In the past, a common data structure, or model, has been used for all phases of a computer graphics design system. This has meant that the model used during conversational interaction was the same as the model used in the subsequent analysis operations, usually resulting in poor overall performance. This paper suggests the use of separate models for each phase, providing a general model for the conversational drawing phase which is suitable as a front end in many different application areas.

Described are data structures and programs for both phases: a conversational display image manipulation program (DIM) *and its interconnections with an existing analysis application program* (IBM 1130 *Continuous System Modeling Program*). *Examples of the use of this particular multilevel modeling design facility are included.*

INTERACTIVE GRAPHICS IN DATA PROCESSING

A multilevel modeling structure for interactive graphic design

by H. B. Baskin and S. P. Morse

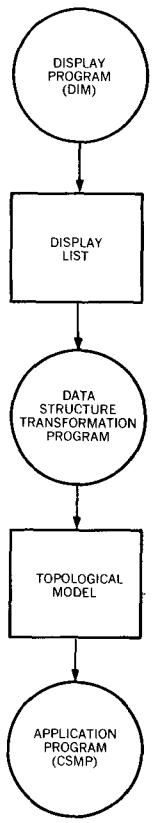
One approach to implementing a graphics application program for computer-aided design consists of both defining a set of functions for the particular application and developing a sufficiently comprehensive data structure within which these functions can be executed. Examples of such facilities are CIRCAL[1] for electrical circuit design, CADIC[2] for integrated circuit design, and CADD[3] for logic circuit synthesis. Typically, the functions in such systems are designed for specific purposes and are not readily usable for other applications. Therefore, developing a facility for a new application often requires nearly as great an effort as the original development.

This paper describes an approach to the problem by partitioning the total set of functions so that an application-independent subset can be defined and need not be repeated for each application. The remaining application-dependent functions can be performed by a separate noninteractive, nongraphics program. The design and implementation of the nongraphics program is significantly simpler than the creation of a complete graphics application package.

Such an experimental partitioning, or multilevel graphics program structure, is the subject of this paper. Discussed first are general characteristics of the application-independent graphic functions and of the application-dependent nongraphic functions. As a result of this partitioning, separate data structures can be

used for each subset of functions. The paper illustrates this principle by means of example data structures and discusses the transforming from one data structure to the other. The interrelations among the programs and data structures described in this paper are shown in Figure 1. The paper concludes with a brief review of implementation and results. The functional division discussed here leads to a bilevel modeling structure. However, the principle of separate data structures is general and fundamental, and it is applicable to more than two modeling levels.

Figure 1 Interrelations among programs and data structures



General characteristics

The complete application package for use with a graphics console contains a set of functions for creating and modifying displayed drawings and a set of functions for analyzing or operating upon completed drawings to produce meaningful and useful results. The drawing manipulation functions are called *conversational functions*, and the functions for operating on the completed drawings are called *analysis functions*. Some examples of conversational functions are light-pen tracking, line drawing, and picture copying. Examples of analysis functions are determining the current in a circuit, calculating the stress in a beam, and minimizing wire length.

The choice of conversational functions is based on the characteristics of the drawing medium (such as paper and pencil, or display console and light pen) and on human factors. For example, a conversational function such as ROTATE cannot be implemented with pencil and paper, whereas it can be performed by a display console. The application and interpretation of the symbols, on the other hand, do not play a role in the choice of conversational functions. Hence, conversational functions are application-independent and merely provide the user with a simple language for describing a pictorial representation of his problem to the computer.

The conversational functions must provide rapid responses if they are to help the user overcome the "boundary problems" between man and machine. Intervals between executions of these functions are of the order of seconds, and many might be called for consecutively. They generally involve simple data processing and, thus, the rapid-response requirement can usually be met satisfactorily, providing the conversational functions are performed in the machine in which the necessary data resides.

In contrast to conversational functions, analysis functions are oriented toward the particular application, and are usually tailored for each application or application class. They are nongraphic in nature and not required to be interactive. Analysis functions are executed much less frequently than conversational functions. Usually a single analysis function is executed after the execution of many conversational functions. Rapid responses to analysis functions are not essential because most users do not expect results of a complex operation without a noticeable delay. Analysis

functions generally perform numerical calculations and can be well-defined and specified in such conventional noninteractive, nongraphic languages as FORTRAN or ALGOL.

An advantage of the partitioning of the set of functions is that it makes possible the use of two separate data structures, each appropriately designed to support a particular subset of system functions. Such partitioning, in conjunction with the use of more than one data structure to represent the same problem, is referred to as *multilevel modeling*. Neither individual data structure need be as complex as a single data structure designed to support both conversational and analysis functions. Also, the design and implementation of both conversational and analysis functions are greatly simplified if separate data structures are used. The conversational functions with their own data structure can then be application-independent.

Because conversational functions involve the creation and manipulation of drawings, a display-list type of data structure seems most suitable for these functions. The type of data structure used for analysis functions depends on the particular analysis being performed. For example, a simplified ring-type structure is often the best type of data structure for the analysis of topological networks.

Another advantage of separate data structures is that conversational and analysis functions can be performed in different machines without requiring a wide bandwidth (high speed) interconnection. Conversational functions are best performed interactively in real time at the console, whereas analysis functions are best performed by batch processing. Both can be accomplished through the use of an interactive intelligent terminal (small processor) attached to a large shared central processor, thereby removing a large portion of the real-time load from the central processor.

Another merit of the partitioning idea is that the specification and implementation of the analysis functions are no more complicated in an interactive design facility than in a noninteractive batch-processing computer, because real-time and graphic aspects need not be considered. Thus, many existing noninteractive application programs can be incorporated in an interactive design environment by coupling them to a facility that provides a set of conversational graphic functions.

The use of multilevel models may create such requirements as the ability to reflect a change in one model as a change in another model, or the ability to generate one model from another. Such requirements might be unidirectional or bidirectional, depending upon the application. To study possible problems, we have implemented a system that couples a conversational program, which we call Display Image Manipulator (DIM),[4] to an application program called the IBM 1130 Continuous System Modeling Program (CSMP).[5] The DIM program was written in assembler language and designed to be application-independent. The CSMP program was written in FORTRAN and was not intended for use in a graphic en-

vironment. The result of DIM plus CSMP is a responsive real-time graphics facility.

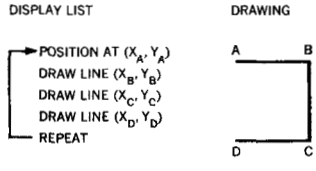
The CSMP program simulates the response of an analog computer on a digital computer. Its data structure is a list of the interconnections between the analog elements. Such a data structure can be used by any other application program concerned with topology. For example, a circuit analysis program can use the same data structure to specify the interconnections between circuit elements. Thus, there is little loss of generality in selecting CSMP as a research vehicle.

Data structures

display lists

A representation of a line drawing suitable for graphic display usually consists of a list of instructions that control the movement of a pen or electron beam. Such a list of instructions is called a *display list* and is illustrated in Figure 2. A *structured display list* is one that contains, in addition to pen- and beam-movement instructions, branches to and returns from display sublists. (Sublists are similar to subroutines in a computer program.) By storing the return address in the sublists, nesting of sublists is possible. An example of a structured display list and its accompanying drawing are shown in Figure 3.

Figure 2 Display list and drawing



A structured display list allows for the extraction of topological relationships from a line drawing without having to resort to pattern recognition techniques. For example, consider a structured display list for an electrical circuit. The shape of each type of component (resistor, capacitor, etc.) is contained in a display sublist. The sublist corresponding to each component type is known in advance. The display list for the circuit contains (1) instructions for drawing the circuit wiring and (2) the branches to sublists for drawing the circuit components. Thus, the presence of a resistor at a particular circuit location is indicated by the presence of a branch instruction properly located in the display list and pointing to the resistor sublist.

conversational functions

The fact that a structured display list can support conversational functions is illustrated by examples that follow. Each conver-

Figure 3 Structured display list

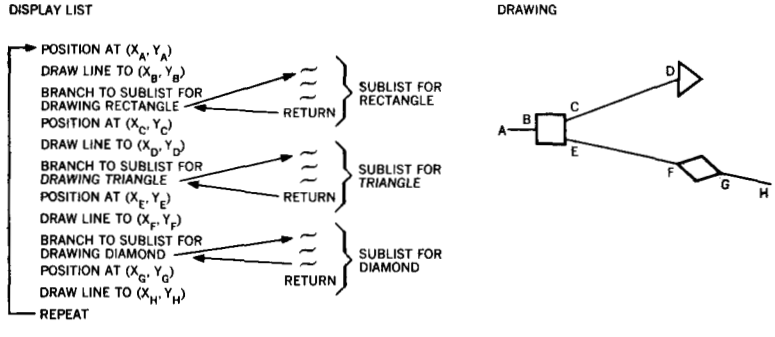


Figure 4 Structured display list for square and triangle

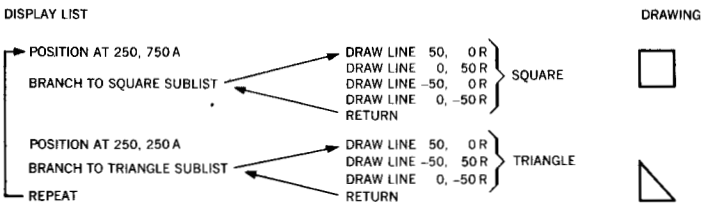
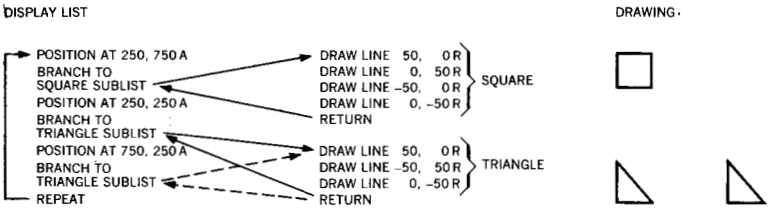


Figure 5 Sharing a subpicture



sational function to be described corresponds to a single button depression on an IBM 2250 function keyboard. Figure 4 shows a structured display list for drawings of a square and a triangle. The A or R following each pair of coordinates indicates whether the coordinates are absolute (X, Y) or relative (ΔX, ΔY). A pair of absolute coordinates corresponds to a point whose origin lies at the lower left corner of the display screen. A pair of relative coordinates corresponds to a point whose origin is the most recently encountered point in the display list.

Another copy of the triangle can be added to the picture by inserting in the display list another positioning instruction and another branch to the triangle sublist as shown in Figure 5. This is accomplished by the SHARE function. The SHARE function has the property that any change to one copy is reflected in the other copy. Another function, called DUPLICATE, does not have this property, but creates a new copy of the sublist for a particular subpicture. Figure 6 shows another copy of the square that has been added by the DUPLICATE function. Subsequent changes made to one copy of the square are not reflected in the other copy.

A subpicture can be moved by changing the coordinates of a positioning vector, as shown in Figure 7, by means of the TRANSLATE function. Because TRANSLATE is not executed on the sublist level, one copy of a shared subpicture can be translated without affecting the other copy. With our system configuration, functions such as ROTATE and SCALE must be performed at the sublist level. Thus, the scaling of one copy of a shared subpicture causes a scaling of the other copy as shown in Figure 8.

A line can be removed from a subpicture by the ERASE or DELETE function. The ERASE function (illustrated in Figure 9)

Figure 6 Duplicating a subpicture

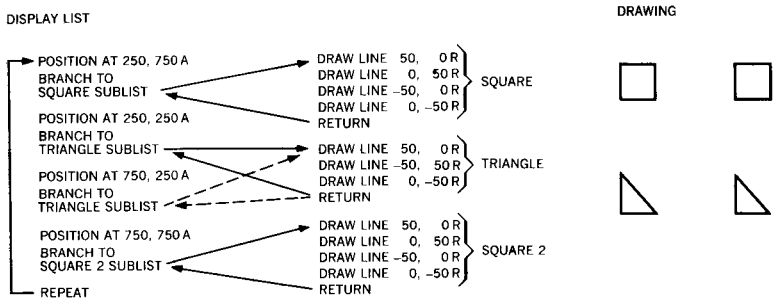


Figure 7 Translating a subpicture

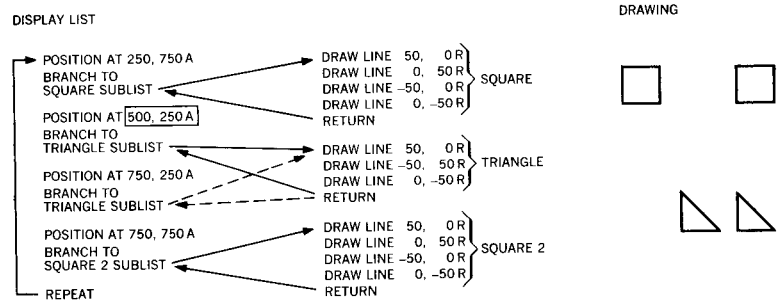


Figure 8 Scaling a subpicture

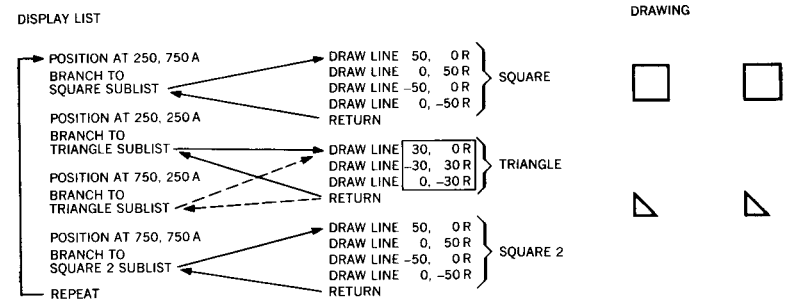


Figure 9 Erasing a line

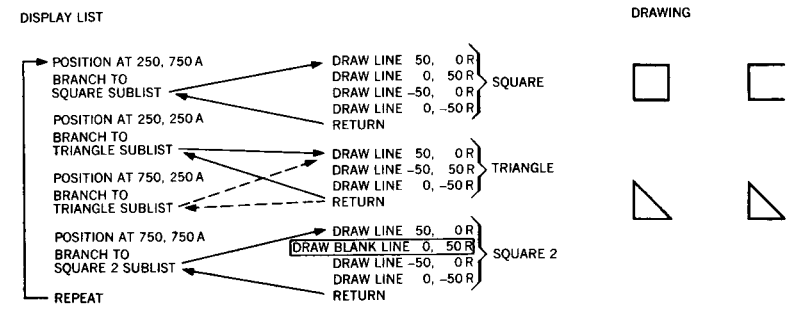


Figure 12 Subpictures, nets, and attacher points of a topological network

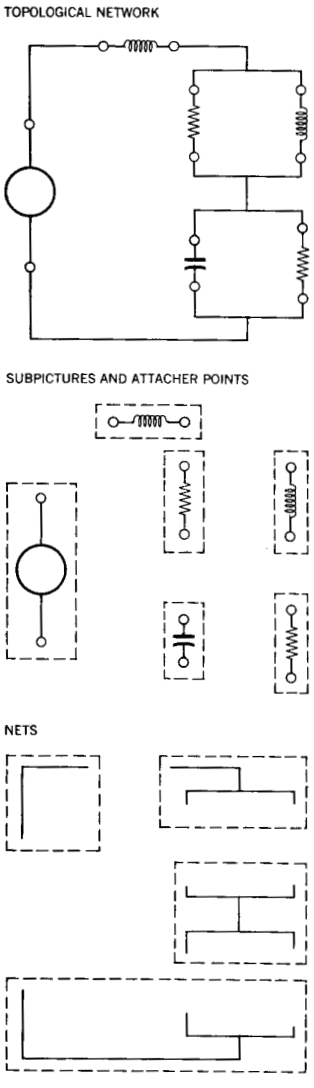


Figure 10 Deleting a line

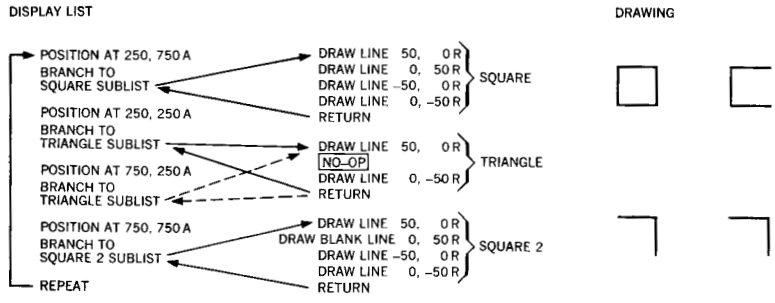


Figure 11 Deleting a subpicture

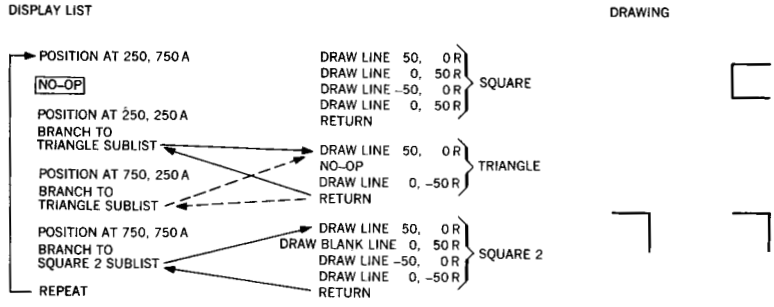
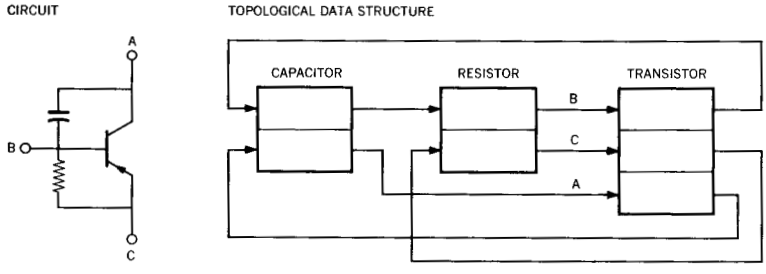


Figure 13 Circuit and topological data structure



A, B, C, REPRESENT ATTACHER POINTS

replaces a line with a blank line, and the DELETE function (in Figure 10) removes the line from the sequence of lines forming the subpicture. Both ERASE and DELETE are performed at the sublist level and are, therefore, reflected in all copies of the subpicture. Also, a subpicture can be removed by the DELETE function. As shown in Figure 11, the DELETE function removes the branch to the subpicture.

The DIM program uses the structured display list as its data structure, and provides a user with such conversational functions as just described. Other conversational functions, such as tracking, filing, and retrieving are also provided by DIM.

topological networks

Many graphics problems pertain to the analysis of *topological networks*, which are interconnections of elements where the position of the individual elements has no relevance. Such networks are completely specified by the types of elements and their interconnections. Examples of topological networks are electrical circuits, program flowcharts, and logic diagrams. Also, designers frequently abstract a practical problem by transforming it into a topological network. As an example, a mechanical circuit may be represented by a spring-mass-damper network.

Drawings of topological networks consist of *subpictures* (repeated where necessary) joined by *connecting lines*, which join the subpictures only at specified points called *attacher points*. The set of all connecting lines of a picture consists of disjointed connected subsets of lines, called *nets*. Figure 12 shows the nets and subpictures of a topological network. The topology of a picture is completely specified by the attacher points and the nets on which they lie.

topological data structure

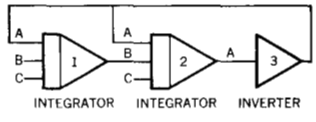
Such information can be stored in a ring-type data structure, called a *topological data structure*, which contains a group of words for each subpicture. Each group has a word, called a *link word*, for each attacher point in the corresponding subpicture. Link words for attacher points lying on a common net are tied together in a circularly linked list. In such a list, each link word contains a pointer to the next link word, and the last link word contains a pointer back to the first. An example of a topological data structure is shown in Figure 13.

analysis functions

An application program that can use such a data structure is the IBM 1130 Continuous Systems Modeling Program (CSMP). This program uses a digital computer to simulate the response of an analog computer. The analog computer configuration is specified to CSMP by a table containing a list of the analog blocks used. This is exemplified by Figure 14. Rows in the table are labeled with block numbers. Each row contains the name of the block type and the block numbers of other blocks whose outputs are connected to the inputs of the given block. For example, Row 2 in Figure 14 indicates that there is an integration block whose first input is the output of Block 3, whose second input is the output of Block 1, and which has no third input. (Note that the table is only a subset of the complete topological data structure.)

Figure 14 Specifying an analog circuit to 1130 CSMP

CIRCUIT



SPECIFICATIONS

BLOCK NUMBER	BLOCK TYPE	INPUTS A	B	C
1	INTEG.	3	X	X
2	INTEG.	3	1	X
3	INV.	2	X	X

The structured display list, created under light-pen control by DIM, contains picture-drawing instructions, but it does not contain explicit information about topological block interconnections within the picture. Although they appear on the display screen, the computer does not "know" about interconnections until it processes the display list. Next is described a method of extracting information about topological interconnections from the display list, thereby generating a data structure suitable for the CSMP program to analyze.

transforming data structures

One method of producing a topological data structure from a display list might be to scan the display list one time for each

relevant point of the picture (initial and terminal points of connecting lines, and attacher points of subpictures) to determine if any relevant points have the same coordinates. For complicated pictures, the number of relevant points (hence, the number of required scans) might become so large that processing time is prohibitive. Thus, a method of extracting topological data by relatively few scans is needed.

Such a method has been developed by utilizing certain properties of line drawings. Observe that a large percentage of the picture area of a line drawing is not used. Thus, if the picture area is finely divided, many of the divisions (*units*) are blank. If the area of each unit is decreased, the number of units and the percentage of blank units is increased. (A useful unit size was found to be one sixty-fourth of the area of the display screen.) If units containing no relevant points are also considered as blank units, the percentage of blank units is further increased. Because all information (i.e., relevant points) is in the non-blank units, picture processing time can be reduced by examining only those units.

It is desirable to set up a table for each non-blank unit in the picture. An initial pass is made through the display list to determine which are the non-blank units. When a relevant point is encountered, a check is made to determine whether the unit containing that point was previously encountered. The check is made by consulting a table, called a *table of headcells*, containing one word for each unit. (The headcell word contains all zeros if the unit was not previously encountered.) When a unit is encountered for the first time, a fixed-length group of words (called a *directory*) is reserved for that unit, and the address of the directory is entered into the corresponding word in the table of headcells. Information about relevant points lying within each non-blank unit is entered into the directories during the first pass. Included in the information are the relative coordinates of the point (measured from the lower left corner of the unit). The remaining information for endpoints of connecting lines is the display list address of the connecting line containing the endpoint. The remaining information for attacher points is the address of the link word of the attacher point. For attacher points, a pointer to the directory entry is entered into the link word of the attacher point. The pointers to directories must be replaced by pointers to other attacher points to put the topological data structure in final form.

Each connecting line requires two entries in the directories—one corresponding to the initial point of the line, and the other corresponding to the terminal point. However, if the preceding instruction in the display list also corresponds to a connecting line, the terminal point of the first line is the same as the initial point of the second line. In this case, no directory entry need be made for the initial point of the second line.

An example of a picture with its associated directories is shown in Figure 15. Note that a directory is not reserved for a unit until a relevant point lying in the unit is encountered. Since most units

Figure 15 Picture and associated directories

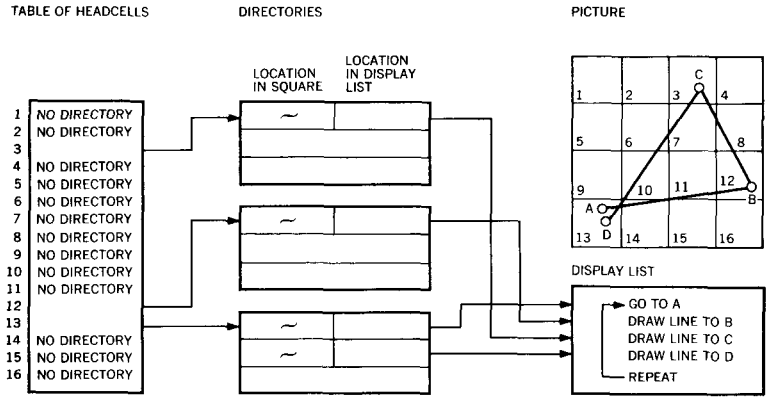
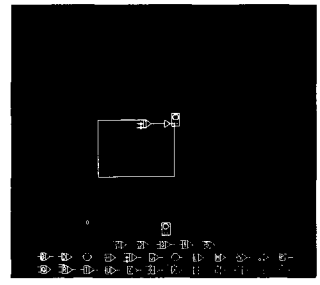


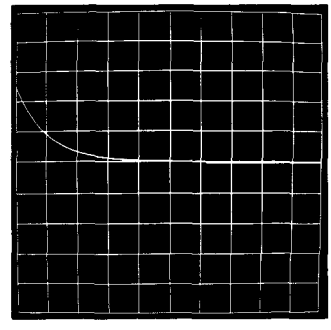
Figure 16 Analog computer configuration produced by DIM



are blank, relatively few directories are needed; hence, storage is not wasted on unused picture areas.

The topological data structure assumes its final form when link-word pointers to directories are replaced with pointers to other link words. This is done by tracing out the net for each attacher point. The first attacher point is selected, and the directory containing it is obtained from its link word. That directory is searched for other relevant points having the same relative coordinates; such relevant points all lie on the same net as the original attacher point. If one endpoint of a connecting line is found to have the required relative coordinates, the opposite endpoint also lies on the net. This opposite endpoint provides a new pair of coordinates for which another directory search for relevant points is performed. The search is repeated for each new pair of coordinates until no new relevant points lying on the net are obtained. This procedure, which is repeated once for each net, traces out the entire net and—at the same time—obtains the link words of all attacher points on the net. The link words are then tied together to form a circular linked list of attacher points on a common net.

Figure 17 Analog computer response generated by the 1130 CSMP



Concluding remarks

An experimental graphic facility has been described as an example of a multilevel modeling structure for interactive graphic design. The facility, consisting of the Display Image Manipulator coupled to the 1130 Continuous System Modeling Program, has been implemented on an IBM 1130 with 8K words of 16 bits each. The 1130 is attached to an IBM 2250 that uses the 1130 main storage as a display buffer.[6] The implemented facility allows a user to draw a configuration of analog elements. A display list is created by DIM during the drawing phase. After the picture is completed, a program converts the display list into a topological model, which CSMP then uses to obtain the response of the analog circuit.

Figure 18 First modified analog computer configuration

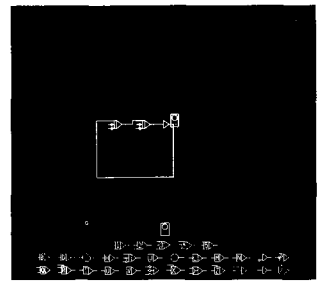


Figure 19 Computed response to first modification

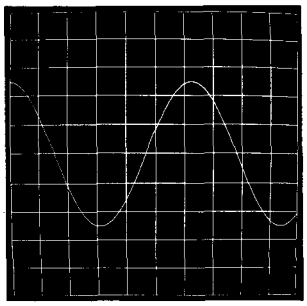


Figure 20 Second analog configuration modification

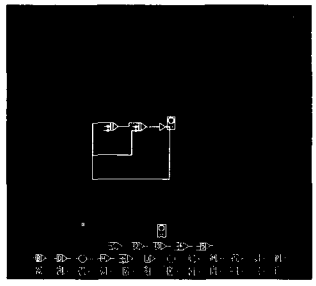
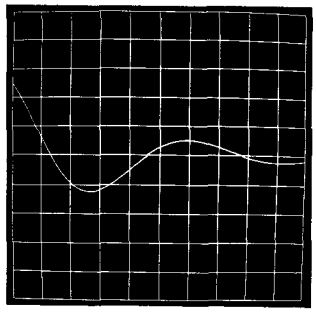


Figure 21 Computed response to second modification



A picture of an analog computer configuration drawn with DIM is shown in Figure 16, and the response generated by CSMP is shown in Figure 17. Figure 18 shows the configuration after a change has been made, and the response of the changed configuration is shown in Figure 19. A second analog configuration modification is shown in Figure 20, and its computed response is shown in Figure 21.

In the CSMP application, the interface program is required to be unidirectional. That is, the program must be able to generate the topological model from the display list, but not vice versa. In the present implementation, if the display list is changed after generating a topological model, a new topological model is generated. Future work should consider the problem of having changes in the display list reflected in the topological model, thereby making it unnecessary to generate new topological models after each display list change.

CITED REFERENCES AND FOOTNOTE

1. M. L. Dertouzos, "CIRCAL: On-line circuit design," *Proceedings of the IEEE* **55**, No. 5, 637–654 (May 1967).
2. F. S. Preston, et. al., *Development of Techniques for Automatic Manufacture of Integrated Circuits*, Technical Report AFML-TR-65-386, Volumes I & II, Electronics Branch, Air Force Materials Laboratory, Wright-Patterson AFB, Ohio, (November 1965). Performed under contract by Norden Division of United Aircraft Corporation.
3. M. L. Dertouzos and P. J. Santos, Jr., CADD: *On-Line Synthesis of Logic Circuits*, Electronic Systems Laboratory, Massachusetts Institute of Technology, Report ESL-R-253, Cambridge, Massachusetts (December 1965).
4. R. H. Riekert and D. V. Lieberman, DIM—*A Low Level Modeling System for Conversational Graphics*, IBM Research Report RC-1981, IBM T. J. Watson Research Center, Yorktown, New York (October 1967).
5. R. D. Brennan, "Digital simulation for control system design," *Proceedings of the* SHARE *Design Automation Workshop*, New Orleans, Louisiana, (May 1966).
6. The work described in this paper was implemented on a prototype IBM 1130/2250 system. Therefore, these programs are not currently available for the IBM 2250 Model 4.