For the ACM SIGPLAN Symp.
on Two-Dimensional Man-Machine
Communication, Los Alamos, NM
Oct. 5-6, 1972

LBL-1008
Preprint c. 2

PICASSO
A GENERAL GRAPHICS MODELING PROGRAM

H. H. Holmes and D. M. Austin

July 1972

AEC Contract No. W-7405-eng-48

32

PICASSO
A GENERAL GRAPHICS MODELING PROGRAM

by

H. H. Holmes and D. M. Austin
Lawrence Berkeley Laboratory, University of California
Berkeley, California 94720

ABSTRACT

A new two-dimensional graphics modeling program which has been fully implemented is introduced. Features include a generalized extendable graphics editor, an open-ended library of elements, a translator-macro processor for interface with a wide range of analysis routines, and on-line operating system control.

I. Introduction

Two-dimensional communication may be considered as the transmitting of information by pictures. One form of picture information is position, or coordinate information, as used in architectural or circuit board design and map-making. Another form is symbolic pictures, or modeling.

Modeling is the construction of a symbolic representation of a hypothetical reality. Graphics modeling is especially suited to models composed of many elements, where each element corresponds to a simple, well-defined operation, and where the investigator is most interested in the elementary operations required to mimic some natural or artificial phenomena.

In the following, a brief review of graphics modeling programs is presented and a new modeling system incorporating both forms of picture communication is described.

II. Graphics Modeling Programs before PICASSO

The first graphic modeling program was SKETCHPAD [Sutherland, 1963]. It manipulates lines, constraints, and subpictures (which may include other subpictures), and evaluates the constraints to produce a drawing which satisfies them. We may characterize SKETCHPAD by noting that a single data structure and a single analysis routine are used. Following SKETCHPAD, there appeared programs with more practical analysis routines, such as CIRCAL [Dertouzos, 1967], CADIC [Preston, 1963] and CADD [Dertouzos, 1965]. Baskin and Morse [Baskin, 1968] introduce the idea of a partition in the set of modeling functions as well as a multilevel data structure. A conversational program (DIM) [Riekert, 1967] creates the graphic data structure, an intermediate routine analyzes the topology and creates a list of elements (second level data structure), and a batch type simulator (CSMP) [Brennan, 1966] evaluates the elements. This work was extended to the development of an experimental block diagram program called DESIGNPAD [Baskin, 1969 and Belady, 1971] which was designed as a more general two-system (host-satellite) graphics package with an interface data file accessible by host-resident analysis programs.

Past work has suffered from a lack of modularity and a lack of extension capability. The programming required for one application is not easily transferred to another. More importantly, they lack a facility for user extension of the set of primitive elements, the building blocks of the system.

III. Requirements of a General Graphics Modeling System

A comprehensive system of interactive graphics requires facilities which allow the user to draw his model and "plug in" the appropriate analysis package. To implement this idea, we must separate the model building facilities from the analysis routines and define a simple and concise interface between the modules. With a single graphics package for all analysis routines, we can justify making it very elaborate and general. A translator is required to mediate between the graphics data structure(s) and existing analysis routines, which expect input in the form of card images. Such a translator also provides a convenient starting point for writing new analysis routines.

A second requirement is an open-ended library of elements which may be altered or supplemented by the user whenever necessary. The alteration must, of course, be reflected in every instance of the element without further user intervention.

A third requirement is hard copy documentation of both the graphics and analysis packages. Film output (including microfilm and microfiche) seems the most desirable form, but printer output should also be available on demand.

A fourth requirement is permanent storage with easy access for the data structure generated. Data cell, chipstore, magnetic tape or punched cards (in decreasing order of convenience) are examples of permanent storage facilities available.

An additional feature which should be included is real-time control of the job process, so that model editing and analysis may be done sequentially in real time for maximum feedback to the user.

Other useful features include multiple definitions of elements, so that a single model may be analyzed by several programs, and user-extendible graphic command structures.

IV. PICASSO

PICASSO [Austin, 1972] is designed to be a general interactive graphics program capable of producing card image and other forms of input for a variety of analysis routines. Initially, we will distinguish two broad classes of user activities: the creation of primitive elements, and modeling, the interconnection of established primitive elements. PICASSO,however, does not distinguish between these activities, and so they gradually blur together, with the experienced user working in both areas simultaneously.

We next describe how the graphic and text editing facilities in PICASSO are used to create a primitive element, and the meaning associated with these operations.

After the user names a primitive element, the graphic editor is used to draw a symbol for it. For many disciplines (e. g., electronic circuits, digital logic diagrams) the choice of a symbol is obvious. The user must then specify the formal parameters for the definition. These parameters take the form of labels attached to the sketch of the symbol at strategic points. These labels, by their location on the symbol, define points of reference (attacher points) where the parameter may be referenced when an instance of the symbol appears in a model (or macro definition). The user must now create a definition for his new element. Using the text editor, he may write a mathematical expression, or other text using the format and syntax of the input language for the analysis routine which he desires to use. A simple symbol and its text definition are illustrated in Fig. 1. Usually, most of the common definitions will have been created, and the new user can borrow these without having to start from scratch. If an empirical definition is desired, a curve may be drawn

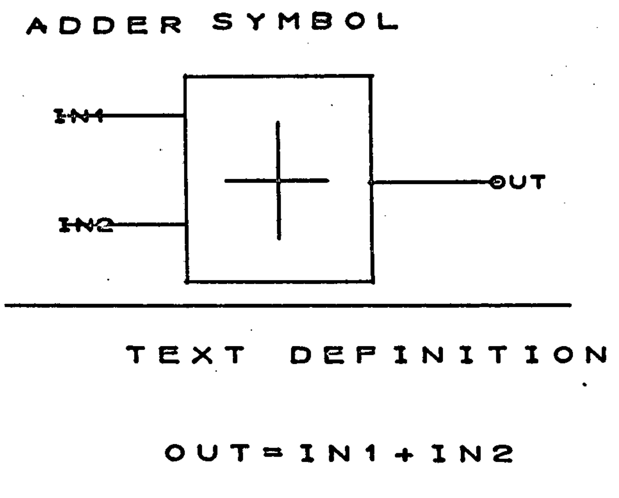


Fig. 1. A symbol and definition of an element named ADDER.

on the screen, or a curve may be created from data in auxiliary storage. The graphics editor can supply a grid overlay and labels may be used to establish scales. To define an element in terms of existing elements (a macro definition) the modeling techniques to be described are used.

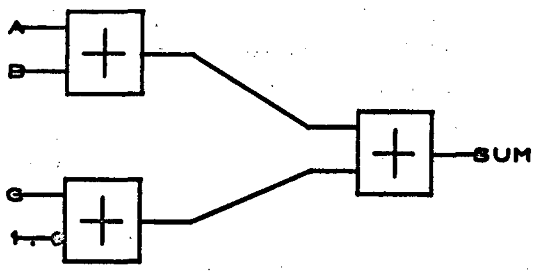
Models are created by composing symbols, lines and labels to describe graphically the desired arrangement of elements. Symbols are selected (by name) and placed on the screen with the light pen. A symbol used in a model represents a call on its definition. Line segments are drawn connecting the attacher points of the symbols. A set of attacher points connected by line segments constitutes a net; each net represents a flow of information between the attacher points on the net. The labels for a net may be placed on either lines or attacher points; different labels on the same set are equivalenced by the translator. The label for a net may be described as the actual parameter which will replace the formal parameter (associated with the particular attacher point) when the instance of the primitive element is processed by the translator. Thus a net which topologically joins a set of attacher points causes them all to refer to the same piece(s) of information. For example, a net will typically consist of a line joining the output of one element to the input of another element, and the net label will replace the corresponding formal parameters of both attacher points when the translator analyzes the model.

Figure 2 shows three adders joined by nets, and the translator output corresponding to this model. Disjoint sets may also be connected by means of identical labels instead of by line segments. This feature facilitates model construction on a " sheet" which is too large to appear on the screen.

By means of a special convention in the translator labels may represent either local or global parameters. The use of labels in specifying parameters also allows analysis routines to recognize special variables (such as the time variable T in MIMIC).

The data structure includes a directory of elements, a dynamic storage area, a label storage area and a working area. The directory contains the name and definition type (macro, empirical or

ADD3 MACRO DEFINITION



TRANSLATOR OUTPUT

```
G001=A+B

G002=C+1.0

SUM=G001+G002
```

Fig. 2. The model named ADD3 and the translator output.

text) of the element and pointers to the location in the dynamic storage area of its symbol, its definition and its topology (not all of these components need exist for a given element of course). When a symbol or definition is to be edited (or created), the data is copied from the storage area into the working area and control is passed to either the graphics editor (for symbols, macro and empirical definitions) or the text editor. When editing is complete, the data is copied from the working area to the storage area and the directory pointers are updated. The storage area is dynamic in the sense that the program automatically captures the memory required to store the data, and releases surplus memory to the system, so that the program always occupies the minimum amount of memory.

The graphics editor creates lines, labels and symbol calls and stores them in the working area. These components are encoded in pseudo display instruction format; each item requires one machine word. A line consists of a type field, and four 12 bit coordinates specifying the endpoints. A label consists of a type field, two coordinates specifying the position, size and orientation fields and a pointer to the label storage area. A symbol call is similar to the label item, except the pointer indicates the position in the directory of the symbol name. Data points for empirical curves are stored in floating point form.

Text definitions are created in the text editor and stored in compressed display code format with a word count.

When a model is analyzed, its topology is encoded and stored for later use by the translator. Nets are numbered sequentially and a list of nodes is stored in the dynamic storage area. A new topological analysis is done only when the model has been re-edited and a new analysis is called for.

All graphic interactions are under the control of the graphics editor, wherein symbols, models (macro definitions) and empirical definitions are created and edited. The graphics editor uses two types of commands - light pen commands and teletype (or keyboard) commands. The teletype must be used to specify symbol names and other text-string type data (labels, numbers). A list of command words appears on the screen for implementing the graphic features, such as zooming, grid overlay, erasing, etc. Subeditors are called for the placing of labels and symbols. These editors display their own list of commands appropriate to their functions (such as specifying the size and orientation of labels and symbols). The zoom feature effectively makes the "work sheet" up to 64 times the size of the screen. The grid overlay can be specified with arbitrary spacing between dots, which facilitates accurate drawing. Other commands allow for variable line-straightening parameters, an absolute scale factor for use with the grid overlay and an external picture syntax for I/O of the drawing. The latter feature is useful for the construction of accurate drawings (via teletype or card image) when the zoom and grid overlay features are inadequate. Pictures converted to card images can be edited, copied into other drawings, or serve as input to analysis routines.

The text editor was designed primarily for teletype input of text definitions, but the inclusion of powerful I/O facilities have made it useful for many other duties, such as creating on-line control card sequences to guide the operating system, displaying the output of analysis programs, editing the external picture card images produced in the graphics editor and creating and releasing disk files for temporary storage. These features make most of the BKY system facilities available on-line as needed.

The translator analyzes a model by using the output of the topology routine for that model. In the translator's interpretation the model represents a series of subroutine (or macro) calls on the elements represented by the symbols. These in turn may be defined as a set of calls on elements. Thus the translator produces a tree structure of subroutine calls, an interpretation which is commonly desired, easily perceived and understood by the user and yet difficult to state computationally. Equally important are the features of the translator which make it easier to create models. For example, for one-node nets (unused attacher points), the translator recognizes default values specified by appending a slash and a default string to the formal parameter ("name" becomes "name/1.0").

The first phase of the translator is the extraction of the topology from the drawings which compose the model. All drawings which have changed since the last translation are reanalyzed at this time. Each drawing is copied into the working area, with the pseudo-display instructions hashed by their coordinates. For each symbol in the drawing, the locations of its attacher points are computed and these are added to the working area. The routine then matches coordinates and follows lines to create a list of nodes belonging to a particular net. A flag is set to distinguish nets with only one node, and the list of nodes is then stored in the dynamic storage area. Analysis programs may examine this topology directly if they desire, bypassing the macro expansion part of the translator.

After the topological analysis is complete, the translator scans the template and records information about how to process the model. Figure 3 illustrates such a templete. Included in the template are header and trailer cards for the main program and subroutines (or macros) as well as information on how to call subroutines. The

```
*IAM          MIMIC RUN OF * PRODUCED BY MIMVERT
                          END
              *           BMA(
                          EMA
              *           CMA(
              *           CFN(<,0)
              DATA/R        NOEOF       DATA       G

SFL,70000.
REWIND,DATA.
MIMGO,DATA,MIM.
SFL,55000.
DRAW.
```

Fig. 3. The template for producing MIMIC programs.

translator inserts names (in place of the asterisks) and parameters into these template specifications during expansion of the drawings and text definitions. Also included is the set of control cards necessary to run the analysis program. These cards are written to the control file and will be executed when the modeling program (PICASSO) terminates. After reading the template the translator uses the topological analysis to compile a list of all elements which are used directly or indirectly in the model. These are sorted so that elements are defined before they are used, a requirement of many analysis programs (e. g. MIMIC, ALGOL). For each element on the list, the translator scans the topology for the element and assigns a name to every net. If a label is on the net, then that becomes the name for the net. If no name is supplied, the translator generates one as specified by the template (see Fig. 3). These names then become the actual parameters for all the formal parameters (attacher points) on the net. The translator then creates a call on each of the subelements of the element with the proper actual parameters. If an attacher point was not connected, the translator examines the subelement for default specifications. The user may elect to have the translator do the actual string substitution of actual parameter for formal parameter in the definitions. This facility is a necessity, since very few analysis routines understand subroutine (or macro) calls.

Collections of named libraries are stored on a random access disk file while the program is operating. Each named library is a complete group of elements and models. A user may "load" a library, modify it, and save it under a different name, thus preserving the original set of elements. Several libraries may be loaded simultaneously, combining the libraries so that all the elements are accessible together. Permanent storage is available on a Data Cell which is manipulated by control cards. Ordinarily, a collection of libraries is copied from the Data Cell to disk at the beginning of a session and the new libraries are replaced at the end of the session. These operations may be performed at any time, however.

PICASSO provides documentation in the form of microfilm and microfiche for both the graphics and text. Conventional line printer output is also available for the text. Xerox copies of the microfilm and microfiche may be made and most users have microfiche viewers.

V. PICASSO on BKY Operating System

PICASSO is implemented on a CDC 6600 computer with a VISTA 250 display system and is currently used with MIMIC [Control Data, 1968], CORNAP [Pottle, 1966], Fortran, and two coordinate-driven analysis routines. Less than two weeks were spent installing CORNAP as an analysis routine. Most of the time was spent testing various sequences of control cards, learning about CORNAP and its limitations, and creating a library of primitive elements (resistors, capacitors, transistors, etc.) for users. Our experience indicates that PICASSO (compared to card input) is much more likely to produce a working model the first time around, because it keeps track of all the details (especially calling sequences) and doesn't make typing mistakes. We have also found that naive users can construct models from existing elements, but do poorly at defining new primitive elements until they have had some experience. On the other hand, an experienced user can construct definitions which appear to make a given analysis routine act in a completely different manner. Thus, a simulator for continuous systems (MIMIC) can be made to evaluate digital logic.

ACKNOWLEDGMENT

Work performed under the auspices of the U.S. Atomic Energy Commission.

APPENDIX

A more extensive model is illustrated in Figs. A1 through A4. This is a model of water and pollution flow in the San Francisco Bay region, and illustrates a type of compartmental modeling which is very common in biological applications [Horovitz, 1972]. The model is composed of Nodes, which represent the water within a given partition, and Channels which direct the flow of water between the Nodes. Nodes and Channels are documented in Figs. A3 and A4. Also included are several other components to simulate the tides, the flow of rivers, and to allow the analysis program (MIMIC) to recognize input and output quantities. Approximately 100 card images are generated by the translator for this model. Figure A2 shows one of the tides and pollution histories for three of the Nodes.

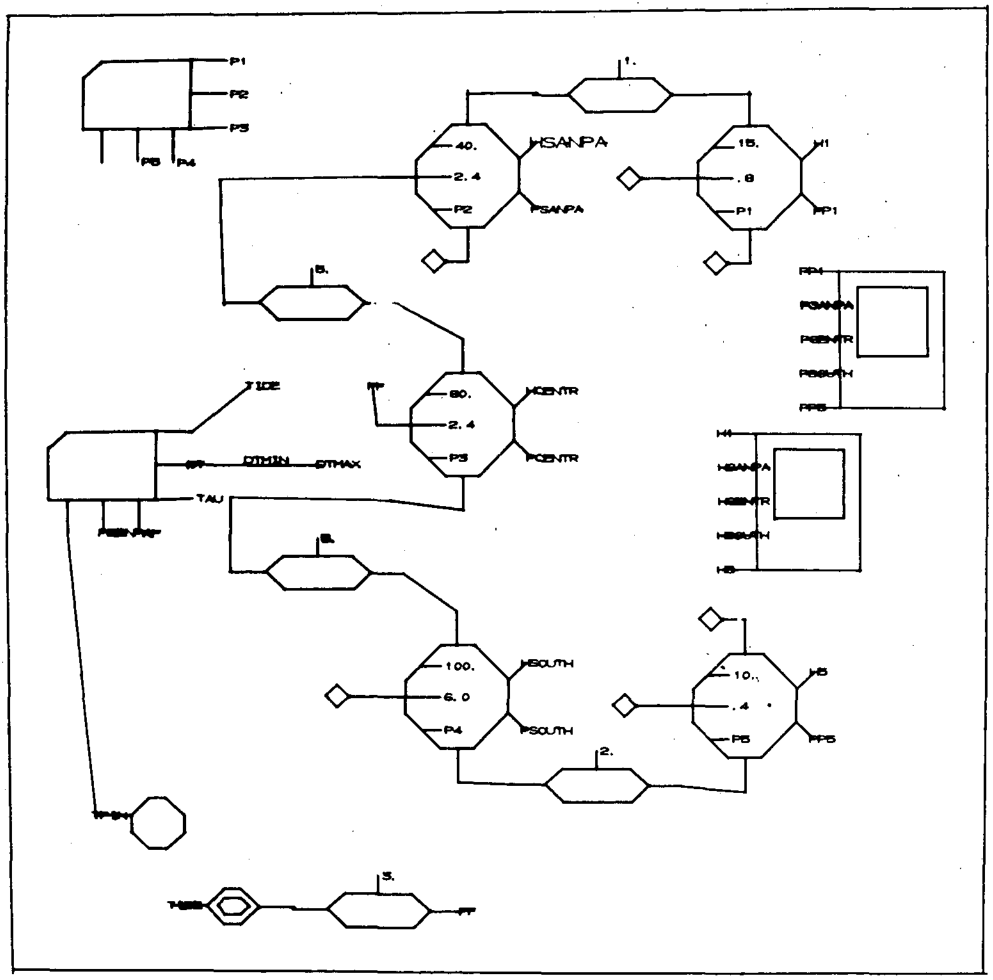


Fig. A1. A water and pollution flow model of the San Francisco Bay region.

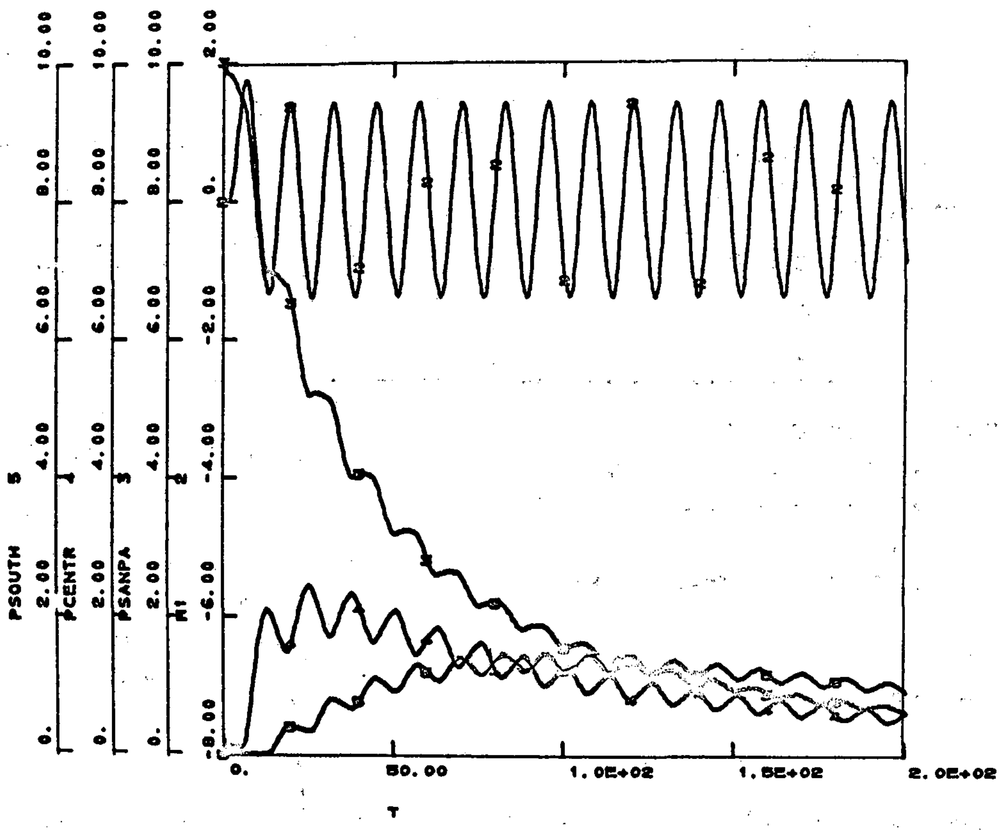
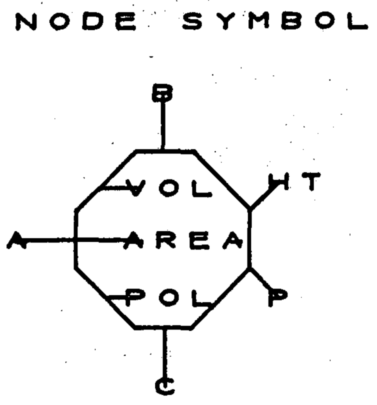


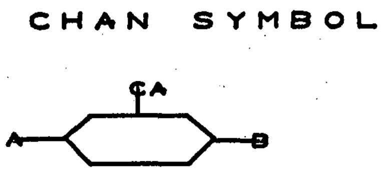
Fig. A2. MIMIC plot for a representative tide and pollution in three San Francisco Bay regions.



TEXT DEFINITION

V	INT(L→A+L→B+L→C, VOL)
HT	(V-VOL)/AREA
H→A	HT
P	INT(F→A+F→B+F→C, PLO)
MT	P/V
M→A	MT
M→B	MT

Fig. A3. A node and its definition in terms of MIMIC integration functions. Note use of the translators concatenation function (→).



TEXT DEFINITION

DH	FTR(H→A-H→B, TAU)
L→B	DH*CA*KF
L→A	-L→B
ZM	MAX(DH+KD, 0.)*M→A
F→B	(ZM-MAX(-DH+KD, 0.)*M→B)*CA*KF
F→A	-F→B

Fig. A4. The channel symbol and its definition in terms of MIMIC first-order transfer function.

REFERENCES

1. Ivan E. Sutherland, SKETCHPAD-A Man-Machine Graphical Communication System, AFIPS Conference Proceedings, Spring Joint Computer Conference 23, 329-346 (1963).
2. M. L. Dertouzos, "CIRCAL: On-line circuit design," Proceedings of the IEEE 55, No. 5, 637-654 (May 1967).
3. 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 (Nov. 1965).
4. 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.
5. H. B. Baskin and S. P. Morse, A Multilevel Modeling Structure for Interactive Graphic Design. IBM Systems Journal, No. 3 & 4, 1968.
6. 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 (Oct. 1967).
7. R. D. Brennan, "Digital Simulation for Control System Design," Proceedings of the SHARE Design Automation Workshop, New Orleans, Louisiana (May 1966).
8. Donald M. Austin and Harvard H. Holmes, PICASSO: A General Interactive Graphics Modeling Program, LBL-580, University of California, Lawrence Berkeley Laboratory, Berkeley, California (Jan. 1972).
9. CONTROL DATA MIMIC; A Digital Simulation Language, Reference Manual, Publication number 44610400. Control Data Corporation, Special Systems Publications, St. Paul, Minnesota, (Apr. 1968).

10. Program CORNAP, Systems Theory Research Group, School of Electrical Engineering, Cornell University, based on:
C. Pottle, State-Space Techniques for General Active Network Analysis: Chapter 3 of "System Analysis by Digital Computer," F. F. Kuo and J. F. Kaiser, Eds., Wiley, 1966; pp. 59-98.

11. H. B Baskin and L. A. Belady, "DESIGNPAD: A Graphic Design/Problem-Solving Facility", Proceedings of the Third Annual Princeton Conference on Information Sciences and Systems, 173 (1969).

12. L. A. Belady, M. W. Blasgen, C. J. Evangelist, and R. D. Tennison, A Computer Graphics System for Block Diagram Problems, IBM Systems Journal 10, No. 2, 143-161 (1971).

13. Mark W. Horovitz, Donald Austin and Harvard Holmes, Symbolic Computer Graphics and Biological Models, ACM/SIGGRAPH Symposium, Pittsburgh, Pa. (March, 1972).