and van Dam propose a system—DSPS (Data Structure Programming System), and a language—DSPL (Data Structure Programming Language), for the declaration and manipulation of tailored data structures including arbitrary list and ring structures. A similar facility is available in the AED (ALGOL Extended for Design) language,[68] thus providing the high-level language programmer with the ability to pursue a similar approach. The CORAL language provides a comparable facility, although it is restricted to a specific ring-structure approach to provide a library of standard ring-manipulation operations for the user.

Counter to the plea for data structure declaration and manipulation languages is the argument that the application programmer is primarily concerned with the application and should not have to be cognizant of the implementation of the data structure. (The problems encountered in implementing a powerful data structure are frequently greater than those involved in the application itself.) Thus, if he must be concerned with a special data structure for each problem, the application programmer's job may be twice as hard. This is especially true if the problem is of such a size and encompasses such a scope that the data structure must reside on auxiliary storage for want of enough main storage.

**list structures**

A general data structure can be provided that allows the programmer to specify the data and its relationship to other data in the data base. He may subsequently access data from this data base not only by name but, in addition, through reference to relationships between the desired information and other data in the

file. As previously stated, the traditional data structure for this sort of operation is the ring structure, which is a special case of list structure. Other types of list structure have also been used for this purpose. However, the list-searching involved may be uneconomical when the data structure is on auxiliary storage rather than in main storage. Until recently, little had been done to address this trait in list-processing languages.

In their paper on DSPS, Evans and van Dam have described techniques which they are currently evaluating for automatic swapping of data structures between a disk or drum and main storage. Here the user is allowed to view the combination of main storage and auxiliary storage as one continuous "virtual" main storage, the system bringing the addressed data into main storage as it is addressed. Very large "pages" (the units of data swapped) are brought in when an element within that page is requested. This increases the probability that related data will be available in the main storage without a second access. A "look-ahead" scheme brings in associated pages if there is available space in main storage. An inverted-file approach is used to reduce the disk-seek overhead by providing direct access to a list element on auxiliary storage. The article by Chen and Dougherty[12] in this issue also discusses a data structure whose implementation includes some similar techniques.

**hashed structures**

Another recent development is the appearance of a possible replacement for a ring structure used as a generalized associative data structure. This data structure is the work of Feldman[30] and has been extended by Rovner,[69,70] T. Johnson,[46] and Symonds.[84] Similar data structures have been developed at the University of Michigan.[90,92] It is known variously as a *hashed* data structure, a software-simulated paged associative memory, or a relational store. As with the list and ring structures, the objective is to provide a relational data structure capable of the storage and retrieval of information based on relations between problem model components. An example of the use of this data structure is the retrieval of names and lengths of all the lines connected to a given point in a drawing. In large computer-aided design applications, this information structure consists of so many associations that it is necessary to place it on an auxiliary storage device such as a magnetic disk. The hashed associative store approach is one attempt to speed up retrieval from such a disk-resident store by eliminating list-searching. List-searching on disk devices is extremely time-consuming if the lists are long and their elements are scattered widely over the address space of the disk. This technique is being used in LEAP, SYSTEM/360 SKETCHPAD III, and GPLI as the standard problem data structure.

Hashing is a standard operation, normally used in commercial data processing to transform a reference to a large, sparsely populated address space into a reference to a smaller address space in a manner that minimizes the probability of conflicts (two symbolic addresses which "hash" to refer to the same actual storage cell). This

operation is common to such applications as inventory control where the part numbers which may be long must be mapped into disk addresses referring to the record for that part (e.g., the part number may be ten digits, whereas five digits would suffice to address all the space needed to contain the file).

In the implementation of the relational data structure on disk, one may think of a relation as a triple A, O, V, or *attribute* of *object* equals *value,* such as the SON of TOM is FRED. The file space may be thought of as a two-dimensional array of cells containing names (values) such as FRED, the rows and columns representing attributes (SON) and objects (TOM) such that the FORTRAN statement

ARRAY (SON, TOM) = FRED

might be one way of establishing the above relationship. The hashing operation, as depicted in Figure 4, maps this large, sparsely populated name space into a reference to a smaller, more highly populated one-dimensional array called DISK such that the hashed assignment might be referred to by the FORTRAN statement

DISK (H(SON, TOM)) = FRED

where H represents a single-valued hashing function. In the case of the multiple-valued relation— SON of TOM = FRED, SAM, DON, the cell DISK (H(SON, TOM)) would contain not the value but a pointer to a chained list of elements—FRED, SAM, DON.

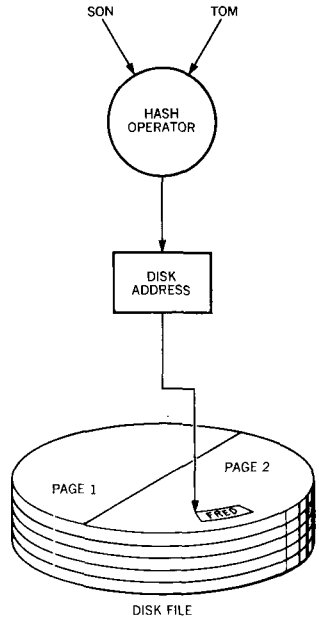
A similar solution exists for other relations *a* of *b* = *c* that hash to the same cell and where sufficient information is provided to differentiate between the two relations. Data may be associated with the values of the stated relations and may be retrieved from the store via reference to the desired relation. Redundant storage techniques are used to reduce accesses to disk for permutations of the accessing triple (*a* of ? = *c*, *a* of *b* = ?, etc.).

The proponents of hashing techniques consider the major attributes to be:

- Direct access to associations versus indirect references in list or ring searches.
- Explicit representation of associations or relations rather than implicit representation via list structures provides for more rapid response to queries.
- Open-ended set of relations is provided as opposed to list structures where possible relationships must be known when the structure elements are defined.
- Explicit relation representation allows relations to be referred to by name enabling them in turn to be referred to as objects or values of other relations.

The structure herein described is very general; the user pays a price: disk storage space requirements may be very large. It is argued that if list structures were to be implemented to solve all the same problems rapidly, the overhead entailed in terms of storage space would at least equal that of the hashing approach.

Figure 4



Thus there are three views of associative data structures:

- Languages should be provided to allow the user to customize the structure for his application.
- A general structure can be provided using lists or rings on auxiliary (disk) storage.
- A hash-coded associative store is the best approach for the general data structure.

The resolution of the three-way argument is the object of current research. The papers referred to constitute reports on the current state of that research. All of the approaches listed perform swapping of data in and out of main storage via a programmed "paging" scheme, with the exception of General Motors APL system which takes advantage of the equipment of the SYSTEM/360 Model 67 for dynamic relocation. Much more research is needed in the use of this or similar equipment-assisted paging schemes.

None of the data structure systems described can be considered complete unless facilities are available for reclaiming released storage (caused by delete operations). This process is called *garbage collection*. The facility may be provided as an automatic function to be initiated by the data structure routines when space is needed, or it can be provided as a utility operation which the user may call when he wishes. However, it is necessary.

## Graphics facilities in high-level languages

High-level languages can be segregated into two categories: problem-oriented, in which the problem is described, and procedural, in which the procedure or algorithm for solution of the problem is specified. In these two categories, graphics languages are separated again into two forms: the written form and the pointing form. All programmers are familiar with the first form where the input to the compiler are statements consisting of strings of alphanumeric characters. The second form provides a facility by which the display programmer can, in the case of procedural language, specify program operation by written statements (as in the written form) combined with use of the function keys and light pen as additional inputs.

Problem-oriented graphics languages are those in which the user describes the picture he wishes to see rather than prescribing the procedure by which the image is to be constructed. The extension to PL/I proposed by Comba[16] is thus, in part, a problem-oriented language although some procedural statements are available. The PLAN Graphic Support (PGS) system[12] is also a problem-oriented language.

Because a procedural language in written form is a necessity for the construction of the other three types of languages specified, we concern ourselves for the rest of this section with the requirements of such a language.

A graphics procedural high-level language must provide the

facilities for the definition and management of the following:

- Asynchronous events (attentions)[1,36]
- Display device input/output and display data structures[39,40]
- Images, views, and geometrical entities[40]
- Problem (associative or relational) data structures
- Processor-to-processor communication

attention-handling facilities

There are several currently used techniques for the specification of actions to be taken when an event, such as a function key depression or light-pen detect, occurs at the display console. In general, all of them are techniques for specifying a procedure to be executed upon the occurrence of such an event. In each case, the system generally collects certain information concerning the status of the display and program when the attention signal occurred. As a minimum, this information comprises the type of attention, the name of the image detected, and the button depressed, as well as the register and machine state. Some systems may record more application-dependent data at this point. This information is made available, by the system, to the user's attention-handling routine. The major differences in the attention-handling schemes known to the author are the methods of specifying the procedures to be executed upon occurrence of the attention and the means by which these procedures are actually invoked after the attention event occurs.

Concerning the point of invocation we only state here that attentions are handled in one of two ways:

- Asynchronously, where the attention procedure is dynamically invoked immediately upon receipt of the attention (or as soon as the program attains an interruptible state)
- Synchronously, where the programmer must poll (explicitly test for an attention event)

In discussing specification of attention routines, we consider four possible language forms:

- Program-flow modification specified by a FORTRAN- or PL/I-type "IF" statement[37]
- Tabular specification as in GPAK[39] and GSP[40]
- Dynamic declaration of attention-handling procedures as in the PL/I "ON" statement[37]
- State diagram approaches[58]

IF statement approach

With the "IF" statement, the following approach might be used. Upon receipt of an attention, the system would record the attention information from the display such as the name of the item detected or the key which was depressed. One of several system Boolean variables would be set to .TRUE. (i.e., LIGHTPEN or KEY). A subsequent statement such as

```
IF LIGHTPEN THEN GO TO LABEL_1;
or
IF KEY THEN CALL ROUTINE;
```

would cause a change in the program flow to a statement or procedure that would perform some action to reflect the fact that the attention had occurred. This is an example of synchronous attention-handling.

tabular specification

The tabular approach is similar, except that it provides a method for prespecifying the desired change in program flow which is to occur when each type of attention occurs. The method of prespecification is normally provided in the form of a subroutine call, although the language could be extended to include a statement for the same purpose. The subroutine form uses a parameter list of ordered pairs to correlate each attention type with a specific action routine.

```
CALL SETUP ((attntype(1), routine(1), . . .
                . . . , attntype (n), routine (n));
```

This form only sets up the dispatching mechanism, and in the typical FORTRAN system (which is not designed for asynchronous operation), the attention is stacked. Later, in the progress of the program, another routine that will test (poll) for the occurrence of an attention may be called. If an attention has occurred, the call to the appropriate subroutine is dynamically generated and executed. If not, the system waits until an attention does occur. A supplementary routine is needed which performs the call if an attention has occurred but does not wait for one if it hasn't occurred. This again is an example of synchronous attention-handling, although in a language allowing asynchronous invocation of routines, this technique could be used for asynchronous attention-handling, and transfer to the appropriate routine could occur immediately upon the receipt of the attention. With this approach, the table may be redefined to change the dispatching for any or all attentions. Care must be taken, however, to provide proper conventions for saving and restoring the dispatching table when calling subroutines that might in turn modify the table for their own purpose.

PL/I ON statement approach

The approach taken in the PL/I language for specifying procedures to be executed upon the occurrence of such an event as overflow, zero divide, etc., could be extended to specify graphic attention-handling. Thus a sequence of statements such as:

```
ON LIGHTPEN BEGIN;
   statement 1
   .
   .
   .
   statement n
      END;
```

would define such a block. Execution of the ON statement in the example is equivalent to changing a table entry containing the transfer point for light-pen action. The technique differs from the previous one in that the ON statements executed in a called pro-

cedure are in effect only for that procedure, and, upon return to the dynamically encompassing procedure, the previous "table" is reinstated. It should be pointed out here that in addition to specifying the desired dispatching or program routing when an attention occurs, it should be possible to disable or enable specific attentions for certain statement sequences. In an extension to PL/I, this could be done by a statement or procedure "prefix." In the previously mentioned tabular system, a subroutine call could set a disabled/enabled indication in the table. The author has implemented both synchronous and asynchronous versions of this approach.[44]

state diagram approach

The previous two methods of attention specification require the user to specify all attention states. (i.e., if only one attention type is to be allowed, its action must be stated and all other attention types must be disabled. This must be done each time the program state definition changes.) W. R. Sutherland and W. M. Newman[58] have independently proposed a more interesting state-transition diagram notation adapted from automata theory in which all that need be stated are the attention types that will be accepted and the state transitions (program jumps or transfers) that will be made when one of these attentions occurs. This technique provides a much more efficient language form and results in equally more efficient generated code. The format of such a statement might be:

```
statelabel: STATE ((attntype(1), statelbl(1)), . . .
            . . . , (attntype(n), statelbl(n), (WAIT | LOOP));
```

With the LOOP option, the subsequent procedure statements would be repeatedly executed. With the WAIT option, the program becomes dormant. In either case, when one of the specified attention types occurs, a change of state occurs, and the program transfers to the state (routine procedure or block) corresponding to that attention. This statement may have either a synchronous or an asynchronous implementation.

display data structure management

Languages exist for the declaration and manipulation of list and ring structures. It is the author's contention that for graphics, however, the programmer should only be concerned with the description of the graphic entities to be added to the current display structure to create the desired image. A knowledge of the data structure implementation should not be necessary. Thus, the programmer should be required to specify only the type of graphic elements to be added to the picture, data for its construction, the name by which it is to be known, and its hierarchical qualification. In the PL/I syntax, examples of two possible ways of stating this follow. In the first example, it might be

```
CALL LINE (startpt, endpt, name, qualif-list);
```

where "startpt," "endpt," and "name," are implementation-defined array or structure variables containing graphic or naming information, and "qualif-list" is an array containing names of hierarchical classes such as CLASS1 and CLASS2. In a second example,

if graphic variables can be declared, then it might be

```
CLASS1.CLASS2.NAME = LINE(STARTPT,ENDPT);
```

where LINE is a function, and where STARTPT, ENDPT, and CLASS1.CLASS2.NAME are graphic variables.

Either of the previous two examples should cause the orders for a line from STARTPT to ENDPT to be placed into the display file and an entry made in the display data structure to correlate this line with the name CLASS1.CLASS2.NAME. If CLASS1.CLASS2.NAME already exists, then the statement should reassign these values and recorrelate the line.

Other operations for the modification of the display and its data structure are required, but they too should not require knowledge of the implementation. Such operations are the deletion or erasure of items or classes, dynamic class structure declaration, and structure reconfiguration. This would be similar to the addition to PL/I of the ability to dynamically declare and redeclare structures. Further details can be found in a report by the author.[44]

Two approaches to causing the actual display of images are possible. Either the display of data on the CRT can occur as a side-effect of the routine that manipulates the display data structure or graphic variables, or the assignment to an internal variable may be required along with a WRITE or DRAW statement for the actual output operation. It is the author's opinion that the display device should be looked upon as a window through which the application user sees the change in his problem dynamically. The display, then, should not be thought of as just an input/output device; there should be available a mode of operation in which the output is implicit with the manipulation of the data structure that represents the display. The only overt act that the programmer should have to perform is the manipulation of the display data structure. In some applications where this approach would present a continuously changing and possibly unaesthetic display, it should be possible for the programmer to state that automatic display should not occur. Another statement later in the program could reactivate automatic display, allowing the previously "delayed" items to be displayed in one complete picture change. But, this should be the override method provided and not the default. The default (dynamic window) mode is similar to the "movie" mode described by Smith.[75]

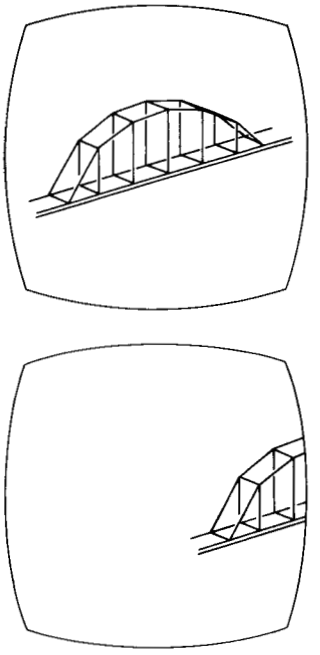
image generation, geometric utilities

A graphic programming language must provide a facility for describing pictures.[6,16,39,40,53,74,88] Pictures used in the wide range of graphic applications are composed of many different types of images which include alphanumeric characters, schematic symbols, geometrical entities, and images not thought of as geometrical (such as plots of arithmetic functions, bar charts, and flowcharts). Thus, it is important that the language provide not only a facility for describing the "nongeometrical" entities in a simple form, but also, a system for describing two-dimensional and three-dimensional geometric entities as well as the routines for constructing these

entities. Routines should also be available to perform geometric transformations upon these entities, i.e., it should be possible to rotate, translate, and scale the image.[1,39,65,66] This paper will not attempt to describe the techniques for these geometrical duties; however, the following statements are within the scope of the discussion.

An extremely systematic approach to manipulation of geometric constructs can be developed by using the homogeneous coordinates[3,21,65,66] of projective geometry. It amounts to the description of n-dimensional shapes in terms of n + 1 dimensions. This technique provides for a simple set of matrix operations for rotation, scaling, translation, perspective projections, and the "clipping" operation.

Figure 5 Scissoring operation

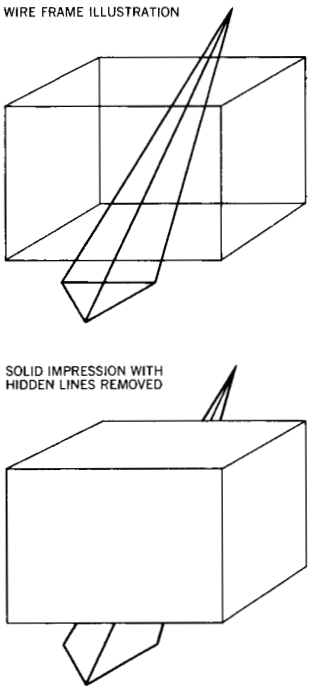


For all except the "clipping" operation, simple matrix multiplications can be used to perform the desired transformation. The "clipping," or "scissoring," operation, shown in Figure 5, is the name given to the algorithm chosen to select the elements that appear within a specified "window" that defines a portion of a larger space that is to be mapped onto the face of the CRT. In this operation, a matrix defining the rectangular window is used. This matrix and the matrix describing each line in the total image are multiplied, and the sign of the result indicates if all or any of the line lies within the desired window. A subsequent computation can determine the point at which any line crosses the edge of the window. Although normally performed by programming, the transformation operations have been performed by special-purpose display devices[79] based on the homogeneous coordinate system. It is the simplicity of the matrix approach, used with homogeneous coordinates, that makes it possible to build equipment for these operations. It might be pointed out that many factors must be considered in deciding whether an equipment approach is sufficiently general for the graphics application.

Since most of the displays currently available do not contain general-purpose curve generators, it is necessary to provide routines for approximating complex curved shapes, using connected short line segments. A technique that provides for reasonably rapid computation of these line segments uses parametric cubics or rational polynomials.[3,19,20] With the use of rational polynomials, a spline-like curve[2] can be computed to provide a smooth approximation to the desired curve. For further information on homogeneous coordinates and parametric surfaces, see the paper by Ahuja and Coons[3] in this issue. Coons and Herzog[19,20] also present a technique for specifying three-dimensional surfaces which can be patched together for approximating such complex shapes as airframes and auto bodies.

The previous discussion has assumed that the representation presented on the display is of the form of a wire frame where all edges in the object are visible at all times. Techniques have been developed for the elimination of the normally hidden lines giving an impression of a three-dimensional solid rather than a wire frame

Figure 6 Removing hidden lines




outline. Figure 6 illustrates the result of employing such a technique. Current research consists mainly of improving the speed and generality of these techniques. This is beyond the scope of this discussion, and the reader is referred to the literature.[4,5,6,17,66,88]

Currently, at the University of Utah, an approach to half-tone rendering is being explored.[89] Computer-generated holograms are also being considered to provide a more accurate representation of three-dimensional images.[54]

As a minimum, a graphics language must provide either strong matrix facilities or built-in routines for performing the previously described geometric calculations, but preferably both. It is important that the matrix arithmetic be extremely efficient if dynamic rotations and translations are to be performed smoothly.

Since not all applications need these sophisticated two- and three-dimensional facilities, a graphics language must provide for easy specification of simple pictures such as schematics and plots. A user should not have to perform his own data scaling and axis layout but should be able to pass his data to a plotting routine and have these operations performed automatically. It should be possible to describe a flowchart or schematic by providing the names of the block types, location, and the connection information. Once the block types and their representations have been described (or retrieved from a library) in the form of named "macros," the representation can be duplicated each time that block type (name) is used again. Depending on the exact nature of the "macros," the subsequent use of the "macro" name creates an instance, or copy.[82]

problem data structure

Implementations of problem data structures were discussed in a previous section. Here we are concerned only with the language for specifying and manipulating them. If it is desired to provide the ability to create special data structures, then special statements for the declaration and manipulation of lists, rings, hashed tables, and "plexes" must be provided as in APL (Associative Programming Language[26]), DSPL,[29] PL/I, AED,[68] and CORAL.[83] If the approach is to provide built-in, generalized data structures, then the programmer must be able to create and access data on the basis of the interrelations between the data elements. Despite the implementation techniques used, it should be possible to access data and then use it via a statement sequence such as:

```
FOREACH LINE SUCH THAT POINTA IS END OF LINE OR POINTB
    IS START OF LINE DO;
                statement 1
                    .
                    .
                statement n
END;
```

The FOREACH and END statements form the beginning and end of a loop such that statements 1 through n are executed once for each value of the variable LINE which starts at point POINTB or ends at point POINTA. LINE, POINTA, and POINTB can be referred to as variables within the loop.

As an example, if COORD is a vector specifying coordinates of a point and COORD is mapped into a structure by a PL/I declaration statement, then the statements

```
LOOP: FOREACH LINE;
      FOREACH PT SUCH THAT PT IS STARTPOINT LINE:
      CALL DRAW (PT → COORD);
      END LOOP;
```

might plot all line start points (where each value of PT is returned by the access routines in the form of a pointer to the structure containing COORD). Similar language features must be provided for the entry of data into the data structure based on relational information. Statements are also required for deleting data and relations from the store or for modifying data associated with a relation.

A comparison of the literature describing the syntaxes of LEAP and APL should indicate that this language form can be used for accessing data from data structures implemented with either ring-manipulation or hashing techniques (see Symonds[84]). Based on similar implementation techniques, Childs[13] has implemented a set-theoretic language. (LEAP includes sets and set operations in addition to relations and relational operations.)

processor-to-processor communication

The use of a small computer to drive the graphic display console provides a solution to two problems:

- Reduction of the conversational overhead on the host (multi-access) system.
- Localizing the interactive functions at the console providing more rapid response when the console is remotely located from the host system.

The approach taken in providing programs for this multi-processor configuration is dependent on the speed of the communications link between the two processors. At one end of the spectrum of possible programming-support approaches is the use of the small computer as a super controller which can dynamically allocate its own main storage and is able to perform some additional functions such as image generation and attention stacking. This approach, however, does not take full advantage of the power of the small computer. It also puts a heavy load on the communication channel as the number of executions for these low-level facilities is rather high.

At the opposite end of the spectrum is the use of the small computer to execute all interactive graphic operations plus much of the analysis program, using the host computer as merely a facility for access to the problem data base. This approach would work quite well for simple applications but would probably require the application programmer to write each application specially to fit all the application and graphics routines into such a small machine.

An approach currently being considered would provide a fairly constant base of conversational facilities upon which new applica-

tions could be constructed. Figure 7 illustrates this approach. The small computer would contain all the geometry, attention-handling, image-generation, and display control routines and a *subset* of the associative data structure facilities. Subsets of the data structure called *data distillates* would be passed between the host and remote computers. The small remote computer would then contain most of the interactive drawing functions and could be used for creating and modifying the data structure. The host computer would perform the analysis, data structure storage, and data structure retrieval programs.

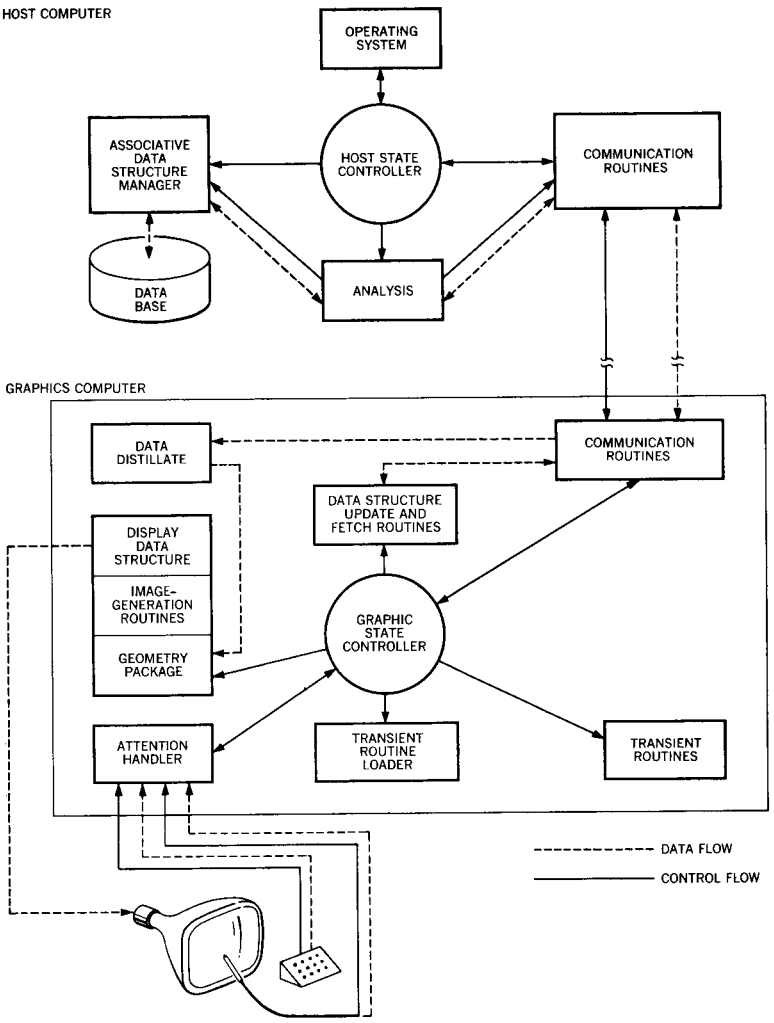
The implications to the language at this time appear unclear and will be the subject of future research. It appears now that the language extensions proposed in an earlier section of this paper reduce to attention-handling, data structure statements, and additional statements for passing data distillates between the two machines, coupled with the corresponding control information to specify the operation to be performed by the receiving processor. The "attentions" mentioned here would not be at the level of those mentioned before, such as light-pen detect or function key depression. In this case, they would signify the asynchronous request of one processor for services to be performed by the other and a corresponding "interrupt" to indicate completion or failure of an assigned task. The functions of, and programming language for, the remote graphics processor configuration are currently the subject of research in the university and laboratory environments.

## Summary

In summary, it should be mentioned once again that most of the problems stated in this paper are not exclusively graphics problems but are general problems of interactive systems and data base systems. The speed of the graphic display console merely emphasizes these problems.

Although our thesis has been the nongraphical nature of so-called graphics problems, we have discussed some problems which are to be considered graphical in nature, and we do not mean to under-emphasize their importance. Certainly attention-handling and the topological data structures used for picture modeling are also used in areas other than graphics data processing. They are, however, fundamental to the interactive graphics application. Geometrical routines for image construction, image manipulation, and image clipping are inherently graphic. No *large* graphics system can exist without a comprehensive problem data structure and its data manipulation routines. *Some* of the more appealing (because they are general) approaches to these problems were discussed. A final thought we might leave is that no existing graphics system commercially available in data processing has, to date, included a complete set of the indicated generalized routines. As a result, each application programmer must program much that is not directly a function of his application when a graphics approach

Figure 7 Two-computer approach




is used. This method is costly and may be the most valid reason for the slow evolution of graphics techniques in production use.

Much research remains to be done in the sharing of resources among many consoles, in design of data bases for associative retrieval, in processor-to-processor communication, and the specification of high-level languages for programming applications which use these facilities. Some of the other papers in this issue describe various approaches to some of the stated problems, others describe interactive applications using graphic display consoles. The bibliography included in this paper lists some of the literature on time-sharing and graphics data processing.

## Appendix

Requirements for a conversational system in which a graphic display terminal is used for communication between the man and the system—the man being the application programmer or the engineer (user):

- Communication with the system should be simple—employing an easy-to-use command language[12,41,45] simple enough for use by application users who have no knowledge of computer programming.
- The command language should be "forgiving," allowing the terminal user to restate any command in error.
- Errors should only activate the diagnostic routines and never force termination of the job at hand; certainly, never affecting another user running concurrently on the system.
- Facilities should be provided for diagnosing program errors from the terminal—the application programmer should be permitted to correct the perceived errors, on line, or change the source code, recompile it, and run it again.
- The command language should provide the basic functions for the creation and maintenance of source program files, program libraries, and data files.
- It should be possible for the application programmer, using the display console, to assemble or compile object code from the source files and then link together the resultant object files into a program and execute it.
- Although it should be possible for the programmer or application user to protect his files from others, it must also be possible for him to designate files as "shared" to allow several users to work on a joint project using a common file as a data base.
- Backup and recovery facilities (especially for files) are extremely important.
- It must be possible for the application programmer, or the application user, to extend the system by defining new commands from the display console.[12]
- A special requirement of display-oriented systems is the ability to produce a printed history of the user commands and the resulting computer actions, as the display device produces no "hard copy" audit trail (trace) by itself.
- Because a conversational system should allow the user to define data and call programs as they are needed, the system must contain a facility for the dynamic allocation of main storage. This facility should be easy for the programmer to use and should be "transparent" to the application user—at least until all the resources have been exhausted. The programs must be linked dynamically as they are called into the system.

These requirements may not seem extraordinary if considered in relation to the teletype or typewriter terminals normally used in conversational systems. However, it should be stressed again: the

display console has greater abilities, and its inherent speed causes the user to be more demanding. Thus the implementation of all the facilities just discussed must be extremely efficient to provide an acceptable response.

BIBLIOGRAPHY


1. Adams Associates, Inc., "Computer display fundamentals—software techniques," *Computer Display Review* **1,** 11.66.0–11.118.0.
2. D. V. Ahuja, "An algorithm for generating spline-like curves," in this issue.
3. D. V. Ahuja and S. A. Coons, "Geometry for construction and display," in this issue.
4. A. Appel, "Some techniques for shading machine renderings of solids," *AFIPS Conference Proceedings, Spring Joint Computer Conference* **32,** 37–45 (1968).
5. A. Appel, "The notion of quantitative invisibility and the machine rendering of solids," *Proceedings of the 22nd National Conference of the Association for Computing Machinery* **P-67,** 387–394 (1967).
6. A. Appel, "Modeling in three dimensions," in this issue.
7. H. B. Baskin and S. P. Morse, "A multi-level modeling structure for interactive graphic design," in this issue.
8. C. G. Beatty, "Graphic approach to numerical information processing," *Proceedings of the Society for Information Display Symposium* (1967).
9. E. Bennett, E. Haines, and J. E. Summers, "AESOP: A prototype for on-line user control of organizational data storage, retrieval and processing," *AFIPS Conference Proceedings, Fall Joint Computer Conference* **27,** Part I, 435–456 (1965).
10. A. L. Bhushan and R. J. Stotz, "Procedures and standards for intercomputer communications," *AFIPS Conference Proceedings, Spring Joint Computer Conference* **32,** 95–104 (1968).
11. S. H. Chasen, "The introduction of man-computer graphics into the aerospace industry," *AFIPS Conference Proceedings, Fall Joint Computer Computer Conference* **27,** 883–892 (1965).
12. F. C. Chen and R. L. Dougherty, "A system for implementing interactive applications," in this issue.
13. D. L. Childs, *Description of a Set-Theoretic Data Structure,* Technical Report 3, CONCOMP Project, University of Michigan (1968).
14. B. B. Clayton, E. K. Dorff, and R. E. Fagen, "An operating system and programming systems for the 6600," *AFIPS Conference Proceedings, Fall Joint Computer Conference* **26,** Part II, 41–57 (1964).
15. M. P. Cole, P. H. Dorn, and C. R. Lewis, "Operational software in a disc-oriented system," *AFIPS Conference Proceedings, Fall Joint Computer Conference* **26,** Part I, 351–362 (1964).
16. P. G. Comba, "A language for three-dimensional geometry," in this issue.
17. P. G. Comba, *A Procedure for Detecting Intersections of Three-Dimensional Objects,* IBM New York Scientific Center Report 320–2924, International Business Machines Corporation, Branch Office (1968).
18. W. T. Comfort, "A computing system design for user service," *AFIPS Conference Proceedings, Fall Joint Computer Conference* **27,** Part I, 619–626 (1965).
19. S. A. Coons, "An outline of the requirements for a computer-aided design system," *AFIPS Conference Proceedings, Spring Joint Computer Conference* **23,** 299–304 (1963).
20. S. A. Coons and B. Herzog, *Surfaces for Computer-Aided Aircraft Design,* American Institute of Aeronautics and Astronautics, New York, New York (October 25, 1967).
21. S. A. Coons, *Surfaces for Computer-Aided Design of Space Forms* MAC-TR-41, Clearinghouse for Federal Scientific and Technical Information, Springfield, Virginia (1967).

22. F. J. Corbato et al., "Session 6: A new remote-accessed man-machine system," *AFIPS Conference Proceedings, Fall Joint Computer Conference* **27,** Part I, 185–247 (1965).
23. G. J. Culler and B. D. Fried, "The TRW two-station on-line scientific computer," *Computer Augmentation of Human Reasoning,* 65–88 (1965).
24. G. J. Culler and R. W. Hurr, "Solution of non-linear integral equations using on-line computer control," *AFIPS Conference Proceedings, Spring Joint Computer Conference* **21,** 129–138 (1962).
25. M. R. Davis and T. O. Ellis, "The RAND Tablet: A man-machine communication device," *AFIPS Conference Proceedings, Fall Joint Computer Conference* **26,** Part I, 325–331 (1964).
26. G. G. Dodd, "APL—A language for associative data handling in PL/I," *AFIPS Conference Proceedings, Fall Joint Computer Conference* **29,** 677–684 (1966).
27. C. Engleman, "MATHLAB: A program for on-line machine assistance in symbolic computations," *AFIPS Conference Proceedings, Fall Joint Computer Conference* **27,** Part I, 413–421 (1965).
28. A. L. Eshlemap and H. B. Meriwether, "Graphic applications to aerospace structural design problems," *Proceedings of the* SHARE *Design Automation Workshop* (1967).
29. D. Evans and A. van Dam, *Data Structure Programming System,* accepted for IFIP Congress in Edinburgh (1968).
30. J. Feldman, *Aspects of Associative Processing,* MIT Lincoln Laboratory Technical Note 1965-13, Cambridge, Massachusetts (April 1965).
31. J. W. Forgie, "A time- and memory-sharing executive program for quick-response on-line applications," *AFIPS Conference Proceedings, Fall Joint Computer Conference* **27,** Part I, 599–609 (1965).
32. M. J. Goldberg, R. H. Shroeder et al., *Development of On-Line System for Computer-Aided Design of Integrated Circuits,* Technical Report AFML-TR-342, Volumes I and II, Norden Division, United Aircraft.
33. 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).
34. B. Hargreaves et al., "Image processing hardware for a man-machine graphical communication system," *AFIPS Conference Proceedings, Fall Joint Computer Conference* **26,** Part I, 363–386 (1964).
35. OS/360 *Concepts and Facilities,* SYSTEM/360 Reference Library, C28-6535, International Business Machines, Data Processing Division, White Plains, New York.
36. OS/360 *Graphic Programming Services—Basic,* SYSTEM/360 Reference Library, C27-6912, International Business Machines Corporation, Data Processing Division, White Plains, New York.
37. PL/I *Language Specifications,* SYSTEM/360 Reference Library, C28-6571, International Business Machines Corporation, Data Processing Division, White Plains, New York.
38. TSS/360 *Concepts and Facilities,* SYSTEM/360 Reference Library, C28-2003, International Business Machines Corporation, Data Processing Division, White Plains, New York.
39. GPAK—*An On-Line* SYSTEM/360 *Graphic Data Processing Subroutine Package with Real Time 2250 Input and Display,* Version II, SYSTEM/360 General Program Library, International Business Machines Corporation, Data Processing Division, White Plains, New York.
40. IBM SYSTEM/360 *Operating System Graphic Programming Services for* FORTRAN IV, C27-6932-1, International Business Machines Corporation, Data Processing Division, White Plains, New York.
41. *CP/CMS User's Guide,* IBM Cambridge Scientific Center Report 320-2015, International Business Machines Corporation, Branch Office.
42. E. L. Jacks, "A laboratory for the study of graphical man-machine communication," *AFIPS Conference Proceedings, Fall Joint Computer Conference* **26,** Part I, 343–350 (1964).

43. M. J. Jacobs, "Geometric constraint satisfaction for a computer-display system," *Proceedings of the* SHARE *Design Automation Workshop* (1965).
44. C. I. Johnson, *An Experimental* PL/I *Extension for Graphic Programming,* IBM Cambridge Scientific Center Report 320–2025, International Business Machines Corporation, Branch Office (1968).
45. C. I. Johnson and R. M. Mitchell, *A Conversational Partition Monitor for* OS/360 MFT, IBM Cambridge Scientific Center Report 320–2020, International Business Machines Corporation, Branch Office (March 1968).
46. T. E. Johnson, *A Mass Storage Relational Data Structure for Computer Graphics and Other Arbitrary Data Stores,* MIT Preprint 866, Cambridge, Massachusetts.
47. T. E. Johnson, "SKETCHPAD III: A computer program for drawing in three-dimensions," *AFIPS Conference Proceedings, Spring Joint Computer Conference* **23,** 347–353 (1963).
48. R. Kaplow, J. Brackett, and S. Strong, "Man-machine communication in on-line mathematical analysis," *AFIPS Conference Proceedings, Fall Joint Computer Conference* **29,** 465–477 (1966).
49. J. R. Kennedy, "A system for time-sharing graphic consoles," *AFIPS Conference Proceedings, Fall Joint Computer Conference* **29,** 211–222 (1966).
50. H. A. Kinslow, "The time-sharing monitor system," *AFIPS Conference Proceedings, Fall Joint Computer Conference* **26,** Part I, 443–454 (1964).
51. J. S. Koford et al., "Using a graphic data-processing system to design artwork for manufacturing hybrid integrated circuits," *AFIPS Conference Proceedings, Fall Joint Computer Conference* **29,** 229–246 (1966).
52. F. N. Krull and J. E. Foote, "A line scanning system controlled from an on-line console," *AFIPS Conference Proceedings, Fall Joint Computer Conference* **26,** Part I, 397–410 (1964).
53. H. E. Kulsrud, "A general purpose graphic language," *Communications of the ACM* **11,** No. 4, 247–254 (1968).
54. L. B. Lesem et al., *Computer Synthesis of Holograms for 3-D Display,* IBM Houston Scientific Center Report 320-2327 (January 1968).
55. C. Levinthal, "Molecular model-building by computer," *Scientific American* **214,** 42–52 (June 1966).
56. R. W. Lichtenberger and M. W. Pertle, "A facility for experimentation in man-machine interaction," *AFIPS Conference Proceedings, Fall Joint Computer Conference* **27,** 589–598 (1965).
57. N. Negroponte and L. Groisser, "URBAN5: An on-line urban design partner," *Ekistics* **24,** No. 142, 289–291 (September 1967).
58. W. M. Newman, "A system for interactive graphical programming," *AFIPS Conference Proceedings, Spring Joint Computer Conference* **32,** 47–54 (1968).
59. W. H. Ninke, "Graphic I—A remote graphical display console system," *AFIPS Conference Proceedings, Fall Joint Computer Conference* **27,** Part I, 839–846 (1965).
60. Y. Okaya, "Interactive aspects of crystal structure analysis," in this issue.
61. D. B. Parker, "Solving design problems in graphical dialogue," *On-Line Computer Systems,* McGraw-Hill Book Company, New York, New York.
62. R. P. Parmalee, *Three-Dimensional Stress Analysis for Computer-aided Design,* Ph.D. thesis, Department of Mechanical Engineering, MIT, 51–58 (1966).
63. D. E. Rippy, D. E. Humphries, and J. A. Cunningham, "MAGIC: machine for automatic interface to a computer," *AFIPS Conference Proceedings, Fall Joint Computer Conference* **27,** Part I, 819–830 (1965).
64. L. G. Roberts, *Graphical Communication and Control Languages,* MIT Reprints MSS 1173 (1964), Cambridge, Massachusetts.
65. L. G. Roberts, "Homogeneous Matrix Representation and Manipulation of N-Dimensional Constructs," Notes for English Summer Conference Course, University of Michigan (1965).
66. L. G. Roberts, *Machine Perception of Three-Dimensional Solids,* Lincoln Laboratory Technical Report 315, 22, Lexington, Mass. (May 1965).

67. D. T. Ross et al., "The design and programming of a display interface system integrating multi-access and satellite computers," *Proceedings of the* SHARE *Design Automation Workshop* (1967).
68. D. T. Ross and J. E. Rodriguez, "Theoretical foundation for the computer-aided design system," *AFIPS Conference Proceedings, Spring Joint Computer Conference,* **23,** 305–322 (1963).
69. P. D. Rovner and J. A. Feldman, *An Associative Processing System for Conventional Digital Computers,* Technical Note 1967-19, Lincoln Laboratories, MIT, Lexington, Massachusetts (1967).
70. P. D. Rovner and J. A. Feldman, *The Leap Language and Data Structure,* accepted for IFIP Congress in Edinburgh (1968).
71. A. Ruyle, "Development of systems for on-line mathematics at Harvard," *Project* TACT *Memorandum* **71,** Aiken Computation Laboratory, Harvard University, Cambridge, Massachusetts (1967).
72. A. Ruyle, J. W. Brackett, and R. Kaplow, "The status of systems for on-line mathematical assistance," *Proceedings of the 22nd National Conference of the Association for Computing Machinery* **P-67,** 151–167 (1967).
73. P. M. Schwinn, "A problem-oriented graphic language," *Proceedings of the 22nd National Conference of the Association for Computing Machinery* **P-67,** 471–476 (1967).
74. SICGRAPH, "Bibliography of selected articles on computer graphics," *ACM Special Interest Committee on Computer Graphics Newsletter* **1,** No. 1 (1967).
75. R. V. Smith, *The Electronic Coding Pad,* IBM Thomas J. Watson Research Report NC-731, Yorktown, New York (1967).
76. O. D. Smith and H. R. Harris, "Autodraft," *Proceedings of the* SHARE *Design Automation Workshop* (1965).
77. T. H. Spellman, T. R. Allen, and J. E. Foote, "Input/output software capability for a man-machine communication and image-processing system," *AFIPS Conference Proceedings, Fall Joint Computer Conference* **26,** Part I, 387–396 (1964).
78. T. G. Stockham, Jr., "Some method of graphical debugging," *Proceedings of the IBM Scientific Computing Symposium on Man-Machine Communication,* 57–72 (1965).
79. R. H. Stotz, "Man-machine console facilities for computer-aided design," *AFIPS Conference Proceedings, Spring Joint Computer Conference* **23,** 323–328 (1963).
80. A. N. Stowe et al., "The Lincolner Reckoner: An operation-oriented, on-line facility with distributed control," *AFIPS Conference Proceedings, Fall Joint Computer Conference* **29,** 433–444 (1966).
81. I. E. Sutherland, "Computer graphics—ten unsolved problems," *Datamation* **12,** No. 5, 22–27 (May 1966).
82. I. E. Sutherland, "SKETCHPAD—A man-machine graphical communication system," *AFIPS Conference Proceedings, Spring Joint Computer Conference* **23,** 329–346 (1963).
83. W. R. Sutherland, *The Coral Language and Data Structure,* Appendix C, Lincoln Laboratory Technical Report 405 (1965).
84. A. J. Symonds, "Auxiliary-storage associative data structure for PL/I," in this issue.
85. J. R. Teixeira and R. P. Sallen, "The Sylvania Data Tablet: A new approach to graphic data input," *AFIPS Conference Proceedings, Spring Joint Computer Conference* **32,** 315–321 (1968).
86. E. M. Thomas, "GRASP—A graphic service program," *Proceedings of the 22nd National Conference of the Association of Computing Machinery* **P-67,** 395–402 (1967).
87. J. S. Waxman et al., "Automated logic design techniques applicable to integrated circuit technology," *AFIPS Conference Proceedings, Fall Joint Computer Conference* **29,** 247–265 (1966).
88. R. A. Weiss, "BE VISION, a package of IBM 7090 FORTRAN programs to draw orthographic views of plane and quadric surfaces," *Journal of the Association for Computing Machinery* **13,** No. 2, 194–204 (1966).

89. C. Wylie et al., "Half-tone perspective drawings by computer," *AFIPS Conference Proceedings, Fall Joint Computer Conference* **31,** 49–58 (1967).
90. W. Ash and E. H. Sibley, "TRAMP: a relational memory with an associative base," *AFIPS Conference Proceedings, Fall Joint Computer Conference* **33,** (1968).
91. W. K. English, D. C. Engelbart, and M. L. Berman, "Display selection techniques for text manipulation," *IEEE Transactions on Human Factors in Electronics* **HFE-8,** No. 1, 5–15 (March 1967).
92. E. H. Sibley, R. W. Taylor, and D. C. Gordon, "Graphical systems communication: an associative memory approach," *Proceedings of the 23rd National Conference of the Association for Computing Machinery* (1968).