SKETCHPAD, A MAN-MACHINE GRAPHICAL COMMUNICATION SYSTEM

by

IVAN EDWARD SUTHERLAND

B.S., Carnegie Institute of Technology

(1959)

M.S., California Institute of Technology

(1960)

SUBMITTED IN PARTIAL FULFILLMENT OF THE

REQUIREMENTS FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

January, 1963

Signature of Author.....
Department of Electrical Engineering, January 7, 1963

Certified by.....
Thesis Supervisor

Accepted by.....
Chairman, Departmental Committee on Graduate Students

SKETCHPAD, A MAN-MACHINE GRAPHICAL COMMUNICATION SYSTEM

by

IVAN EDWARD SUTHERLAND

Submitted to the Department of Electrical Engineering on January 7, 1963, in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

ABSTRACT

The Sketchpad system uses drawing as a novel communication medium for a computer. The system contains input, output, and computation programs which enable it to interpret information drawn directly on a computer display. It has been used to draw electrical, mechanical, scientific, mathematical, and animated drawings; it is a general purpose system. Sketchpad has shown the most usefulness as an aid to the understanding of processes, such as the motion of linkages, which can be described with pictures. Sketchpad also makes it easy to draw highly repetitive or highly accurate drawings and to change drawings previously drawn with it. The many drawings in this thesis were all made with Sketchpad.

A Sketchpad user sketches directly on a computer display with a "light pen." The light pen is used both to position parts of the drawing on the display and to point to them to change them. A set of push buttons controls the changes to be made such as "erase," or "move." Except for legends, no written language is used.

Information sketched can include straight line segments and circle arcs. Arbitrary symbols may be defined from any collection of line segments, circle arcs, and previously defined symbols. A user may define and use as many symbols as he wishes. Any change in the definition of a symbol is at once seen wherever that symbol appears.

Sketchpad stores explicit information about the topology of a drawing. If the user moves one vertex of a polygon, both adjacent sides will be moved. If the user moves a symbol, all lines attached to that symbol will automatically move to stay attached to it. The topological connections of the drawing are automatically indicated by the user as he sketches. Since Sketchpad is able to accept topological information from a human being in a picture language perfectly natural to the human, it can be used as an input program for computation programs which require topological data, e.g., circuit simulators.

Sketchpad itself is able to move parts of the drawing around to meet new conditions which the user may apply to them. The user indicates conditions with the light pen and push buttons. For example, to make two lines parallel, he successively points to the lines with

the light pen and presses a button. The conditions themselves are displayed on the drawing so that they may be erased or changed with the light pen language. Any combination of conditions can be defined as a composite condition and applied in one step.

It is easy to add entirely new types of conditions to Sketchpad's vocabulary. Since the conditions can involve anything computable, Sketchpad can be used for a very wide range of problems. For example, Sketchpad has been used to find the distribution of forces in the members of truss bridges drawn with it.

Sketchpad drawings are stored in the computer in a specially designed "ring" structure. The ring structure features rapid processing of topological information with no searching at all. The basic operations used in Sketchpad for manipulating the ring structure are described.

Thesis Supervisor: Claude E. Shannon

Title: Donner Professor of Science

ACKNOWLEDGEMENTS

I am indebted to Professors Claude E. Shannon and Marvin Minsky for their help and advice throughout the course of this research. Their helpful suggestions at several critical times gave Sketchpad much of its present character.

Special thanks are due to Professor Steven A. Coons of the Mechanical Engineering Department and to Douglas T. Ross of the Electronic Systems Laboratory. Even though I was outside their Computer Aided Design group, they provided at least as unstintingly of their time and ideas as if I had been their only concern.

I owe a great debt to the MIT Lincoln Laboratory for the tremendous support it afforded me. I wish to thank Wesley A. Clark and Jack L. Mitchell for making the TX-2 computer available to me and for providing help to make the special equipment I needed. I appreciate the helpful suggestions and interest that they and all the members of Group 51 provided. Special thanks are due Leonard M. Hantman for the additions he made to Sketchpad.

The Research Laboratory of Electronics at MIT provided me with office space and congenial office mates whose discussion and interest I greatly appreciate.

Finally, I wish to thank Lawrence G. Roberts who was a constant source of answers to specific questions I had both about the best ways to program TX-2 and about the mathematics of difference equations and matrix manipulations.

TABLE OF CONTENTS

LIST OF FIGURES

Chapter I

INTRODUCTION

The Sketchpad system makes it possible for a man and a computer to converse rapidly through the medium of line drawings. Heretofore, most interaction between men and computers has been slowed down by the need to reduce all communication to written statements that can be typed; in the past, we have been writing letters to rather than conferring with our computers. For many types of communication, such as describing the shape of a mechanical part or the connections of an electrical circuit, typed statements can prove cumbersome. The Sketchpad system, by eliminating typed statements (except for legends) in favor of line drawings, opens up a new area of man-machine communication.

The decision actually to implement a drawing system reflected our feeling that knowledge of the facilities which would prove useful. could only be obtained by actually trying them. The decision actually to implement a drawing system did not mean, however, that brute force techniques were to be used to computerize ordinary drafting tools; it was implicit in the research nature of the work that simple new facilities should be discovered which, when implemented, should be useful in a wide range of applications, preferably including some unforseen ones. It has turned out that the properties of a computer drawing are entirely different from a paper drawing not only because of the accuracy, ease of drawing, and speed of erasing provided by the computer, but also primarily because of the ability to move drawing parts around on a computer drawing without the need to erase them. Had a working system not been developed, our thinking would have been too strongly influenced by a lifetime of drawing on paper to discover many of the useful services

that the computer can provide.

As the work has progressed, several simple and very widely applicable facilities have been discovered and implemented. They provide a subpicture capability for including arbitrary symbols on a drawing, a constraint capability for relating the parts of a drawing in any computable way, and a definition copying capability for building complex relationships from combinations of simple atomic constraints.* When combined with the ability to point at picture parts given by the demonstrative light pen language, the subpicture, constraint, and definition copying capabilities produce a system of extraordinary power. As was hoped at the outset, the system is useful in a wide range of applications, and unforseen uses are turning up.

AN INTRODUCTORY EXAMPLE

To understand what is possible with the system at present let us consider using it to draw the hexagonal pattern of Figure 1.1. We will issue specific commands with a set of push buttons, turn functions on and off with switches, indicate position information and point to existing drawing parts with the light pen, rotate and magnify picture parts by turning knobs,and observe the drawing on the display system. This equipment as provided at Lincoln Laboratory's TX-2 computer[1] is shown in Figure 1.2. When our drawing is complete it may be inked on paper, as were all the drawings in the thesis, by the plotter[12] shown in

---

* Terms with specialized meanings are listed in the glossary at the very end of this thesis.

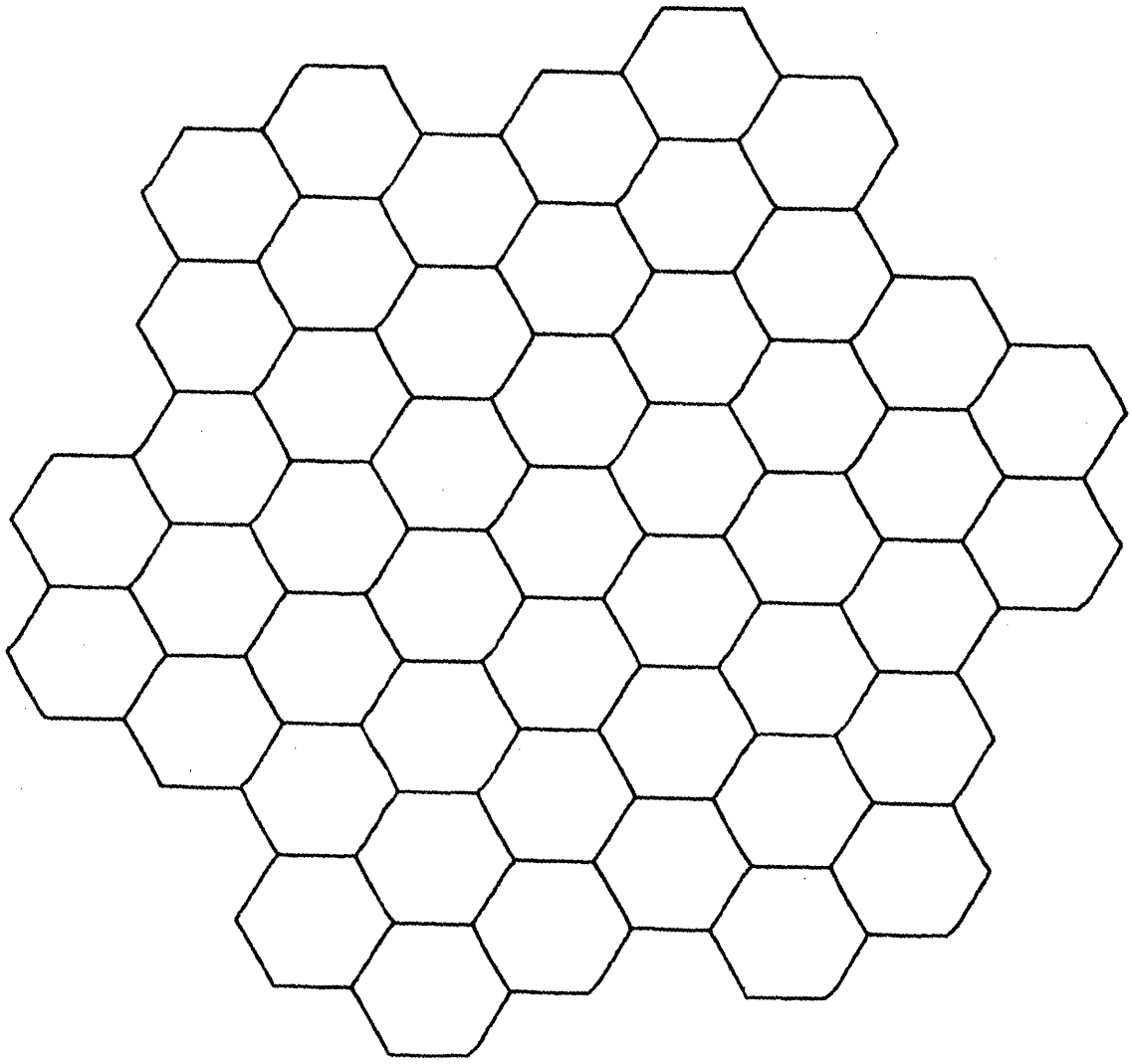


FIGURE 1.1. HEXAGONAL PATTERN

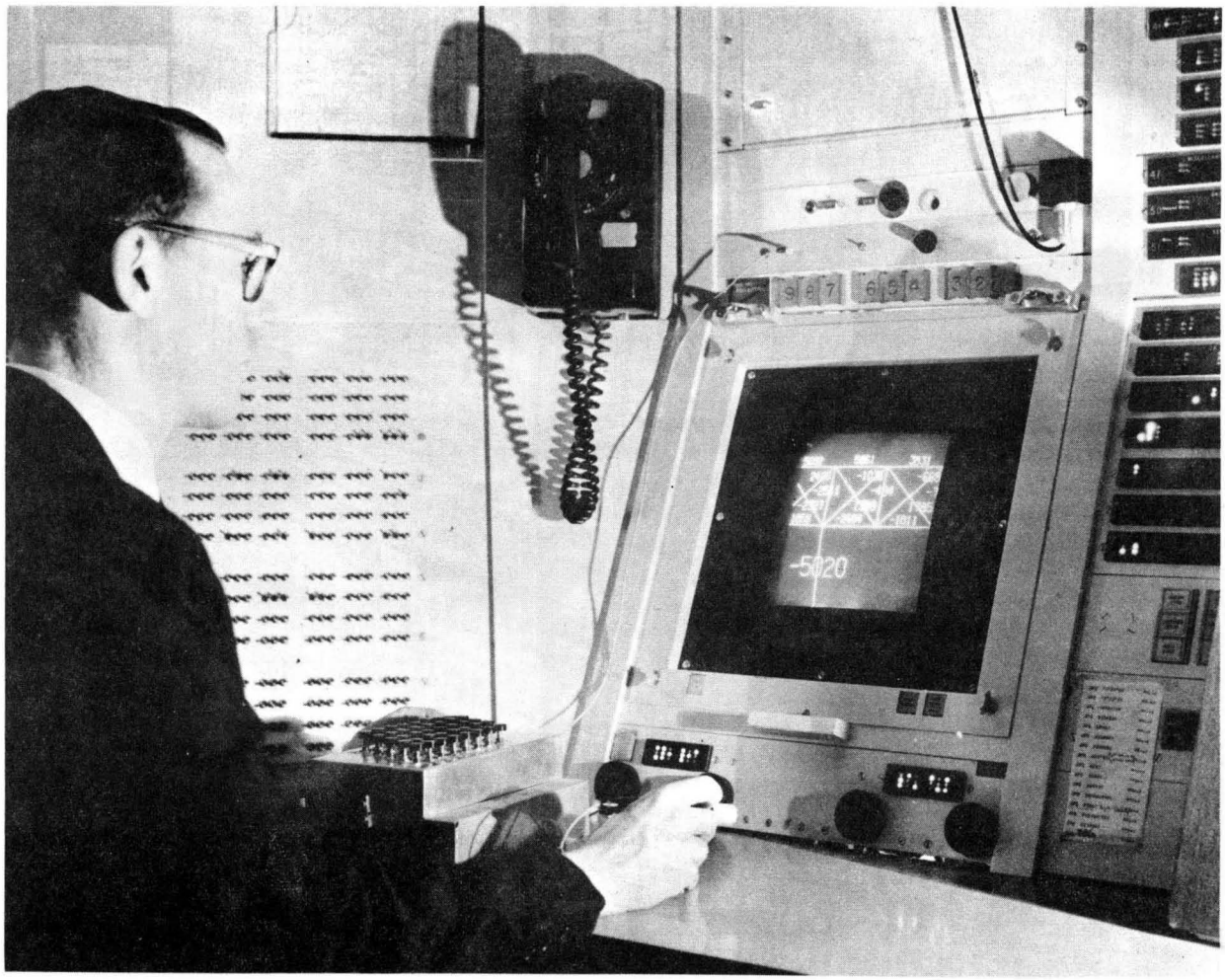


FIGURE 1.2. TX-2 OPERATING AREA - SKETCHPAD IN USE.
On the display can be seen part of a bridge similar to that of Figure 9.6. The Author is holding the Light pen. The push buttons used to control specific drawing functions are on the box in front of the Author. Part of the bank of toggle switches can be seen behind the Author. The size and position of the part of the total picture seen on the display is obtained through the four black knobs just above the table.

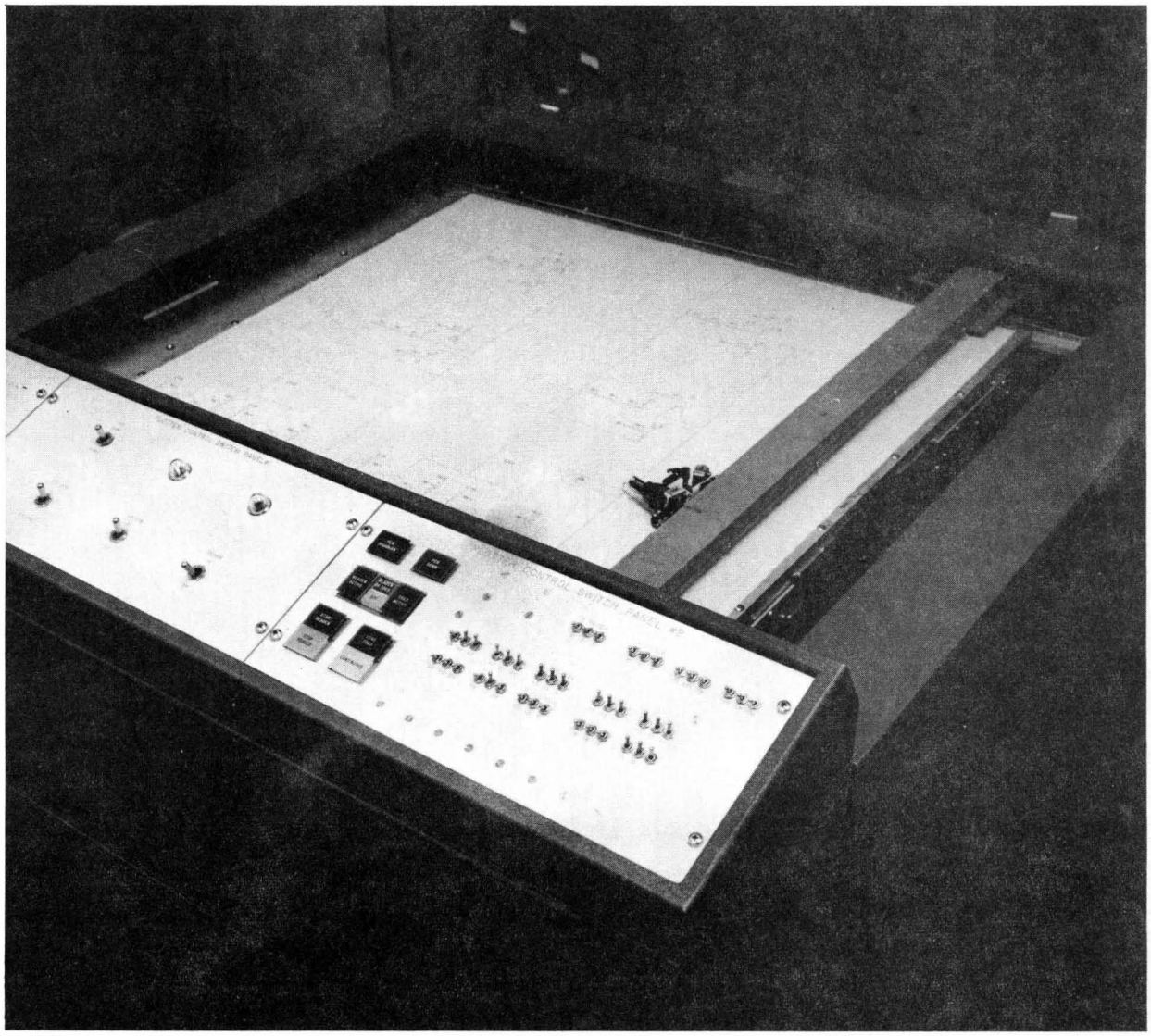


FIGURE 1.3. PLOTTER USED WITH SKETCHPAD
A digital and analog control system makes the plotter draw straight lines and circles either under direct control of the TX-2 or off-line from punched paper tape.

Figure 1.3. It is our intent with this example to show what the computer can do to help us draw while leaving the details of how it performs its functions for the chapters which follow.

If we point the light pen at the display system and press a button called "draw", the computer will construct a straight line segment* which stretches like a rubber band from the initial to the present location of the pen as shown in Figure 1.4. Additional presses of the button will produce additional lines until we have made six, enough for a single hexagon. To close the figure we return the light pen to near the end of the first line drawn where it will "lock on" to the end exactly. A sudden flick of the pen terminates drawing, leaving the closed irregular hexagon shown in Figure 1.5A.

To make the hexagon regular, we can inscribe it in a circle. To draw the circle we place the light pen where the center is to be and press the button "circle center", leaving behind a center point. Now, choosing a point on the circle (which fixes the radius,) we press the button "draw" again, this time getting a circle arc* whose length only is controlled by light pen position as shown in Figure 1.4.

Next we move the hexagon into the circle by pointing to a corner of the hexagon and pressing the button "move" so that the corner follows

---

*The terms "circle" and "line" may be used in place of "circle arc" and "line segment" respectively since a full circle in Sketchpad is a circle arc of 360 or more degrees and no infinite line can be drawn.

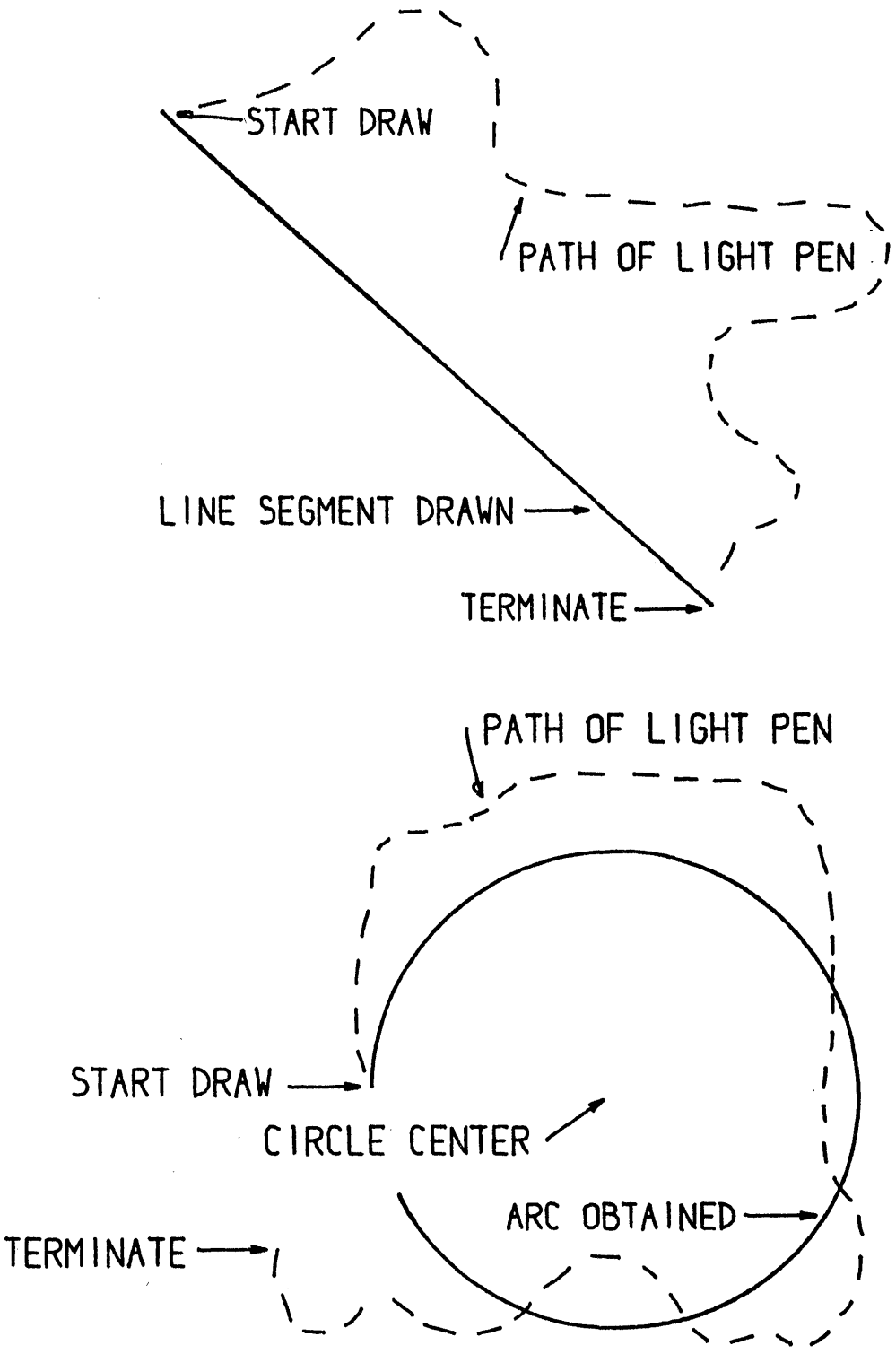
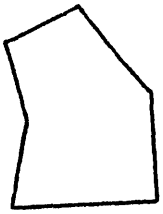
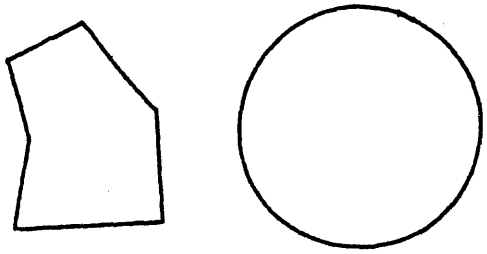


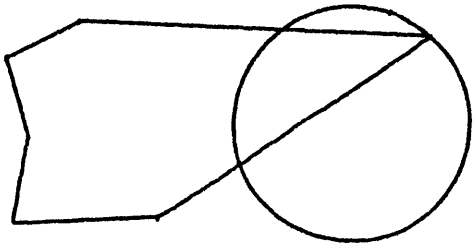
FIGURE 1.4.
LINE AND CIRCLE DRAWING



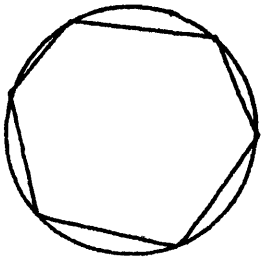
A. SIX SIDED FIGURE



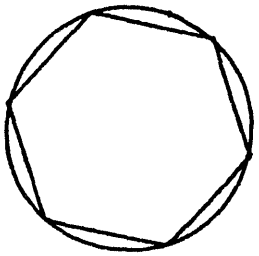
B. TO BE INSCRIBED IN CIRCLE



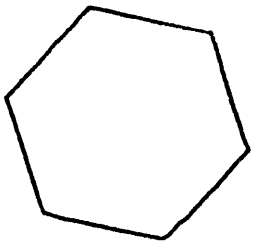
C. BY MOVING EACH CORNER



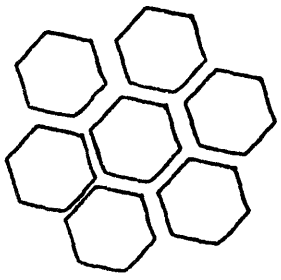
D. ON TO CIRCLE



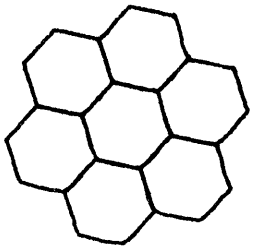
E. MAKE SIDES EQUAL



F. ERASE CIRCLE



G. CALL 7 HEXAGONS



H. JOIN CORNERS

FIGURE 1.5. ILLUSTRATIVE EXAMPLE

the light pen, stretching two rubber band line segments behind it. By pointing to the circle and giving the termination flick we indicate that the corner is to lie on the circle. Each corner is in this way moved onto the circle at roughly equal spacing around it as shown in Figure 1.5D.

We have indicated that the vertices of the hexagon are to lie on the circle, and they will remain on the circle throughout our further manipulations. If we also insist that the sides of the hexagon be of equal length, a regular hexagon will be constructed. This we can do by pointing to one side and pressing the "copy" button, and then to another side and giving the termination flick. The button in this case copies a definition of equal length lines and applies it to the lines indicated. We have said, in effect, make this line equal in length to that line. We indicate that all six lines are equal in length by five such statements. The computer satisfies all existing conditions (if it is possible) whenever we turn on a toggle switch. This done, we have a complete regular hexagon inscribed in a circle. We can erase the entire circle by pointing to any part of it and pressing the "delete" button. The completed hexagon is shown in Figure 1.5F.

To make the hexagonal pattern of Figure 1.1 we wish to attach a large number of hexagons together by their corners, and so we designate the six corners of our hexagon as attachment points by pointing to each and pressing a button. We now file away the basic hexagon and begin work on a fresh "sheet of paper" by changing a switch setting. On the new sheet we assemble, by pressing a button to create each hexagon as a subpicture, six hexagons around a central seventh in approximate position as shown in Figure 1.5G. Subpictures may be positioned, each in its entirety, with the light pen, rotated or scaled with the knobs

and fixed in position by the pen flick termination signal, but their internal shape is fixed. By pointing to the corner of one hexagon, pressing a button, and then pointing to the corner of another hexagon we can fasten those corners together, because these corners have been designated as attachment points. If we attach two corners of each outer hexagon to the appropriate corners of the inner hexagon, the seven are uniquely related, and the computer will reposition them as shown in Figure 1.5H. An entire group of hexagons, once assembled, can be treated as a symbol. The entire group can be called up on another "sheet of paper" as a subpicture and assembled with other groups or with single hexagons to make a very large pattern. Using Figure 1.5H seven times we get the pattern of Figure 1.1. Constructing the pattern of Figure 1.1 takes less than five minutes with the Sketchpad system.

INTERPRETATION OF INTRODUCTORY EXAMPLE

In the introductory example above we have seen how to draw lines and circles and how to move existing parts of the drawing around. We used the light pen both to position parts of the drawing and to point to existing parts. For example, we pointed to the circle to erase it, and while drawing the sixth line, we pointed to the end of the first line drawn to close the hexagon. We also saw in action the very general subpicture, constraint, and definition copying capabilities of the system.

Subpicture:

> The original hexagon might just as well have been anything else: a picture of a transistor, a roller bearing, an airplane wing, a letter, or an entire figure for this report. Any number of different symbols may be drawn, in terms of other simpler symbols if desired, and any symbol may be used as often as desired.

Constraint:

When we asked that the vertices of the hexagon lie on the circle we were making use of a basic relationship between picture parts that is built into the system. Basic relationships (atomic constraints) to make lines vertical, horizontal, parallel, or perpendicular; to make points lie on lines or circles; to make symbols appear upright, vertically above one another or be of equal size; and to relate symbols to other drawing parts such as points and lines have been included in the system. It is so easy to program new constraint types that the set of atomic constraints was expanded from five to the seventeen listed in Appendix A in a period of about two days; specialized constraint types may be added as needed.

Definition Copying:

In the introductory example above we asked that the sides of the hexagon be equal in length by pressing a button while pointing to the side in question. Here we were using the definition copying capability of the system. Had we defined a composite operation such as to make two lines both parallel and equal in length, we could have applied it just as easily. The number of operations which can be defined from the basic constraints applied to various picture parts is almost unlimited. Useful new definitions are drawn regularly; they are as simple as horizontal lines and as complicated as dimension lines complete with arrowheads and a number which indicates the length of the line correctly. The definition copying capability makes using the constraint capability easy.

IMPLICATIONS OF INTRODUCTORY EXAMPLE

As we have seen in the introductory example, drawing with the Sketchpad system is different from drawing with an ordinary pencil and paper. Most important of all, the Sketchpad drawing itself is entirely different from the trail of carbon left on a piece of paper. Information about how the drawing is tied together is stored in the computer as well as the information which gives the drawing its particular appearance. Since the drawing is tied together, it will keep a useful appearance even when parts of it are moved. For example, when we moved the corners of the hexagon onto the circle, the lines next to each corner

were automatically moved so that the closed topology of the hexagon was preserved. Again, since we indicated that the corners of the hexagon were to lie on the circle they remained on the circle throughout our further manipulations.

It is this ability to store information relating the parts of a drawing to each other that makes Sketchpad most useful. For example, the linkage shown in Figure 1.6 was drawn with Sketchpad in just a few minutes. Constraints were applied to the linkage to keep the length of its various members constant. Rotation of the short central link is supposed to move the left end of the dotted line vertically. Since exact information about the properties of the linkage has been stored in Sketchpad, it is possible to observe the motion of the entire linkage when the short central link is rotated. The value of the number in Figure 1.6 was constrained to indicate the length of the dotted line, comparing the actual motion with the vertical line at the right of the linkage. One can observe that for all positions of the linkage the length of the dotted line is constant, demonstrating that this is indeed a straight line linkage. Other examples of moving drawings made with Sketchpad may be found in the final chapter.

As well as storing how the various parts of the drawing are related, Sketchpad stores the structure of the subpicture used. For example, the storage for the hexagonal pattern of Figure 1.1 indicates that this pattern is made of smaller patterns which are in turn made of smaller patterns which are composed of single hexagons. If the master hexagon is changed, the entire appearance of the hexagonal pattern will be changed. The structure of the pattern will, of course, be the same. For example, if we change the basic hexagon into a semicircle, the fish scale pattern shown in Figure 1.7 instantly results.

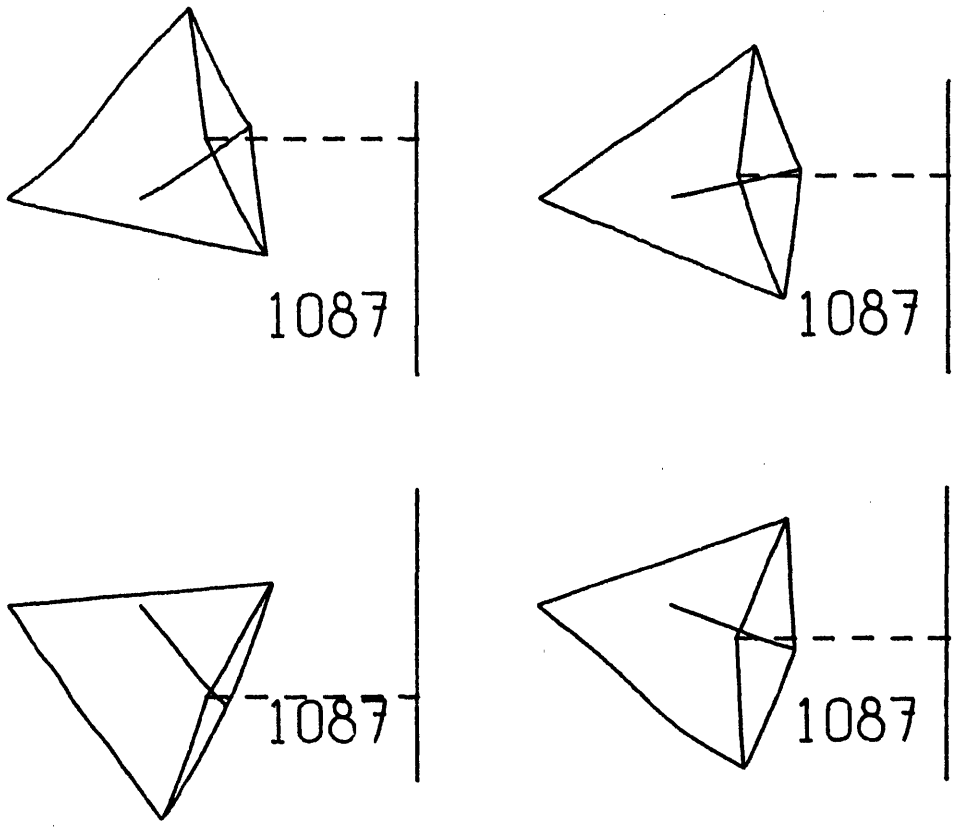


FIGURE 1.6.
FOUR POSITIONS OF LINKAGE
NUMBER SHOWS LENGTH OF DOTTED LINE

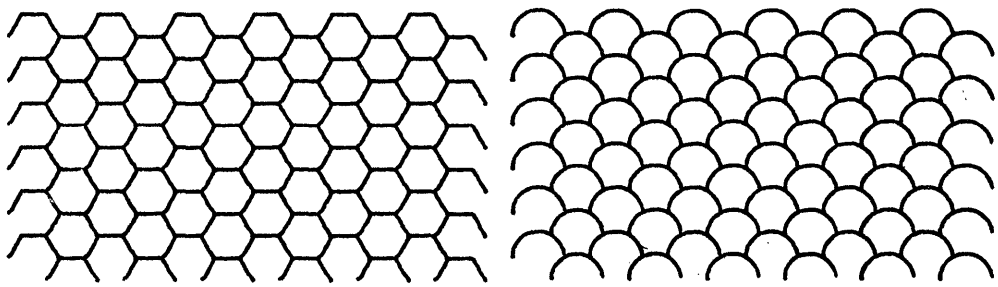


FIGURE 1.7.
⌒ AND ⌒ ON SAME LATTICE

Since Sketchpad stores the structure of a drawing, a Sketchpad drawing explicitly indicates similarity of symbols. In an electrical drawing, for example, all transistor symbols are created from a single master transistor drawing. If some change to the basic transistor symbol is made, this change appears at once in all transistor symbols without further effort. Most important of all, the computer "knows" that a "transistor" is intended at that place in the circuit. It has no need to interpret the collection of lines which we would easily recognize as a transistor symbol. Since Sketchpad stores the topology of the drawing as we saw in closing the hexagon, one indicates both what a circuit looks like and its electrical connections when one draws it with Sketchpad. One can see that the circuit connections are stored because moving a component automatically moves any wiring on that component to maintain the correct connections. Sketchpad circuit drawings will soon be used as inputs for a circuit simulator. Having drawn a circuit one will find out its electrical properties.

SKETCHPAD AND THE DESIGN PROCESS

Construction of a drawing with Sketchpad is itself a model of the design process. The locations of the points and lines of the drawing model the variables of a design, and the goemetric constraints applied to the points and lines of the drawing model the design constraints which limit the values of design variables. The ability of Sketchpad to satisfy the geometric constraints applied to the parts of a drawing models the ability of a good designer to satisfy all the design conditions imposed by the limitations of his materials, cost, etc. In fact, since designers in many fields produce nothing themselves but a drawing of a part, design conditions may well be thought of as applying to the

drawing of a part rather than to the part itself. If such design conditions were added to Sketchpad's vocabulary of constraints the computer could assist a user not only in arriving at a nice looking drawing, but also in arriving at a sound design.

PRESENT USEFULNESS

At the outset of the research no one had ever drawn engineering drawings directly on a computer display with nearly the facility now possible, and consequently no one knew what it would be like. We have now accumulated about a hundred hours of experience actually making drawings with a working system. As is shown in the final chapter, application of computer drawing techniques to a variety of problems has been made. As more and more applications have been made it has become clear that the properties of Sketchpad drawings make them most useful in four broad areas:

For Making Small Changes to Existing Drawings:

> Each time a drawing is made, a description of that drawing is stored in the computer in a form that is readily transferred to magnetic tape. Thus, as time passes, a library of drawings will develop, parts of which may be used in other drawings at only a fraction of the investment of time that was put into the original drawing. Since a drawing stored in the computer may contain explicit representation of design conditions in its constraints, manual change of a critical part will automatically result in appropriate changes to related parts.

For Gaining Scientific or Engineering Understanding of Operations That Can Be Described Graphically:

> The description of a drawing stored in the Sketchpad system is more than a collection of static drawing parts, lines and curves, etc. A drawing in the Sketchpad system may contain explicit statements about the relations between its parts so that as one part is changed the implications of this change become evident throughout the drawing. It is possible, as we saw in Figure 1.6, to give the property of fixed length to lines so as to study mechanical linkages, observing the

path of some parts when others are moved.

As we saw in Figure 1.7 any change made in the definition of a subpicture is at once reflected in the appearance of that subpicture wherever it may occur. By making such changes, understanding of the relationships of complex sets of subpictures can be gained. For example, one can study how a change in the basic element of a crystal structure is reflected throughout the crystal.

As a Topological Input Device for Circuit Simulators, etc.:

Since the ring structure storage of Sketchpad reflects the topology of any circuit or diagram, it can serve as an input for many network or circuit simulating programs. The additional effort required to draw a circuit completely from scratch with the Sketchpad system may well be recompensed if the properties of the circuit are obtainable through simulation of the circuit drawn.

For Highly Repetitive Drawings:

The ability of the computer to reproduce any drawn symbol anywhere at the press of a button, and to recursively include subpictures within subpictures makes it easy to produce drawings which are composed of huge numbers of parts all similar in shape. Great interest in doing this comes from people in such fields as memory development and micro logic where vast numbers of elements are to be generated at once through photographic processes. Master drawings of the repetitive patterns necessary can be easily drawn. Here again, the ability to change the individual element of the repetitive structure and have the change at once brought into all sub-elements makes it possible to change the elements of an array without redrawing the entire array.

Those readers who are primarily interested in the application of Sketchpad are invited to turn next to Chapter IX, page 120 for aditional examples and conclusions.

Chapter II

HISTORY OF SKETCHPAD

When at the end of the summer of 1960 Jack I. Raffel told me that there was considerable interest at Lincoln Laboratory in making a computer "more approachable" through advanced use of displays, I paid little heed, but a seed had been planted. As work on TX-0 computer at MIT during the winter of 1960-61 brought me some familiarity with using display and light pen, the idea began to grow in my mind that application of computers to making line drawings would be exciting and might prove fruitful. Late in April, 1961, following up Mr. Raffel's earlier suggestion, I approached Wesley A. Clark, then in charge of computer applications in Group 51 of Lincoln Laboratory, with the proposal that I use TX-2 in an investigation of computer drawing techniques. I owe a great deal to Mr. Clark's initial enthusiasm and, though I didn't know it at the time, to the many design features[1] he had incorporated into TX-2 seemingly with just such a project in mind.

During the summer of 1960, Herschel H. Loomis had done some preliminary drawing work[6] on TX-2 which he was kind enough to demonstrate for me in May, 1961, as my first contact with TX-2. During the summer of 1961 I devised a curve tracing program and some of the first notions about interlaced and twinkled display. Late in the summer of 1961 a project to connect an ink-line-on-paper plotting system to TX-2 was revived. An EAI plotter,[12] painted bright red, had been at Lincoln Laboratory for two or three years before, but interest in the project had faded for lack of a user. Throughout the Sketchpad effort I have

maintained a collateral interest in the hardware development necessary to get the plotter working. The plotting system has been incorporated as a part of the overall Sketchpad system, but of course its development is only incidental to the research embodied in the thesis.

From the earliest stages of the project development I had had the closest contact with Professor Claude E. Shannon whose penetrating questions have served as the measuring stick by which I could judge my progress. He agreed to supervise the drawing effort as a thesis project in the fall of 1961. In the process of contacting faculty members to form a thesis committee I became aware that my effort was not unique and that I was not alone in my interest and enthusiasm for graphical computer input and output. The availability of computer controlled display systems and particularly of light pen devices for manual input made it almost inevitable that computers would one day be involved in engineering drawing. People at MIT had long talked of such an application.

Computer application to geometric problems was not new. The APT (Automatically Programmed Tool) development through which a computer is able to control a milling machine to produce a complex metal part had evolved many useful geometric manipulation techniques. I made contact with the Computer Aided Design group at MIT which was composed partly of the people of the MIT Electronic Systems Laboratory (formerly called the Servomechanisms Laboratory) who developed APT and partly of people in the Mechanical Engineering Department who brought a knowledge of the problems faced by designers to the project.

I had been surprised that so little practical work had been done in application of computers to line drawing, especially since display

systems and light pens were relatively common when my work began. I can see now, however, that I have had a unique opportunity to pursue my interest. I was able to visit many separate laboratories for discussion and ideas without becoming so attached to any one that I was forced into its way of thinking. In particular, members of the Mechanical Engineering Department, notably Professor Steven A. Coons, who agreed to serve on my thesis committee, suggested mechanical design problems and applications. Members of the Electronic Systems Laboratory, notably Douglas Ross, provided help and advice on n-component elements. The Artificial Intelligence group, notably Professor Marvin Minsky, another committee member, gave advice and encouragement in the niceties of picture representation and the kind of interest aimed more at a fundamental understanding of the processes developed than in their practical application. Lincoln Laboratory provided not only advice but also technical support including to date about 600 hours of time on the TX-2.

Whatever success the Sketchpad effort has had can in no small measure be traced to the use of TX-2.* TX-2's 70,000 word memory, 64 index registers, flexible input-output control and liberal supply of manual intervention facilities such as toggle switches, shaft encoder knobs, and push buttons all contributed to the speed with which ideas could be tried and accepted or rejected. Moreover, being an experimental machine it was possible to make minor modifications to TX-2 to match it better to the problem. For example, a push button register was installed at my request. Now that we know what drawing on a computer is like, much smaller machines can be used for practical applications.

---

* A brief description of TX-2 may be found in Appendix G.

RESEARCH PROGRESS

Thus it was that in the fall of 1961 work began in earnest on a drawing system for TX-2. In the early fall I perfected my light pen tracking programs and subroutines for displaying straight lines and presenting a portion of the total picture on the display at increased magnification. In early November 1961, my first light pen controlled drawing program was working. It is significant that at this time a notion of "strong conditions" was used to give geometric nicety to the drawing. For example, lines could be drawn parallel or perpendicular to existing lines but carried no permanent trace of the relationship other than the accident of their position. This early effort in effect provided the T-square and triangle capabilities of conventional drafting. Somewhat before my first effort was working, Welden Clark of Bolt, Beranek, and Newman demonstrated a similar program to me on the PDP-1 computer.[5]

Early in December 1961 Professor Shannon visited TX-2 to see the work I had been doing. As a result of that visit the entire effort took new form. First, Professor Shannon suggested point blank that I include circle capability in the system. Second, I realized when he asked for paper to sketch a drawing he intended to enter into the computer that the strong conditions notion which simulated the conventional tools of drafting was not adequate for computer drawing. As a result of including circles into the Sketchpad system a richness of display experience has been obtained without which the research might have been rather dry. As a result of trying to improve upon conventional drafting tools the full new capability of the computer-aided drafting system has come into being.

In December 1961, and during the first part of 1962, then, I began working on the problems of display generation for cirlces outlined in Chapter V. The circle generating subroutine gave great difficulty especially in the details of edge detection and closure. At about this same time I started work on the ring structure representation of the drawing outlined in Chapter III; the preliminary drawing effort of November 1961 had used conventional table storage. By the first of February 1962, the ring structure was in use, but without the generic blocks which give it its present flexibility. Intersection programs for lines and circles had been written and debugged, and the second generation drawing program could be begun.

In making the second generation drawing program, explicit representation of constraints and automatic constraint satisfaction were to be included. I learned of the matrix method described in Appendix F for finding the minimum mean square error solution to linear equations from Lester D. Earnest of the MITRE Corporation and obtained a macro, SOLVE, from Lawrence G. Roberts which did the arithmetic involved.[8] Armed with the tools for representing and doing arithmetic for constraints I went gaily ahead with programming.

In the first crack at representing constraints I made two basic errors which have subsequently been corrected. First, I provided that the constraints be tied not only to the variables constrained but also to related nonvariables. For example, the horizontal constraint not only referred (as it should) to the two end points of a line, but also (as has been subsequently removed) to the line itself. It was impossible to make points have the same y coordinate without having a line

between them; deletion of the line deleted the constraint as well. In more recent work constraints refer only to variables so that lines need not be present to make points have the same y coordinate.

The second error in constraint representation was in the numerical computation of the relationship represented by the constraint. At first I insisted that for any constrained variable it be possible to compute directly the linear equation of best fit to the constraint. That is, for each constraint on a variable the equation of a line could be found along which the constraint would be satisfied or nearly so. Not only was it difficult to compute the equation of such a line, requiring a special purpose program for each type of constraint, but also my lack of regard for the niceties of the scale factor of the computed equation resulted in instabilities in the constraint satisfaction process. Whereas for the relaxation procedure to operate properly it is necessary to remove "energy" from the system at each stage, my computations for certain cases added energy. It was early summer of 1962 before definition of the mathematical properties of constraint types in terms of a subroutine for computing directly the error introduced by a constraint not only cured the instability troubles but also made it easy to add new constraint types.

Along with the new capabilities of the constraint satisfaction programs and the extensive use to be made of constraints, the second generation drawing program included for the first time the recursive instance expansion which made possible instances within instances. The trials of getting systems to work are many; one which stands out in my mind was that instances within instances rotated in the wrong direction when the

outer instance was rotated. Neither were the things I tried to do always correct. For example, the initial instance expansion routine forced each instance of a picture to be smaller than the master drawing for that instance. I have since come to appreciate the value of having some normalizing factor in products so that all fixed point numbers can be treated as signed fractions in the range, $-1 \geqslant x \geqslant 1$, representing the fraction of full scale deflection on the coordinate system in question.

In late March 1962, I discovered that points could be related to instances through the use of two linear equations relating the coordinates of the point to the four components of the instance position. Armed with this new information, the difficulties I had been having with attachers on instances yielded to the same general format used for other constraints. It became possible for a single instance to have as many attachers as desired, each of which could serve as attachment point for any number of instances.

The first actual programming of the maze-solving high-speed constraint satisfaction methods proposed much earlier began about March 1962. I had not had enough experience before that time with the ring structure to face the extensive ring manipulations which would be required for this part of the work. The development of the ring manipulation macros shown in Appendix D was started in connection with the maze solving routines.

By Memorial Day 1962, the second version of Sketchpad was considered working well enough that a motion picture was made showing the various drawing manipulations possible. It was possible to draw line

segments and circle arcs and point to them to erase them or move the points on which they depend. A limited number of constraints were available which could make lines horizontal or vertical, force points to lie on lines or circles, and relate instances to their attachment points. Constraint satisfaction was primarily by relaxation, but for certain simple cases the maze solving methods would give more rapid results. It was possible to see that sketching could indeed be done on the computer.

Not yet available were display for points or constraints, or any notion of digits, text, scalars and dummy variables. It was almost impossible to add new constraint types to the system, and even had they been added, the recursive merging and the definition copying capability were not available to apply them easily to the object picture. Sketchpad at this stage was a nice demonstration and toy but as yet lacked the richness of detail now available.

During the late spring of 1962, then, enough experience had been gained with computer drawings to realize that more capabilities were needed. It was possible for me, armed with photographs of the latest developments, to approach a great many people in an effort to get new ideas to carry the work on to a successful conclusion. Out of these discussions came the notions of copying definitions and of recursive merging which are, to me, the most important contributions of the Sketchpad system. Also out of these talks came the conviction that a generic structure would be necessary if the system were to be made easy to expand. On June 9, 1962 all this new information came to a head and an entirely new system was begun which has grown with relatively little

change into the final version described here. Had I the work to do again, I could start afresh with the sure knowledge that generic structure, separation of subroutines into general purpose ones applying to all types of picture parts and ones specific to particular types of picture parts, and unlimited applicability of functions (e.g. anything should be moveable) would more than recompense the effort involved in achieving them. I have great admiration for those people who were able to tell me these things all along, but I, personally, had to follow the stumbling trail described in this chapter to become convinced myself. It is to be hoped that future workers can either grasp the power of generality at once and strive for it or have the courage to stumble along a trail like mine until they achieve it.

Towards the end of the summer of 1962 the third and final version of Sketchpad was beginning to show remarkable power. I had the good fortune at this time to obtain the services of Leonard M. Hantman, a Lincoln Laboratory Staff Programmer, who added innumerable service functions, such as magnetic tape manipulation routines, to the system. He also cleaned up some of the messy programming left over from my rushed efforts at getting things working. For example, he shortened and improved my original ring manipulation macros. Also, towards the end of the summer the plotting system began to be able to give useable output. Hantman added plotting programs to Sketchpad through which the figures in this paper were made.

Computer time began to be spent less and less on program debugging and more and more on applications of the system. It was possible to provide preliminary services to other people, and so a user group was

formed and informal instruction was given in the use of Sketchpad. A library tape was obtained and has ever since been collecting pictures for possible future use. The user group experience showed that relatively new users with no programming knowledge could produce simple drawings with the system if a skilled user (myself) prepared the building blocks necessary. For example, a secretary designed and drew an alphabet with the aid of a 10 x 10 raster of points to use as end points. Both the raster and the alphabet are now a part of the library.

Even now, however, there are possibilities for application of the system not yet even dreamed of. The richness of the possibilities of the definition copying function, and the new types of constraints which might easily be added to the system for special purposes suggest that further application will bring about a new body of knowledge of system application. For example, the bridge design examples shown at the end of this paper were not anticipated.

There are, of course, limitations to the system. In the last chapter are suggested the improvements, some just minor changes, but some major additions which would change the entire character of the system. It is to be hoped that future work will far surpass my effort.

Chapter III

RING STRUCTURE

The Sketchpad system stores information about drawings in two separate forms. One is a table of display spot coordinates designed to make display as rapid as possible; the other is a file designed to contain the topology of the drawing. The topological file is set up in a specially designed ring structure which will form the major subject of this chapter. The ring structure was designed to permit rearrangement of the data storage structure for editing pictures with a minimum of file searching, and to permit rapid constraint satisfaction and display file generation. The ring structure was not intended to pack the required information into the smallest possible storage space. We felt that we could write faster running programs in less time by including some redundancy in the ring structure. This was considered more important than the ability to store huge drawings. Moreover, the large storage capacity of the TX-2 did not force storage conservation. The particular form of the ring structure chosen has led to some of the most interesting features of the system simply because the changes required to keep the ring structure consistent led to useful facilities such as recursive merging discussed in Chapter VI.

N-COMPONENT ELEMENTS

In the drawings made by the Sketchpad system there are large populations of relatively few types of entities with very little variation in format between entities of each type. For example, an entire

drawing may be composed of line segments and end points, each line segment connecting exactly two end points, and each point having exactly two coordinates. Because of this uniformity within each given type of entity, it is possible to set up a standard storage format for each type of entity with standardized locations for information about the various properties which entities of that particular type usually have. Each entity, therefore, is represented in the computer as an n-component element, that is, by a block of n consecutive registers in storage each of which contains a specific kind of information about that element. For example, the coordinates of a point are always stored in the i^{th} and j^{th} registers of its n-component element or block. Similarly, the n^{th} and m^{th} registers of a line block always contain the addresses of the start and end point blocks for that line as Figure 3.1 shows. Particular numerical locations for various pieces of information are shown in Appendix C.

MNEMONICS AND CONVENTIONS

In using n-component elements it has been found useful to give symbolic names to the various registers of each element so that the actual numerical locations of various kinds of information need not be remembered. Thus, for example, the coordinates of a point are stored in the $PVAL^{th}$ and $PVAL+1^{st}$ (for Point VALue) registers of its n-component element. Since all programming for Sketchpad is done in a symbolic programming language in terms of mnemonics, it is easy to rearrange the internal format of any kind of n-component element by changing the numerical values assigned to the mnemonic symbols used within that kind of element. In the figures in this thesis, symbolic locations of

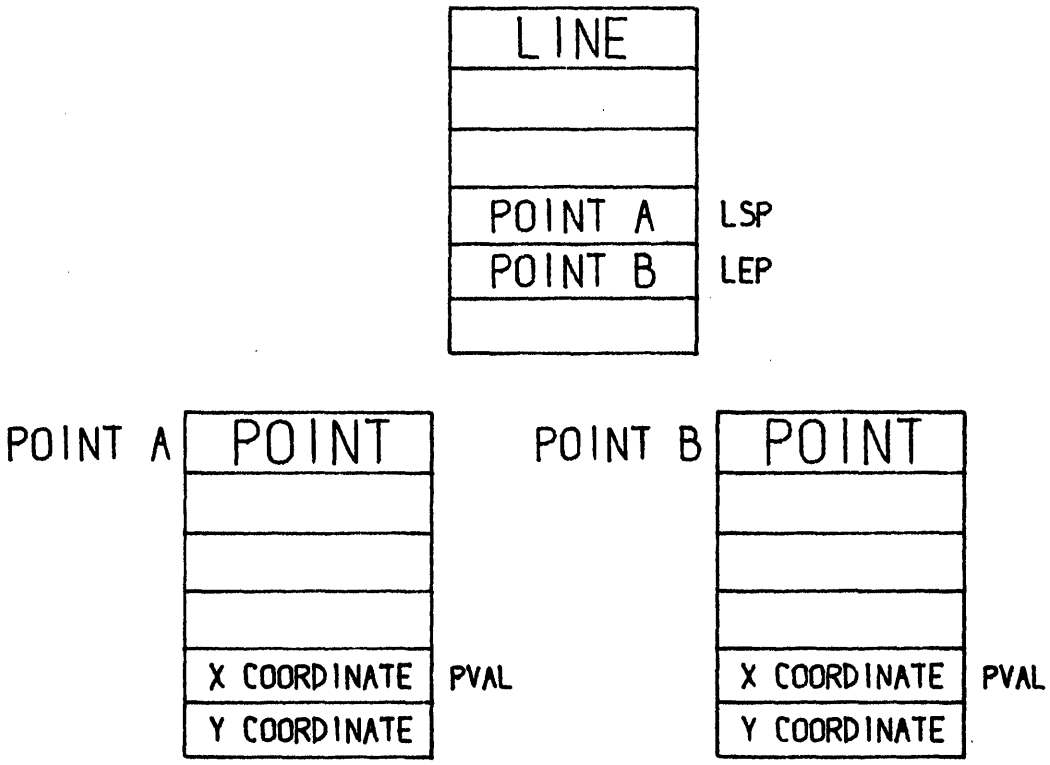


FIGURE 3.1. N-COMPONENT ELEMENTS

pieces of data within n-component elements are shown to the right of the data. Actual register addresses are shown to the left of the data. The position of particular pieces of data may change from figure to figure as it becomes necessary to more fully illustrate the structure, but the mnemonic address will indicate which data are being shown.

Although the use of mnemonics gives complete flexibility to the format of n-component elements, certain conventions were followed in implementing Sketchpad and in the figures of this thesis.

1. The location of an n-component element is the address of its first (lowest numbered) register;

2. The first component of the element (the contents of its first register) is used to indicate the type of element; and

3. All numerical information such as values of coordinates is located at the end (highest numbered locations) of the element.

In the figures, higher numbered registers run down the page, making the location of an element the address of its top register. Such element locations are indicated by symbolic names to the left of the n-component element or contained within components of other elements which make reference to them.

Most of the components of the n-component elements in the Sketchpad system are pointers containing addresses of other elements. Such pointers indicate topological information such as the end points of a line segment. If an n-component element is to be relocated in storage, that is, if the information it contained is to be stored in some other registers to compact the storage structure prior to saving it on magnetic tape, the contents of any topological component referring to the element which is to be relocated must be changed to refer to the new location.

However, relocation of an element in storage should not change the appearance of the picture represented, and so numerical information such as the coordinates of points or the size of subpictures must not be changed. Segregation of numerical information at the end of the n-component element facilitates the relocation of elements.

Gross transfers of the entire storage structure can be accomplished by treating all topological pointers as relative to some basic address. In Sketchpad, for example, a topological pointer to an n-component element contains not the absolute computer address of that element, but the location of the n-component element relative to the first address of the storage structure area, LIST. At various times it has been necessary to change the location of the storage area, giving LIST a different value. The use of relative pointers proves useful for inter-machine communication also, making it possible to store a given data structure anywhere in memory. In the illustrations, however, the relative pointing is suppressed, as if LIST = 0.

REVERSE INDEXING

Suppose that index register α contains the relative location of the n-component element for a line segment and that it is desired to know the coordinates of that line's start point (LSP). The address of the start point block may be found in the LSP^{th} entry of the line block as shown in Figure 3.1. We can pick up this address using reverse indexing by the instruction:

$$LDA_{\alpha}\ LSP + LIST$$

load the accumulator from the LSP^{th} entry of the block pointed to by

index register α. LIST enters in because pointers are relative. Now if we transfer the contents of the accumulator to index register β and perform the instruction:

$$\text{LDA}_{\beta}\ \text{PVAL} + \text{LIST}$$

the X coordinate of the start point of the line will be placed in the accumulator.

Note that in these instructions we used the index register to indicate which n-component element is being considered and the address portion of the instructions to indicate the specific component selected. This is called "reverse indexing" to distinguish it from "normal" indexing in which the index register indicates the i^{th} entry of the table referred to in the address portion of the instruction. The only normal thing about "normal" indexing, however, is the widespread inclusion in computers of an instruction which increments an index register and transfers control to a specified location if the index register has not yet reached some specified value, usually 0. The 709's TIX instruction is typical.

A real value of the TX-2 for implementing the Sketchpad system turned out to be its ability to reset an index register from a register indicated by the contents of another index register (or even the prior contents of the index register to be reset!). TX-2's accumulator is not used in this index register processing. A special symbolism was built into the compiler to make it easy to use double index instructions; the instruction:

$$\text{RSX}_{\beta|\alpha}\ \text{LSP} + \text{LIST}$$

puts into β the address of the start point of the line pointed to by

index register α. The Sketchpad program consists in large part of such instructions.

RING STRUCTURE

The basic n-component element structure described above has been somewhat expanded in the implementation of Sketchpad so that all references made to a particular n-component element or block are collected together by a string of pointers which originates within that block. For example, all the line segments which terminate on a particular point may be found by following a string of pointers which starts within the point block. This string of pointers closes on itself; the last pointer points back to the first. Moreover, the string points both ways to make it easy to find both the next and the previous member of the string in case some change must be made to them.

The ring structure, then, assigns two registers to each component in the n-component element. One is used for the direct reference shown in Figure 3.1; the other register is used to string similar references together. The basic ring consists of two kinds of register pairs, the "hen" and "chicken." The hen pair is contained within a block which will be referred to, for example, in a point block, while the chicken pair is contained in a block making reference to another, for example, a line block making reference to the point. The chickens which belong to a particular hen constitute <u>all</u> the references made to the block containing the hen. Figure 3.2 shows a typical ring; the inserting operation and ordering shown will be explained below. Appendix C shows how the hen and chicken blocks are arranged in different kinds of elements.

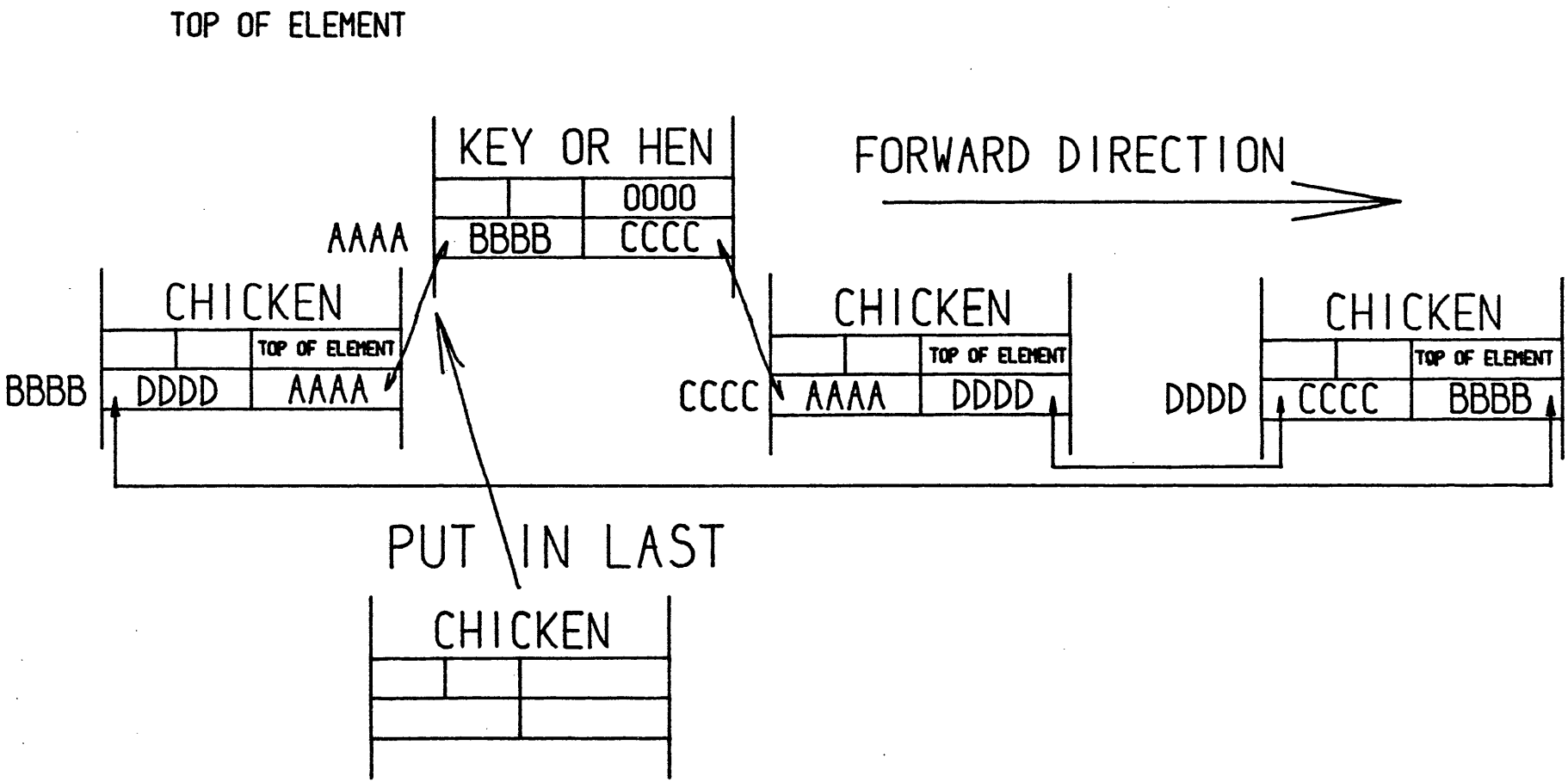


FIGURE 3.2. BASIC RING STRUCTURE

Figure 3.3 shows the complete structure for a line segment and two end points with the appropriate rings shown.

The mnemonic for a component is taken to be the upper (lower numbered) of the register pair. The ring collecting ties, of course, are relative to LIST but this has been suppressed in the illustrations. The part of the upper register not occupied by the chicken pointer contains a number which indicates how far this particular element is from the top of the n-component element. This is the small negative number showing in Figure 3.3. It is used to find the top of a block when a component of it has been found as a member of a ring.

HUMAN REPRESENTATION OF RING STRUCTURE

In representing ring structures the chickens should be thought of as beside the hens, and perhaps slightly below them, but not directly below them. The reason for this is that in the ring registers, regardless of whether in a hen or a chicken, the left half of one register points to another register whose right half always points back. By placing all such registers in a row, this feature is clearly displayed. Moreover, the meaning of placing a new chicken "to the left of" an existing chicken or the hen is absolutely clear. The convention of going "forward" around a ring by progressing to the right in such a representation is clear, as is the fact that putting in new chickens to the left of the hen puts them "last," as shown in Figure 3.2. Until this representation was settled on, no end of confusion prevailed because there was no adequate understanding of "first," "last," "forward," "left of," or "before."

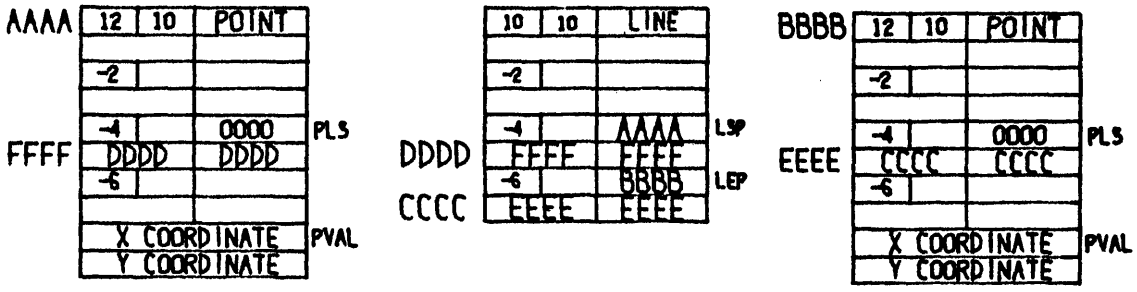
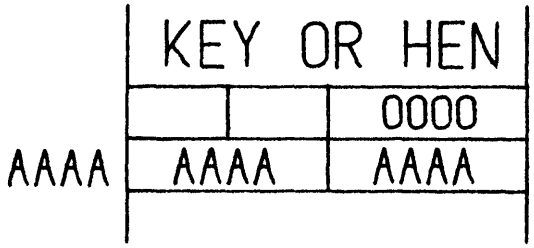


FIGURE 3.3.
LINE SEGMENT AND END POINTS
IN RING STRUCTURE NOTATION



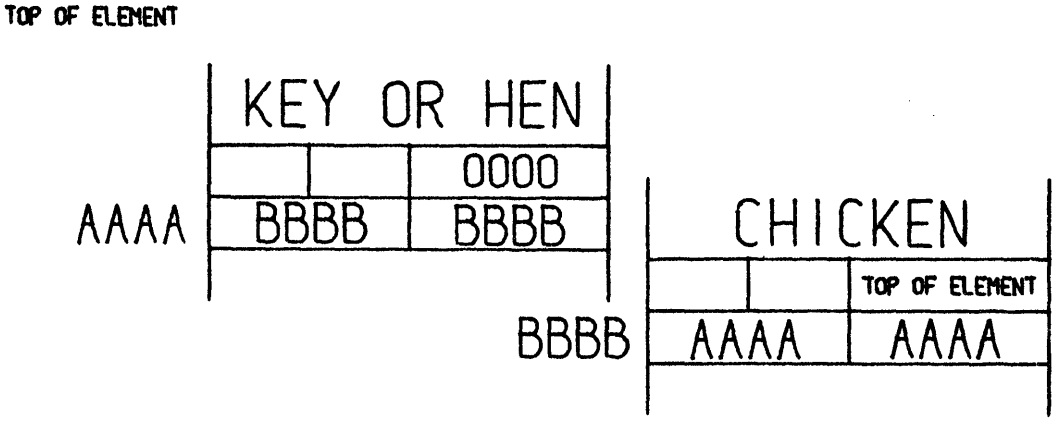


FIGURE 3.4.
ZERO AND ONE MEMBER RINGS

BASIC OPERATIONS

The basic ring structure operations are:

1. Inserting a new chicken into a ring at some specified location in it, usually first or last.

2. Removing a chicken from a ring.

3. Putting all the chickens of one ring, in order, into another at some specified location in it, usually first or last.

4. Performing some auxiliary operation on each member of a ring in either forward or reverse order.

These basic ring structure operations are implemented by short sections of program defined as MACRO instructions in the compiler language. By suitable treatment of zero and one member rings, that is of hens with none or one chicken, as shown in Figure 3.4, the basic operation programs operate without making special cases. As stated in the macro language, the basic operations become trivially easy to use. For example,

$$\text{PUTL} \equiv \text{LSP} \times \alpha \rightarrow \text{PLS} \times \beta$$

puts the LSP (Line Start Point) entry of the line block pointed to by index register α into the ring whose hen is the PLS (Point LineS entry) of the point indicated by index register β, thus making β be the start point of α. If "x" is read as "of" and "→" is read as "into", the macro statement almost makes sense in English. The format and function of all the ring manipulation macro instructions used in Sketchpad can be found in Appendix D.

GENERATION OF NEW ELEMENTS

Subroutines are used for setting up new n-component elements in free spaces in the storage structure. These subroutines place the

distance-to-the-top numbers in alternate registers as required and clear out the components so that each is an empty ring as shown in Figure 3.5. As parts of the drawing are deleted, the registers which were used to represent them become free, indicated by placing them in the FREES ring. Data for new n-component elements could be put into these free registers if sufficiently long continuous blocks of free storage were available, but Sketchpad is not at present equipped to do this. Rather, new components are set up at the end of the storage area, lengthening it, while free blocks are allowed to accumulate. Garbage collection periodically compacts the storage structure by removal of the free blocks and relocation of the information above them (that is, information in higher numbered registers illustrated lower on the page) as shown in Figure 3.6. Storage of a drawing on magnetic tape can be done much more compactly for having removed all internal free registers.

BOOBY TRAPS

Every system which is devised for programming on computers has little problem areas which give humans more trouble than other parts; the ring structure organization and operations are no exception. As was indicated above, the visualization of the ring as a row of elements aids greatly in understanding of the basic operations. The use of relative addressing, while giving great power for data communication, gave the programmer considerable difficulty because the term LIST must often but not always be added to or subtracted from the address portion of instructions. It took months before all the nuances of these problems were learned.

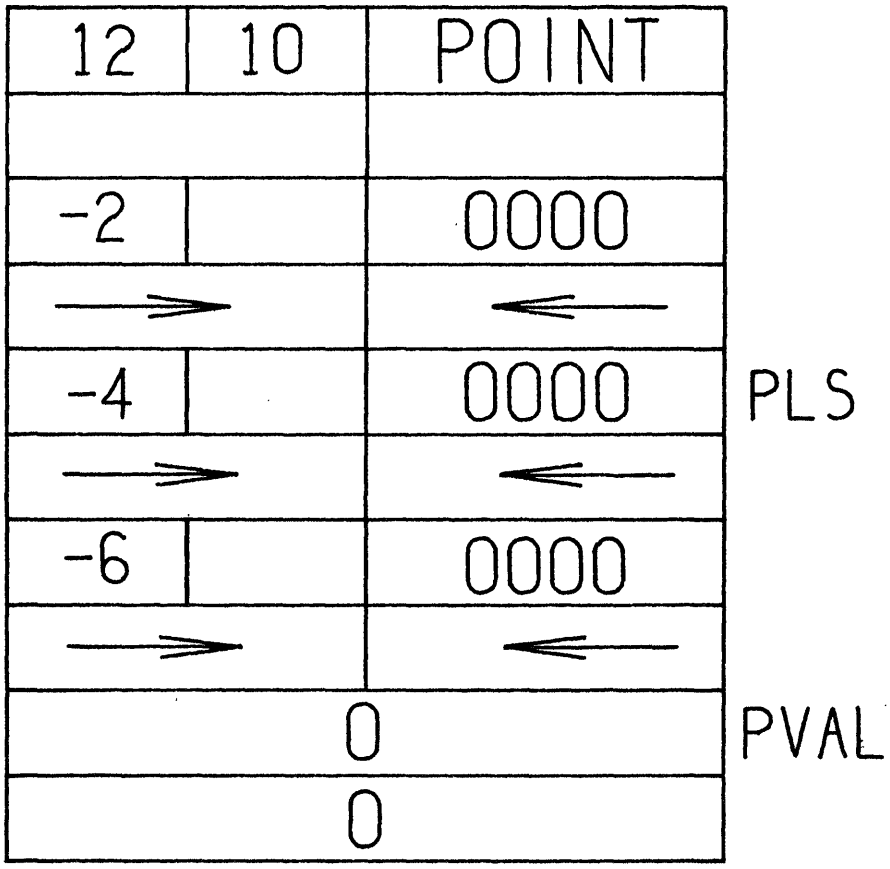


FIGURE 3.5. FRESH POINT BLOCK

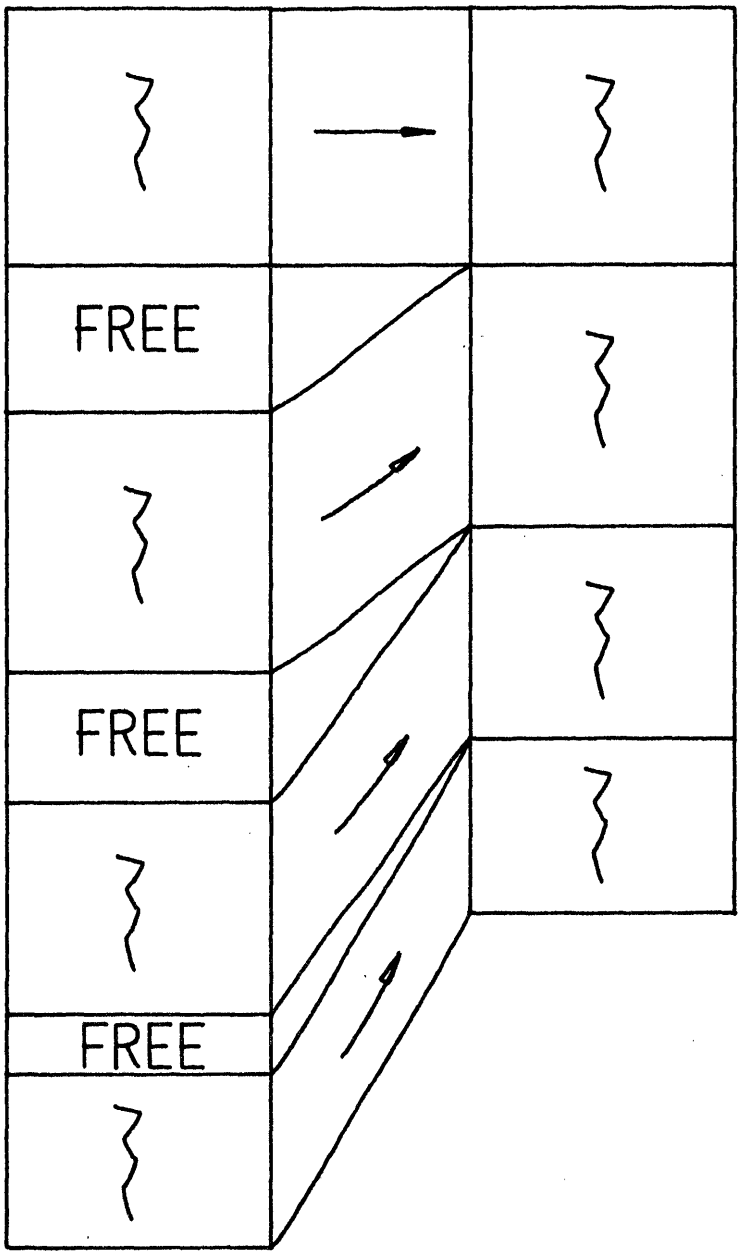


FIGURE 3.6.
COMPACTING THE RING STRUCTURE

By far the greatest difficulty concerned processes which change the ring structure while other operations are taking place on it. For example, there must be two versions of the basic macro which permits auxiliary operations to be performed on all the members of a ring in turn. One version, LGORR (Leonard's GO Round the ring to the Right), performs the auxiliary operation on one ring member while remembering the next ring member so that if the auxiliary operation deletes the current ring member the next one has already been found. Another version of the basic macro, LGORRI (LGORR Insertable), remembers which ring member the auxiliary operation is being performed on so that if the auxiliary operation puts a brand new member into the ring next to the current one, the new one will not be overlooked. Neither macro will function properly if both the current and the next ring members are deleted simultaneously by the auxiliary function.

Early in the research the multiple sequence nature of the TX-2 was utilized to provide immediate updating of the ring structure when push button commands were given by the user. Trouble arose if the display generation program was working in the ring structure at the instant that it changed. It is now clear that multiple sequencing and data channels must be used only to transmit information into the computer and not to process the ring structure, a job properly left to the main computation stream. Main computation stream ring manipulation has implications on future machine design since most of the ring manipulations can be performed with index arithmetic alone without tying up the main arithmetic element which meanwhile could be of use to someone else. Perhaps several machines could share a single powerful arithmetic element if they did the bulk of their processing with index arithmetic.

GENERIC STRUCTURE, HIERARCHIES

The organization of the elements of the drawing into types has facilitated the generalization of the programs which comprise the Sketchpad system. The effort toward generality came relatively late in the research effort because I did not at first appreciate the power that a general approach could bring. Considerable reprogramming was done, however, to include as much generality as possible. Those subroutines which had to do with a single kind of drawing part were collected together and specifically labeled, both in the coding sheets and block diagrams, but most importantly in the mind, as belonging to that particular kind of entity. The remainder of the program was left completely general.

The general part of the program will perform a few basic operations on any drawing part, calling for help from routines specific to particular types of parts when that is necessary. For example the general program can show any part on the display system by calling the appropriate display subroutine. Similarly, the general program is able to relocate objects on the display, making use of specific routines only to apply a transformation to the various kinds of objects. Again, the general program will satisfy any numerical constraints applied to the drawing by the user, calling on specific subroutines only to compute the error introduced into the system by a particular constraint.

The big power of the clear-cut separation of the general and the specific is that it is easy to change the details of specific parts of the program to get quite different results or to expand the system without any need to change the general parts. This was most dramatically

brought out when generality was finally achieved in the constraint display and satisfaction routines and new types of constraints were constructed literally at fifteen minute intervals.

In the data storage structure the separation of general and specific is accomplished by collecting all things of one type together as chickens which belong to a "generic" hen. The generic hen contains all the information which makes this type of thing different from all other types of things. Thus the data storage structure itself contains all the specific information, leaving only general programs for the rest of the system. A typical generic block is shown in Figure 3.7.

The generic blocks are further gathered together under super-generic or generic-generic blocks according to four categories: Variables, Topologicals, Constraints, and Holders, as shown in Figure 3.8. All picture parts which have numerical information are ultimately gathered together under the VARIABLES block by way of their own generic blocks. Ideally the VARIABLES block should in some way indicate that there was numerical information, but the generality has not been carried as far as this yet. Space for information about the number of components of a variable (which is unnecessary for the topological entities) could be omitted from the generic blocks for lines and circles. At present all generic blocks still carry space for all the information in any of them simply because of historical reasons. This accounts for the spaces seen in the Figure 3.7.

For the sake of completeness the four broad categories of things, the generic-generic blocks, are brought together under the UNIVERSE block, which, as a special case, is always located at the exact start of

24	4	VARIABLES	TYPE
-2		0000	SPECB
TYPEWRITER CODE NAME			NAME
SUBROUTINE ENTRY			DISPLAY
FIT SCOPE AROUND IT			HOWBIG
APPLY TRANSFORMATION			MOVIT
24.16..			SIZE
NORMAL PICTURE KIND			KIND
FOUR COMPONENTS			TUPLE
VALUE AT IVAL			VARLOC

FIGURE 3.7.
INSTANCES GENERIC BLOCK

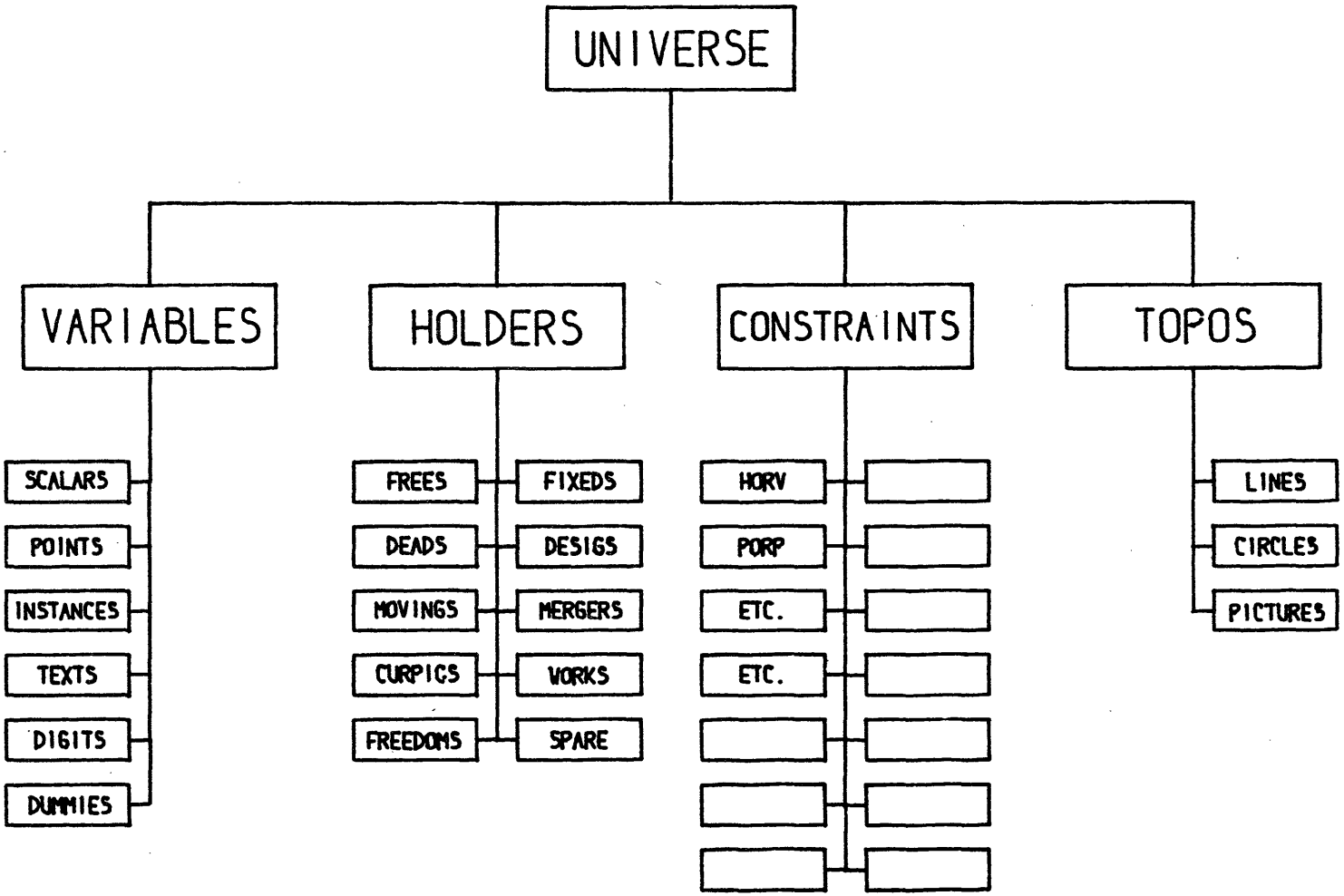


FIGURE 3.8. GENERIC STRUCTURE

the storage structure, relative address 1. The UNIVERSE block belongs to no higher block. I considered making it belong to itself so that continued upward searching through the generic structure would appear to reach an unending string of UNIVERSE blocks, but I could find no solid reason for so doing. Further work may develop one, of course.

EXPANDING SKETCHPAD

Addition of new types of things to the Sketchpad system's vocabulary of picture parts requires only the construction of a new generic block (about 20 registers) and the writing of appropriate subroutines for that thing. The subroutines might be easy to write, as they usually are for new constraints, or difficult to write, as for adding ellipse capability, but at least a finite, well-defined task faces one to add a new ability to the system. Before the generic structure was clarified, it was almost impossible to add the instructions required to handle a new type of element.

Chapter IV

LIGHT PEN

In Sketchpad the light pen is time shared between the functions of coordinate input for positioning picture parts on the drawing and demonstrative input for pointing to existing picture parts to make changes. Although almost any kind of coordinate input device could be used instead of the light pen for positioning, the demonstrative input uses the light pen optics as a sort of analog computer to remove from consideration all but a very few picture parts which happen to fall within its field of view, saving considerable program time. Drawing systems using storage display devices of the Memotron type may not be practical because of the loss of this analog computation feature.

CONSTRUCTION OF LIGHT PEN

The light pen is a hand held photocell which will report to the computer whenever a spot on the display system falls within its small field of view. The housing for the photocell is about the size of a fountain pen and is manipulated much as a pen or pencil, hence the name. Many different varieties of light pens have been built, including large cumbersome ones in the days before miniaturization, to be replaced by transistorized versions, and recently by fiber optic pens connected by a flexible light pipe to a photocell mounted inside the computer frame. The particular pen used for the Sketchpad system consists of a photodiode and transistor preamplifier mounted in the pen housing and connected to the computer by a length of small coaxial cable, as shown in

the photograph of Figure 4.1, and in the drawing of Figure 4.2. It is used by Sketchpad primarily because its operation is relatively independent of the distance it is held from the computer display, since it has a cylindrical field of view.

Since spots on the display system are intensified one after another in time sequence, whether or not each spot is seen by the pen is individually reported just after intensification of that spot. The light pen amplifier is designed so that the pen is sensitive only to the bright blue flash of the first intensification of a display spot and not to the dim yellow afterglow. The amplifier output is strobed only when a display spot has been intensified to minimize room light pickup. Although some computers require an interrogation of a pen flip-flop to find out if a spot was seen, TX-2 uses the interruption of a sequence change to indicate this fact*. Thus if a series of points are displayed on the scope by a set of data transfer instructions, and one of these points falls under the field of view of the pen, subsequent instructions will be performed in the light pen sequence rather than in the display sequence until the light pen sequence is finished. Thus it is unnecessary to interrogate the pen specifically for each display spot, the interruption of sequence changing serving automatic notification that a spot was seen. For pen tracking, where a program branch is made for every spot displayed, interruption by the pen requires more program instructions than would a specific bit testing instruction, but for the demonstrative use of the pen where any spot of the background display may fall within the pen's field of view but is relatively unlikely to do so, the interruption is a real advantage.

---

* TX-2's light pen is treated as an input device separate from its display. See Appendix G.

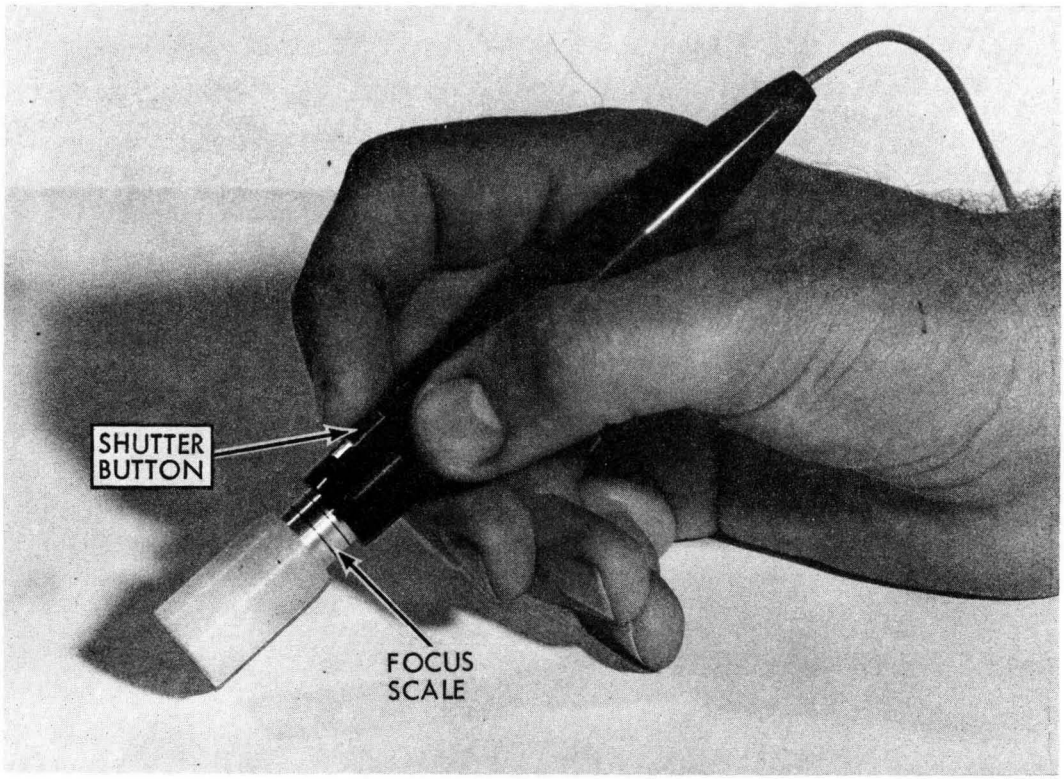


FIGURE 4.1. LIGHT PEN.
Courtesy of MIT
Electronic Systems Laboratory.

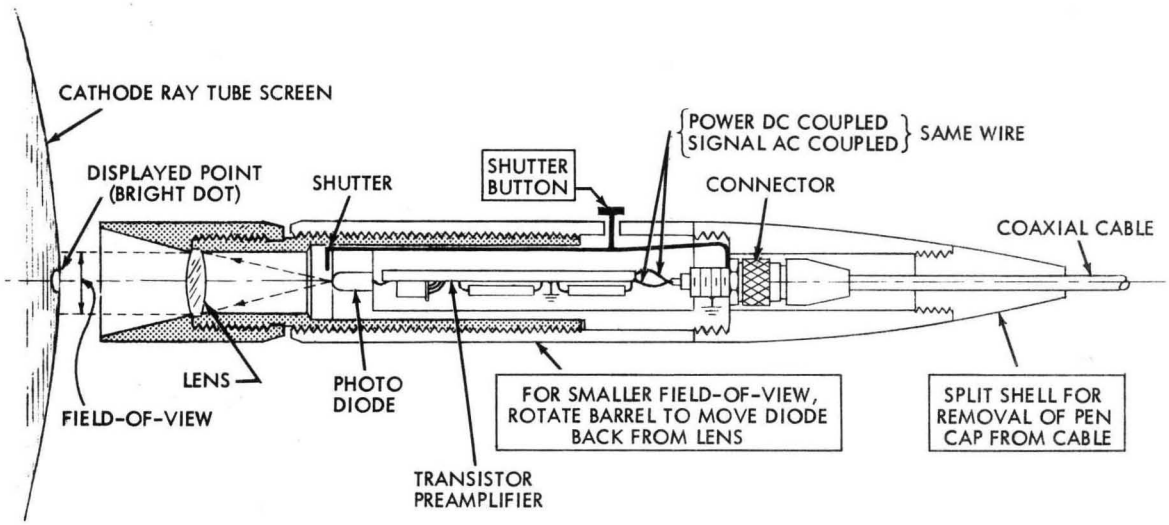


FIGURE 4.2. CONSTRUCTION OF LIGHT PEN.
Drawing courtesy of Electronic Systems Laboratory. This drawing was made by conventional methods.

PEN TRACKING

The light pen and its connecting cable report to the computer immediately after any display spot has been shown which lies within the pen's view. By displaying a cross-like pattern and noticing which spots fall within the light pen's view, the computer can follow the motions of the light pen around the screen. In order to follow normal motions of a hand held light pen I have found it necessary to redisplay the tracking cross about 100 times per second, taking 1 millisecond per display. When the cross is being "dragged" across the screen at the maximum speed I have achieved, successive crosses are displayed about 0.2 inches apart and the maximum pen speed is thus 20 inches per second which has proven quite enough for the experiments conducted. If the light pen is moved faster than that, the tracking cross will fall entirely outside of its field of view and tracking will be lost. I use the loss of tracking as the so-called termination signal for all pen tracking operations.

Early in the system development some effort was spent trying to reduce the computer time spent in pen tracking. It was attempted to have the computer predict the location of the pen based on its past locations so that a longer time might elapse between display of tracking crosses. The assumptions of constant velocity,

$$X_t = (X_{t-1} - X_{t-2}) + X_{t-1} \qquad Y_t = (Y_{t-1} - Y_{t-2}) + Y_{t-1}, \qquad (4\text{-}1)$$

and constant acceleration,

$$X_t = 3(X_{t-1} - X_{t-2}) + X_{t-3} \qquad Y_t = 3(Y_{t-1} - Y_{t-2}) + Y_{t-3}, \qquad (4\text{-}2)$$

where successive pen positions are denoted by subscripts, were tried. A pictorial representation of these assumptions is shown in Figure 4.3.

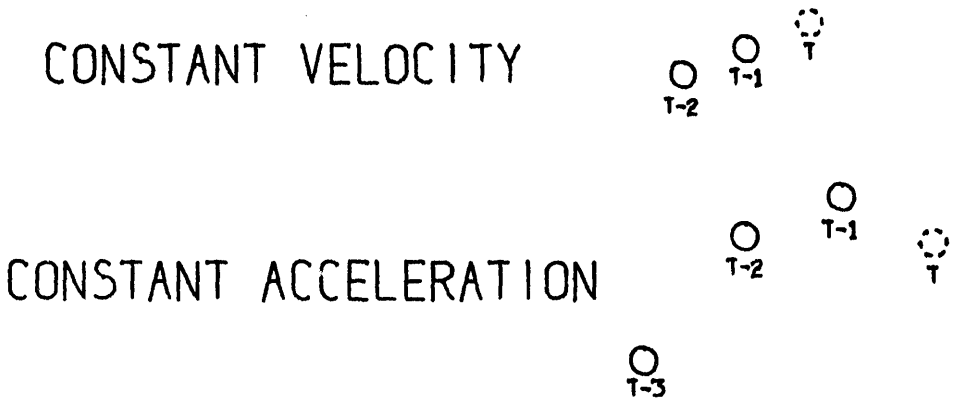


FIGURE 4.3.
PREDICTIVE PEN TRACKING

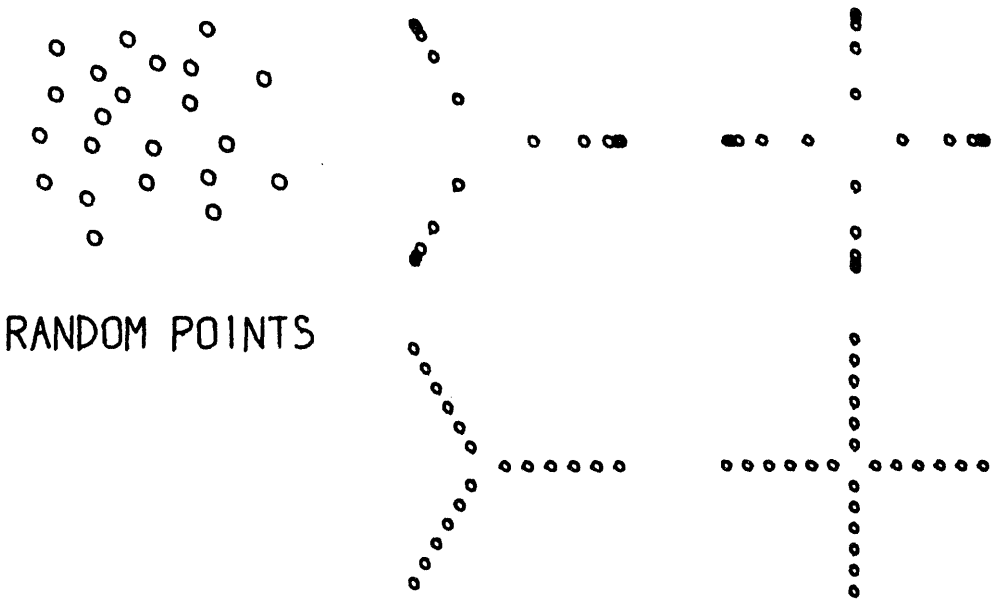


FIGURE 4.4.
DISPLAYS FOR PEN TRACKING

An attempt was made to combine various types of prediction according to the speed of motion of the pen, but all such efforts met with difficult stability problems and were interfering with more important parts of the research. Therefore, I decided to accept the ten per cent of time lost to tracking in order to proceed to more interesting things. Other workers, notably Rolland Silvers formerly of Bolt, Beranek and Newman, report better success with predictive tracking giving numbers like 3% loss.

Different methods of establishing the exact location of the light pen have been tried using many different shapes of display. For example, the displays shown in Figure 4.4 all seem to be about the same as far as time taken to establish pen position and accuracy. As far as I know, no one has taken into account the motion of the pen during the tracking display period. I use the logarithmic scan with four arms.

To initially establish pen tracking the Sketchpad user must inform the computer of an initial pen location. This has come to be known as "inking-up" and is done by "touching" any existing line or spot on the display whereupon the tracking cross appears. If no picture has yet been drawn, the letters INK are always displayed for this purpose.

DEMONSTRATIVE USE OF PEN

During the remaining 90% of the time that the light pen and display system are free from the tracking chore, spots are very rapidly displayed to exhibit the drawing being built, and thus the lines and circles of the drawing appear. The light pen is sensitive to these spots and reports any which fall within its field of view by the

interruption of a sequence change before another spot can be shown. The table within the computer memory which holds the coordinates of the spots also contains a tag on each one as shown in Figure 4.5 so that the picture part to which this spot belongs may be identified if the spot should be seen by the pen.

A table of all such picture parts which fall within the light pen's field of view is assembled during one complete display cycle. At the end of a display cycle this table contains all the picture parts that could even remotely be considered as being "aimed at." During the next display cycle a new table is assembled which at the end of that cycle will replace the one then in use. Thus, two storage spaces are provided, one for assembling a complete table of display parts seen, the other for holding the complete table from the last display cycle so that the aiming computation described below in the sections on demonstrative language and pseudo pen location may avoid using a partially complete table. Note that since the display of the TX-2 is independent of the computations going on, the aiming computation may occur in the middle of a display cycle.

Due to the relatively long time that a complete display cycle for a complicated drawing may take, the aiming computation, by using information from the previous complete display cycle, took excessive time to "become aware" of picture parts newly aimed at by the pen. Therefore, I require that any display part seen by the light pen which is not yet in the table being built for the current display cycle be put not only in that table, but also in the table for the previous display cycle if not already there. This speeds up the process of locking onto elements of

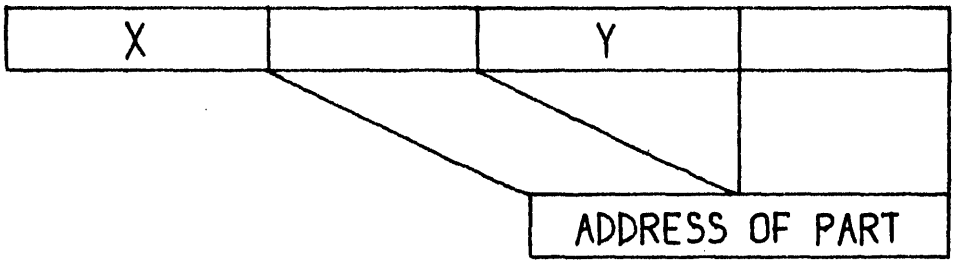


FIGURE 4.5.
ADDRESS IN DISPLAY REGISTER

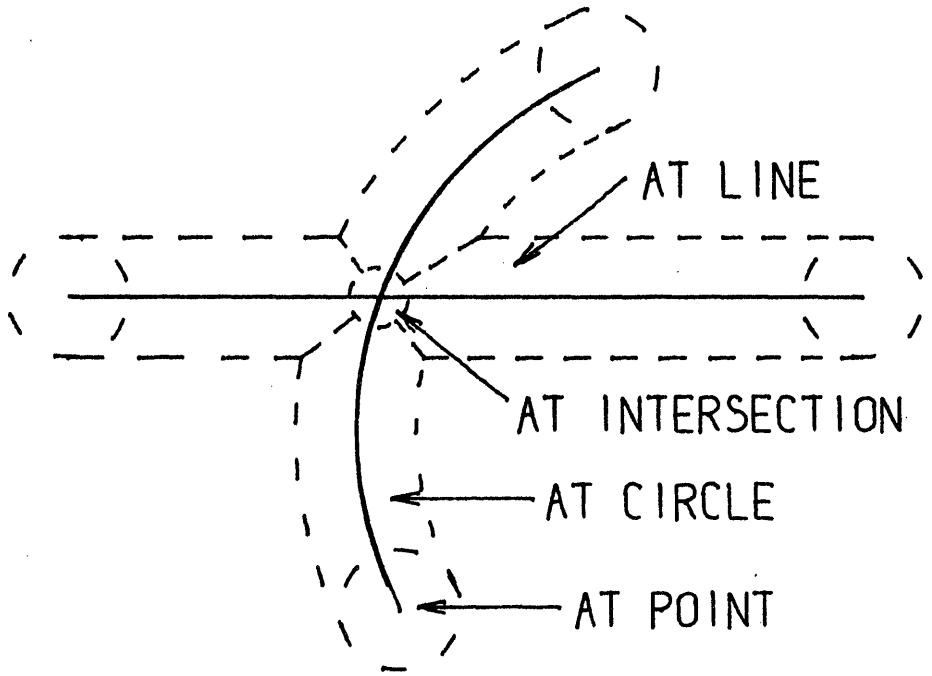


FIGURE 4.6.
OPERATION OF PSEUDO PEN LOCATION

the drawing. Similarly, the information from a previous display cycle may contain many previously seen drawing parts which are not currently within the light pen's field of view, especially if the light pen has moved an appreciable distance since the last complete display cycle. One might attempt to detect large pen displacements during a display cycle and indicate that the old light pen information is too obsolete to use if such displacements occur. However, I have often found it handy to slide appreciable distances along a line or curve, in which case the light pen information is not made entirely obsolete. Therefore, no such obsolescense-by-displacement routine has been incorporated into the Sketchpad system.

DEMONSTRATIVE LANGUAGE

The table of picture parts falling within the field of view of the light pen, assembled during a complete display cycle, contains all the picture parts which might form the object of a statement of the type:

apply function F to _____.

e.g. erase this line (circle, etc.). Since the one half inch diameter field of view of the light pen is relatively large with respect to the precision with which it may be manipulated by the user and located by the computer, the Sketchpad system will reject any such possible demonstrative object which is further from the center of the light pen than some small minimum distance; about 1/8 inch was found to be suitable. Although it is easy to compute the distance from the center of the light pen field to a line segment or circle arc, it is not possible to compute

the distance from the light pen field center to a piece of text or a complicated symbol represented as an instance. For every kind of picture part some method must be provided for computing its distance from the light pen center or indicating that this computation cannot be made.

The distance from an object seen by the light pen to the center of the light pen field is used to decrease the size of the light pen field for aiming purposes. A light pen with two concentric fields of view, a small inner one for demonstrative purposes, and a larger outer one for tracking would make this computation unnecessary and would give better discrimination between objects for which no distance computation exists. Lack of this discrimination is now a problem. Design of such a pen is easy, and consideration of its development for any future large scale use of engineering drawing programs should be given serious consideration.

After eliminating all possible demonstrative objects which lie outside the smaller effective field of view, the Sketchpad system considers objects topologically related to the ones actually seen. End points of lines and attachment points of instances are especially important, but objects on which constraints operate, or the value of a number as opposed to the digits which represent this value may also be considered. Such related objects may not specifically appear in the drawing but it must be possible to reference them easily. If any such object is sufficiently close to the center of the light pen field, it is added to the table of possible demonstrative objects even though it may have no display and, therefore, was not seen by the light pen.

As described above, the aiming or demonstrative program first eliminates from further consideration objects which are too far from the

center of the light pen field to reduce the effective size of the field for aiming purposes. Next it brings into consideration unseen objects related to the objects actually seen. After these two procedures the number of objects still under consideration determines the further course of action. If no objects remain under consideration, nothing is being aimed at. If one object, it is the demonstrative object and the light pen is said to be "at" it, e.g., the pen is at a point, at (on) a line, at (on) a circle, or "at" a symbol (instance). If two objects remain, it may be possible to compute an intersection of them. If the intersection is sufficiently close to the pen position, the pen is "at" the intersection. With two or more objects remaining, the closest object is chosen if such a choice is meaningful; or if not, no object is pointed at, i.e., there is no demonstrative object.

The above consideration of the demonstrative program has been left vague and general purposely to point out that the specific types of objects being used in a drawing differ only in the details of how the various computations are made. For example, although the Sketchpad system is not now able to do anything with curves other than circle arcs and line segments, the demonstrative program requirements to add conic sections to the system, as it stands, involve only the addition of computation procedures for the distance from the pen location to the conic, routines for computing the intersection of conics with conics, lines, and circles, and some indication of what topologically related objects, e.g. foci, need be considered. Figure 4.6 outlines the various regions within which the pen must lie to be considered "at" a line segment, a circle arc, their end points, or their intersection. The relative sizes

of the error tolerated in the "sufficiently close to" statements above are indicated as well. The error tolerated is a fixed distance on the display so that confusion because objects appear too close together can usually be resolved by enlarging the drawing as described in Chapter V.

The organization of the demonstrative program in Sketchpad is in the form of a set of special cases at present. That is, the program itself tests to see whether it is dealing with a line or circle or point or instance and uses different special subroutines accordingly. This organization remains for historical reasons but is not to be considered ideal at all. A far better arrangement is to have within the generic block for a type of picture part all subroutines necessary for it.

PSEUDO PEN LOCATION

The demonstrative program computes for its own use the location on a picture part seen by the light pen nearest the center of the pen's field of view. It also computes the location of the intersection of two picture parts. Thus when the demonstrative program decides which object or intersection the light pen is at, an appropriate pseudo pen location has also been computed. If no object has been named as demonstrative object, the pseudo pen location is taken to be the actual pen location. The statements "at a line," "at a circle," and "at a point" take on true significance, for the pseudo pen location will indeed be at these objects.

The pseudo pen location is displayed as a bright dot which locates itself ordinarily at the center of the pen tracking cross. It is easy

to tell when the demonstrative object is a line, circle, point, or intersection, because this bright dot locks onto the picture part and becomes temporarily independent of the exact pen location. The pseudo pen location or bright dot is used as the point of the pencil in all drawing operations; for example, if a point is being moved, it moves with the pseudo pen location. As the light pen is moved into the areas outlined in Figure 4.6 and the pen locks onto existing parts of the drawing, any moving picture parts jump to their new locations as the pseudo pen location moves to lie on the appropriate picture part. The pseudo pen location at the instant that a new line or circle is created is used as the coordinates of the fixed end of that line or circle.

With just the basic drawing creation and manipulation functions of draw, move, and delete and the power of the pseudo pen location and demonstrative language programs, it is possible to make fairly extensive drawings. Most of the constructions normally provided by straight edge and compass are available in highly accurate form. Most important, however, the pseudo pen location and demonstrative language give the means for entering the topological properties of a drawing into the machine.

Chapter V

DISPLAY GENERATION

The display system, or "scope," on the TX-2 is a ten bit per axis electrostatic deflection system able to display spots at a maximum rate of about 100,000 per second. A display instruction permits a single spot to be shown on the display at any one of slightly more than a million places, requiring 20 bits of information to specify the position of the spot. Due to the multiple sequence design of the TX-2 it is convenient to permit the display system to operate at its own speed. The display will request memory cycles whenever they are required to transmit more information to it, but the time actually taken in displaying a spot will not be lost, for the rest of the TX-2 may be involved with other operations meanwhile. It has been found useful, therefore, to store the locations of all the spots of a drawing in a large table in memory and to produce the drawing by displaying from this table. The display system, then, sees the rest of Sketchpad as 32,000 words of core storage. The rest of the Sketchpad is able to compute and store spot coordinates in the display table without regard to the timing of the display system.

The display spot coordinates are stored one to a memory word. The display subprogram displays each in turn, taking 20 microseconds each so that some time will be left over for computation. If instead of displaying each spot successively, the display program displays every eighth in a system of interlace, the flicker of the display is reduced greatly, but lines appear to be composed of crawling dots. For large displays made up mostly of lines such an interlace is useful. However, for

repetitive patterns of short lines, the effect may be that the entire drawing seems to dance because of synchronization between the interlace and the repetitive nature of the pattern. The interlace may be turned on or off under user control by means of a toggle switch.

Early display work with the display file led to the discovery by the author and others that if the spots were displayed at random, a twinkling picture resulted which is pleasing to the eye and avoids flicker entirely (see Figure 5.1). However, small detail is lost because of the eye's inability to separate the pattern from the random twinkle unless the pattern is gross. Twinkling, like interlace, is under user control by a toggle switch. Twinkling is accomplished by scrambling the order of the display spot locations in the display file. To do this, each successive entry is exchanged with an entry taken at random until every entry has been exchanged at least once. Needless to say, whether a scrambled file is displayed successively or by interlace makes no difference to its twinkling appearance.

MARKING OF DISPLAY FILE

Of the 36 bits available to store each display spot in the display file, 20 are required to give the coordinates of that spot for the display system, and the remaining 16 are used to give the address of the n-component element which is responsible for adding that spot to the display. Thus, all the spots in a line are tagged with the ring structure address of that line, and all the spots in an instance are tagged as belonging to that instance. The tags are used to identify the particular part of the drawing being aimed at by the light pen for demonstrative statements. See Chapter IV, Figure 4.5, p 61.

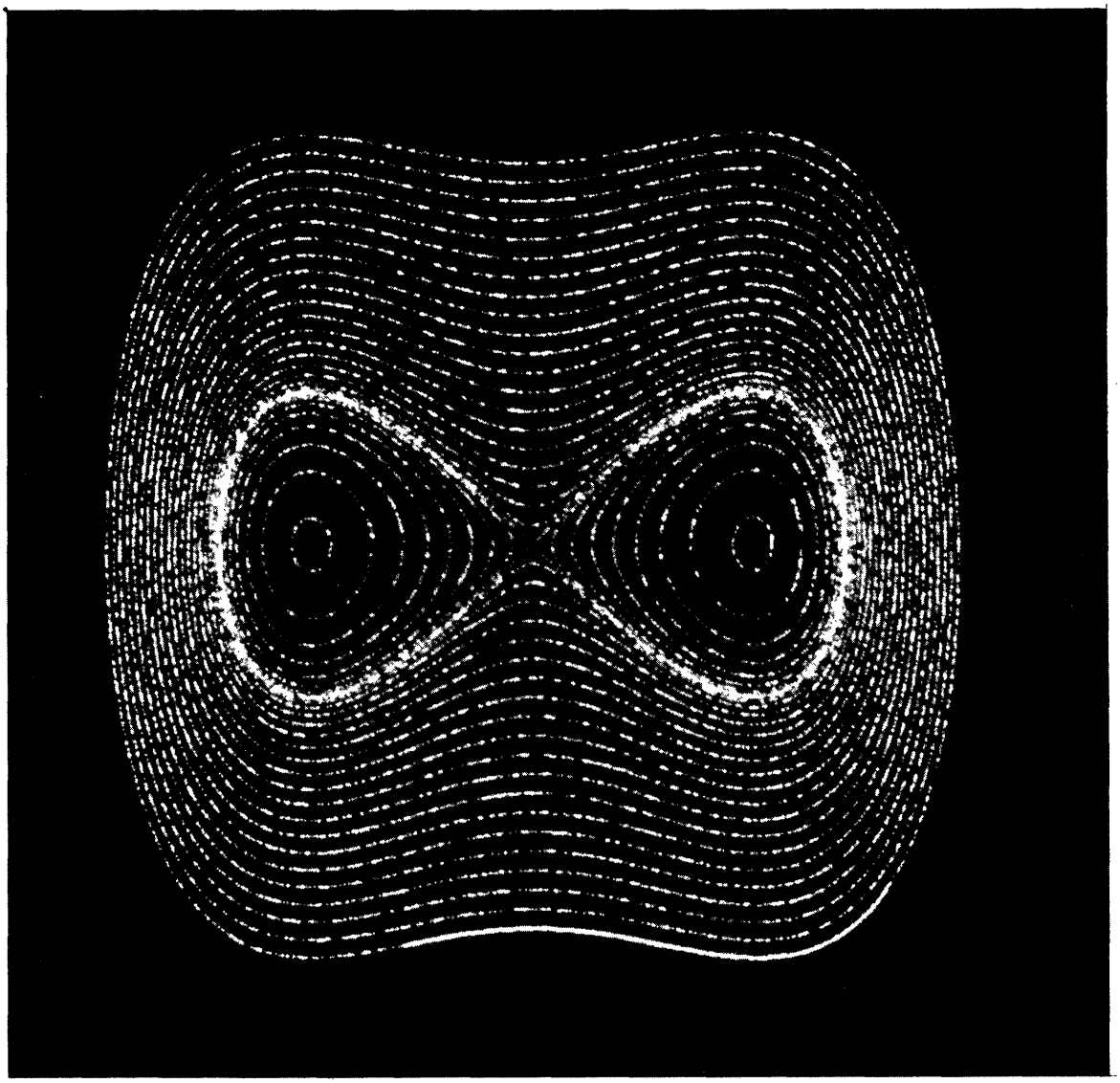


FIGURE 5.1. TWINKLING DISPLAY.

Displaying the spots of a large display in random sequence makes the display appear to "twinkle." This photograph was exposed only long enough to show about half of the spots of a twinkling display. It conveys the impression of a twinklings display as well as any still picture can.

The curves are of the equation $x^4 - x^2 + y^2 = a^2$ for several values of a. They were drawn by another program rather than by Sketchpad.

If a part of the drawing is being moved by the light pen, its display spots will be recomputed as quickly as possible to show it in successive positions. The display spots for such moving parts are stored at the end of the display file so that the display of the many non-moving parts need not be disturbed. Moving parts are made invisible to the light pen so that the demonstrative and pseudo pen location computations described in Chapter IV will not "lock on" to parts moving along with the pen.

COORDINATE SYSTEMS

The coordinate system of the TX-2 display system has origin at the center of the scope and requires ten bits of deflection information located at the left of 18 bit computer subwords for each axis. Treatment of these numbers as signed fractions of full scope deflection leads to the most natural programming because of the fixed point, signed fraction nature of the TX-2 multiply and divide instructions. The scope coordinate system is natural to the ability of the TX-2 to perform arithmetic operations simultaneously on two 18 bit half words. It is not suitable for representing variables with more than two components, nor is the precision available in 18 bits adequate for all the operations for which the Sketchpad system is applicable.

For convenience in representing many component variables and for more than 18 bit precision, Sketchpad uses an internal coordinate system for drawing representation divorced from the representation required by the display system. This internal system is called the "page" coordinate system. In thinking of the drawings in Sketchpad, the page

coordinates are considered as fixed. A page to scope transformation gives the ability to view on the scope any portion of the page desired, at any degree of magnification, as if through a magnifying glass. The magnification feature of the scope window-into-the-page makes it possible to draw the fine details of a drawing. The range of magnification of 2000 available makes it possible to work, in effect, on a 7-inch square portion of a drawing about 1/4 mile on a side.

TRANSFORMATIONS AND SCALE FACTORS

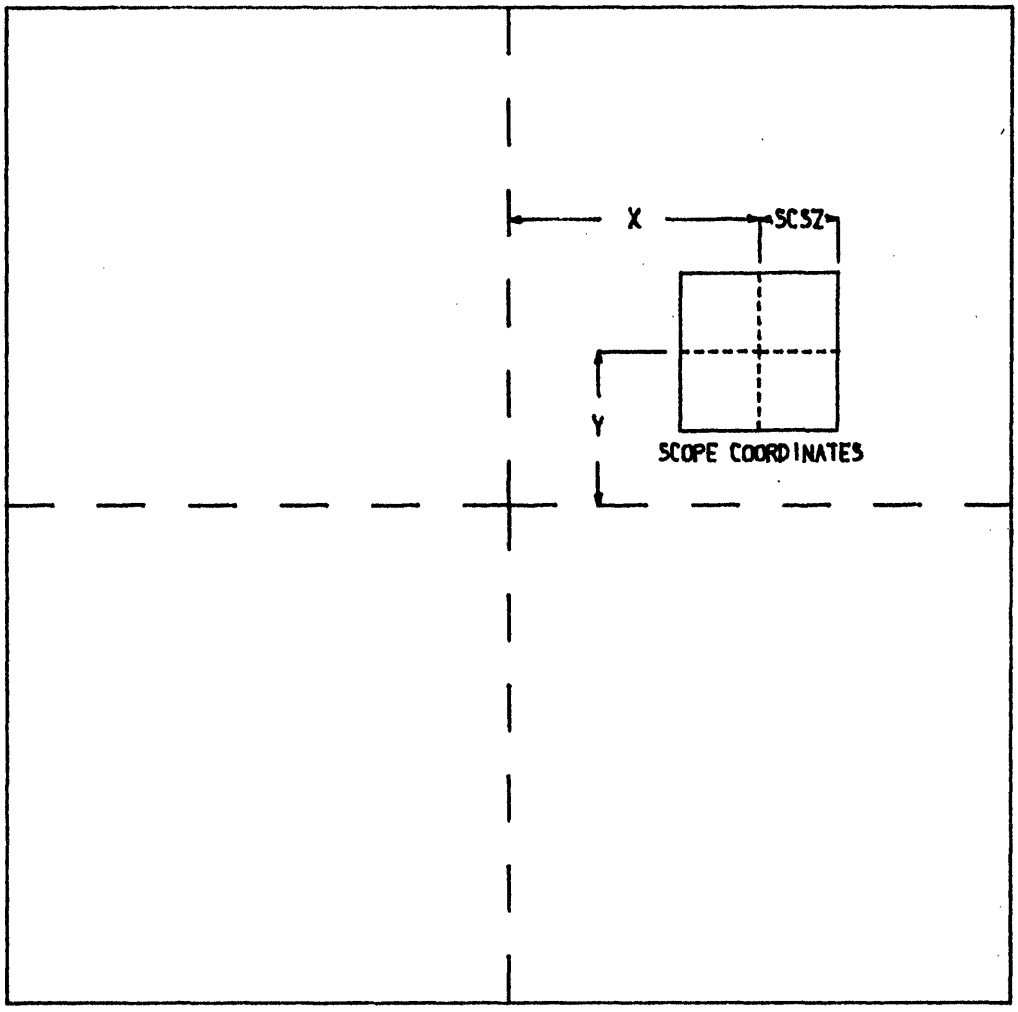
The page coordinate system is intended for use only internally and will always be translated into display or plotter coordinates by the output display subroutines. Therefore, it is impractical to assign any absolute scale factor to the page coordinate system itself; it is meaningless to ask how big is the page. It is, however, very important to know how big the visible representations of Sketchpad drawings will be, for one must make drawings in the correct sizes if one is to communicate with machine shops. Dimensions indicated on the drawing must correspond to the dimensions of the drawing in its final form if full-size drawings are to be produced. The computer's only concern with the actual size of the page coordinate system is to know what decimal number should be displayed for the value of a certain distance in page coordinates. As Sketchpad now stands, the value is such that one-to-one scale drawings can be produced on the plotter if dimensions are read in units of thousandths of an inch.

Page coordinates, then, are dimensionless signed fractions, 36 bits long which are considered as fixed when considering drawing representations. In order to avoid the troubles of overflow, it is made difficult

for the user to generate page coordinates with values in the most significant six or seven bits of the 36 allowed. This is done by artificially limiting the maximum part of the page displayed on the scope to 1/256 of the page's linear dimension. The 29 or 30 bits of precision which remain are sufficient for all applications. The maximum magnification of the display is also limited so that the "grain" of the page coordinates cannot show on the display. The 2000-to-one scale change mentioned above remains.

A scale factor for the display controls the size of the square which will appear on the scope. The actual number saved is the half-length of the side of the square, called SCSZ for SCope SiZe as shown in Figure 5.2. Also saved are the page coordinates of the center of the scope square. By changing these numbers the portion of the page shown on the scope may be changed in size and moved, but not rotated.

The shaft position encoder knobs below the scope (see Figure 1.2, p.11) are used to control the scale factor and square positioning numbers indicated above. Rotation of the knobs tells the program to change the display scale factor or the portion of the page displayed. In order to obtain smooth operation at every degree of magnification, unit knob rotations produce changes in the scope size and position numbers proportional to the existing scope size number, SCSZ. Rotation of the scale change knob, therefore, causes exponential increase or decrease in SCSZ and this results in apparent linear change in the view on the scope.



PAGE COORDINATES

FIGURE 5.2. COORDINATE SYSTEMS

INSIDE OUT AND OUTSIDE IN DISPLAY

How the direction of rotation of the knobs affects the translation of the display is important from the human factors point of view. It is possible to think of moving the scope window above the page or moving the drawing beneath the window. Since to the user the scope is physically there, and no sense of body motion goes with motion of the window, the knobs turn so that the operator thinks of moving the drawing behind his window: rotation to the right results in picture motion to the right or up. Similarly, rotation of another knob to the right results in rotation of picture objects to the right as seen by the user. No such convenient manner of thought for the scale knob has been found. Users get used to either sense of change about equally poorly; the major user so far (the author) still must try the knob before being sure of which way it should be turned.

The translation knobs were primarily used to locate a portion of the picture in the center of the scope so that it could be enlarged for detailed examination. To make centering easier, a special function was provided which relocates the picture so that the immediately preceding light pen position is centered. The knobs are now used for fine positioning of the picture to make the scope display all of an area which just barely fits inside it. The light pen could perhaps be used to control scope size and positioning without reference to the knobs at all, perhaps with a coarse and fine control. The question of what controls are best suited to humans is wide open for investigation.

COORDINATE CONVERSION AND EDGE DETECTION

The reason for having the page-scope transformation in terms of the location of the scope center and the size of the scope is that this form makes it very easy to transform page coordinates into scope coordinates.

$$\frac{\text{PAGE COORDINATE - CENTER OF SCOPE}}{\text{SCOPE SIZE}} = \text{SCOPE COORDINATE}$$

The process of division will yield overflow if the point converted does not lie on the scope. However, one can little afford the time that application of this transformation to each and every spot in a line would require. It is necessary, therefore, to compute which portion(s) of a curve will appear on the scope, and generate ONLY those portions for the human to see. The edge detection problem is the problem of finding suitable end points for the portion of a curve which appears on the scope.

In concept the edge detection problem is trivial. In terms of program time for lines and circles the problem is a small fraction of the total computational load of the system, but in terms of program debugging difficulty the problem was a lulu. For example, the computation of the intersection of a circle with any of the edges of the scope is easy, but computation of the intersection of a circle with all four edges may result in as many as eight intersections, some pairs of which may be identical, the scope corners. Now which of these intersections are actually to be used as starts of circle arcs?

THE SERVICE PROGRAM - LINE AND CIRCLE GENERATION

As the Sketchpad system now exists, all displays are generated from straight line segments, circle arcs, and single points. The details of generating the specific display spots for each of these types of display is relegated to a "service" program. The service program also contains the actual display sub-program for displaying the spots and retains control over the input and output to the display file. The service program takes care of the transformation of coordinates from page coordinates to scope coordinates and computes the portion of the line, circle, or point to be shown, if any. Since these service functions have been working correctly, further programming was not required to make reference to the details of scope size, position, coordinate transformation, or display. For example, the routine which displays text on the scope uses the line and circle service programs to compose each letter.

The independence of the bulk of the program from the specifics of display is a very valuable asset for future expansion and change to the system. For example, when a line drawing scope capability was added to the TX-2, only the service program needed to be changed to accommodate it. Moreover other people can and do use the service subroutines in their programs. The attitude of independent parts divided by independence of function pervades the Sketchpad system; being forced to divide the program into several binary portions because it was, in toto, too big to handle, I divided it in the most natural places I could find.

The actual generation of the lines and circles for the present spot display scope is accomplished by means of the difference equations:

$$x_i = x_{i-1} + \Delta x \qquad y_i = y_{i-1} + \Delta y \qquad (5\text{-}1)$$

for lines, and

$$x_i = x_{i-2} + \frac{2}{R}(y_{i-1} - y_c)$$
$$y_i = y_{i-2} - \frac{2}{R}(x_{i-1} - x_c) \qquad (5\text{-}2)$$

for circles, where subscripts i indicate successive display spots, subscript c indicates the circle center, and R is the radius of the circle in Scope Units. In implementing these difference equations in the program the fullest possible use is made of the coordinate arithmetic capability of the TX-2 so that both the x and y equation computations are performed in parallel on 18 bit subwords. Including marking the points in the display file with the appropriate code for the ring structure block to which they belong (two instructions), and indexing, the program loops contain five instructions for lines and ten for circles. About 3/4 of the total Sketchpad computation time is spent doing these 15 instructions!

CIRCLE CLOSURE

It is an unfortunate property of difference equation approximation to differential equations that the tiny errors introduced by the finite approximation may accumulate to produce gross noticeable errors. Although the difference equation (5-2) listed above for circle generation may seem more complicated than necessary, it is the small details of the equation that make it useable. Considerable effort was required to find an equation which produced faithful circles and could be implemented to take advantage of the parallel 18 bit arithmetic available in the TX-2. Other equations tried either generated logarithmic spirals due to

mathematical inadequacies, required more than 18 bit precision to operate accurately, or required serial processing of the x and y equations, which would consume more time.

For example, the difference equations:

$$\begin{aligned} x_i &= x_{i-1} + \frac{1}{R}(y_{i-1} - y_c) \\ y_i &= y_{i-1} - \frac{1}{R}(x_{i-1} - x_c) \end{aligned} \tag{5-3}$$

produce a logarithmic spiral which grows about (π x step size) in "radius" each turn. This spiral divergence is predicted theoretically and is unrelated to any roundoff error. It could be avoided by using the equations:

$$\begin{aligned} x_i &= \frac{R}{\sqrt{1 + R^2}}\left\{x_{i-1} + \frac{1}{R}(y_{i-1} - y_c)\right\} \\ y_i &= \frac{R}{\sqrt{1 + R^2}}\left\{y_{i-1} - \frac{1}{R}(x_{i-1} - x_c)\right\} \end{aligned} \tag{5-4}$$

but the term $\frac{R}{\sqrt{1 + R^2}}$ is so little different from unity for the usual values of R that it cannot be represented in 18 bits. The simple change from (5-3) to the equations:

$$\begin{aligned} x_i &= x_{i-1} + \frac{1}{R}(y_{i-1} - y_c) \\ y_i &= y_{i-1} - \frac{1}{R}(x_i - x_c) \end{aligned} \tag{5-5}$$

where a new position of x is used to generate the next position of y. Equations (5-5) approximate a circle well enough and are known to close exactly both in theory and when implemented, but because the x and y equations are dissimilar, they cannot make use of TX-2's ability to do two 18 bit additions at once. Note, however, that Equations (5-5) are ideally suited for implementation on machines which can perform only

one addition at a time. In fact, Sketchpad uses Equations (5-5) to generate the sine and cosine functions used for rotations.

DISPLAY PROGRAMS

The display programs for line and circle segments are simply the line and circle drawing subroutines plus a small program which extracts the pertinent numerical information from the ring structure to locate the line or circle segment properly. A similar routine for drawing dotted lines and dotted circles would be useful—the same manipulations that apply to lines and circles could be applied to the dotted curves as well. To be consistent with the existing programs the dotted line program would use the line or circle drawing subroutine many times, once for each dot. Although this would be somewhat inefficient in that the values of Δx and Δy in (5-1) would be recomputed each time, it could be made to work with the minimum programming difficulty. Alternatively, a special dotted line subroutine could be written. This would be especially appropriate if output devices were used for which dotting could be accomplished in a special way as, for example, lifting the plotter pen periodically while it is tracing a curve.

Another variation on lines and circles would permit making lines of various weights or with different styles of dots: center lines and the like. These could each be put into the system as a different type of line, or all could be treated as a single type with some numerical specification of the line characteristics. For example, two scalars might be used to indicate approximate dot frequency and the ratio of dot length to dot period. A single scalar might specify the line weight.

It is important that the properties of such a scalar would be the unitless properties of ratios, invariant under changes to the scale of the drawing and the transformations of instances. The existing scalar with the dimension of length would not serve.

Text, to put legends on a drawing, is displayed by means of special tables which indicate the locations of line and circle segments to make up the letters and numbers. Each piece of text appears in a single line not more than 36 characters in length of equally spaced characters which can be changed by typing. Digits to display the value of an indicated scalar at any position and in any size and rotation are formed from the same type face as text. It is possible to display up to five decimal digits with sign; binary to decimal conversion is provided, and leading zeros are suppressed. Whatever transformation is applied to the magnification of subpictures applies also to the value displayed by the digits. Digits which indicated lengths when a subpicture was originally drawn remain correct however it is used. Digits are intended for making size notations on drawings by means of dimension lines.

The instance, as will be described more fully in Chapter VI, behaves as a single entity. The display spots which represent all the internal parts of instance must be marked with the address of the instance block rather than with the address of the actual line or circle blocks which are the indirect cause of the spots. The instance expansion program makes use of the line, circle, number, and text display programs and itself to expand the internal structure of the instance. A marker is used so that during expansion of an instance, display spots retain the instance marking.

Expansion of instances may be a most time consuming job. When just the existence of an instance is important, but not its internal character, one can display a frame around the instance without having its internal structure show. The framing and expansion of instances are individually controlled by toggle switches. The instance frame is a square drawn around the outline of the instance, that is, the smallest square which fits around the master of the instance in upright position. The size and location of this square are computed whenever a drawing is filed away, provided that no instances of the drawing exist. In fact, the drawing is relocated so that the center of the frame is always at the origin of the page coordinate system. This is done so that the coordinate system of an instance will have origin at about the center of the instance. If instances of the picture exist, the program refrains from relocating picture origin because to do so would slightly relocate all instances of the picture in the other direction.

The instance expansion routine does some edge detection in a crude way to avoid spending inordinate amounts of time deciding that each line and circle in an instance grossly off the scope is individually off the scope. Instances are not expanded unless there is a fair chance that some part of them will appear. The instance outline box is used for this purpose: the instance is not expanded if its center is more than 1.5 times as far from the scope edge as its box size. Since the relatively new addition of avoiding box size recomputation and translation of a picture if instances of it exist, it is possible to have parts of an instance extend any distance outside their box. Therefore, instance parts might disappear inexplicably. This has, however, never been observed in practice.

A more complete treatment of the size of an instance for edge detection which would cure the difficulties outlined above could be made. One would compute not only the size of the smallest outlining square each time an un-instanced drawing is filed away, but also the size of the smallest surrounding circle each time the drawing is filed away, whether or not it is instanced. The smallest circle would be used to determine whether a particular instance was worth expanding at all, or; if the entire circle was contained on the scope, it would indicate that further edge detection would be entirely unnecessary. In computing the smallest enclosing circle, needless to say, subpictures would be considered only as objects which occupy their smallest enclosing circle; internal structure of instances would be ignored. Whereas now only the smallest enclosing box can be seen, in the proposed more complete treatment either the smallest enclosing square or circle could be displayed.

DISPLAY OF ABSTRACTIONS

The usual picture for human consumption displays only lines, circles, text, digits, and instances. However, certain very useful abstractions are represented in the ring structure storage which give the drawing the properties desired by the user. For example, the fact that the start and end points of a circle arc should be equidistant from the circle's center point is represented in storage by a constraint block. To make it possible for a user to manipulate these abstractions, each abstraction must be able to be seen on the display if desired. Not only does displaying abstractions make it possible for the human user to know that they exist, but also displaying abstractions makes it

possible for him to aim at them with the light pen and, for example, erase them. The light pen demonstrative language described in Chapter IV is sufficient for making all changes to objects or abstractions which can be displayed. To make Sketchpad's light pen language universal, all objects and abstractions represented in Sketchpad's ring structure can be displayed. To avoid confusion, the display for particular types of objects may be turned on or off selectively by toggle switches. Thus, for example, one can turn on display of constraints as well as or instead of the lines and circles which are normally seen.

If their selection toggle switch is on, constraints are displayed as shown in Figure 5.3. The central circle and letter are of fixed size on the scope regardless of the drawing scale factor and are located at the average location of the variables constrained. The four arms of a constraint extend from the top, right side, bottom, and left side of the circle to the first, second, third, and fourth variables constrained, respectively. If fewer than four variables are constrained, excess arms are omitted. In Figure 5.3 the constraints are shown applied to "dummy variables," each of which shows as a $\times$.

Two difficulties are encountered with this representation of constraints:

1. The constraint diagrams tend to overlap one another when a geometric figure has several constraints applied to it, and

2. One character is not enough to display all the symbols and mnemonics one would like to have for his constraints.

A more desirable arrangement would let the user draw the constraint representation diagrams in the same way he makes other drawings, permitting him to invent whatever mnemonics he could draw. It would also

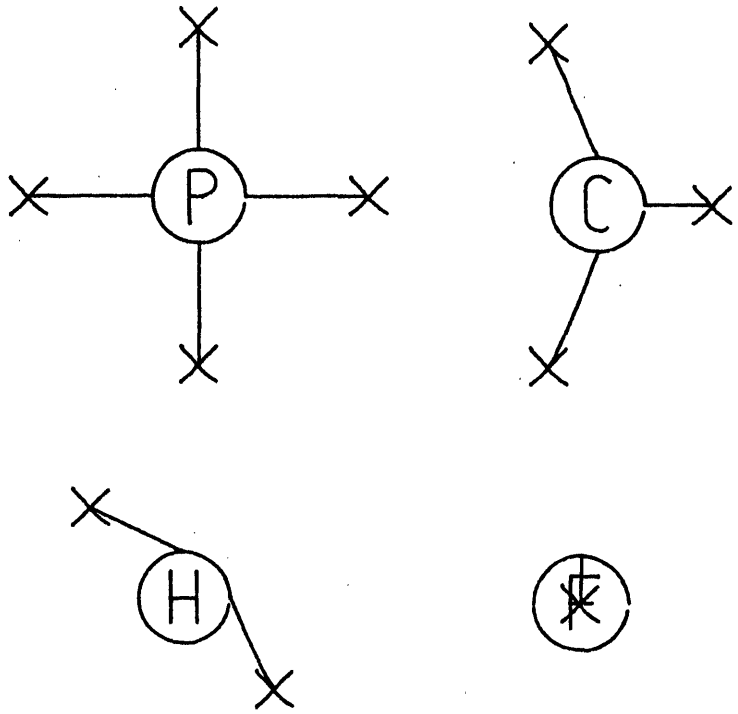


FIGURE 5.3.
DISPLAY OF CONSTRAINTS

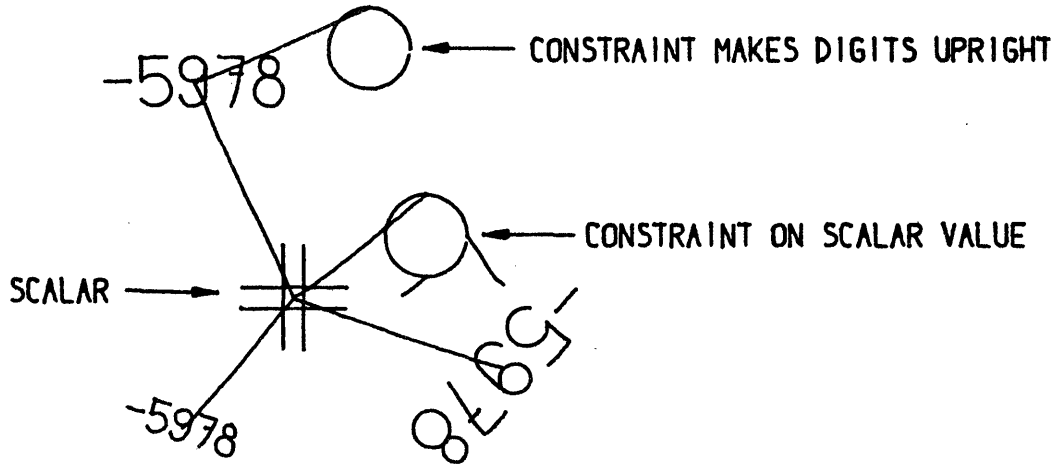


FIGURE 5.4.
DISPLAY OF SCALAR AND DIGITS

be nice to be able to relocate the body of a constraint representation at will to avoid the unfortunate and confusing overlapping. How to locate it without explicit instructions would, however, be a problem. Moreover, the constraint, having a position itself, would have to be treated as a variable and might be used to constrain itself, compounding an already messy business. Alternatively, instead of locating the circle and letter at the center of the variables one could locate them at random nearby. Any confusion of constraints could then be clarified by recomputing the display file to get a new set of random locations.

Another abstraction that can be displayed if desired is the value of a set of digits. The value of a set of digits is stored as a variable separate from the digits themselves. Moving digits means relocating them on the drawing or rotating them. Making the digits bigger means just that, increasing the type size. But making the value bigger changes the particular digits seen and not the type size. The value of a set of digits, a scalar, appears as a # connected to the digits which display it by as many lines as there are sets of digits and located at the average location of these sets, as shown in Figure 5.4. Since there is usually only one set of digits displaying the value of a scalar, the # is usually superimposed on it and connected to it by a zero length line which looks like a dot. The major difficulty with this display is that values which have no digits all lie exactly on top of one another at the origin.

EMPTY DISPLAYS

The frames which may be put around instances can be thought of as abstractions of the existence as opposed to the appearance of the

instance. Moreover, since it is possible to make an instance of a picture and then erase the lines in the master picture, it is possible to have an instance with no appearance at all, an empty instance. Before instance framing was possible such empty instances were inaccessible to the light pen and likely to be forgotten by the user because they could not show on the display. At the present time it is possible to lose only text; a line of text composed entirely of spaces does not show.

THE AS YET UNDREAMT OF THINGS THAT WILL BE DISPLAYED

The organization of Sketchpad display as a set of display subroutines with identical external properties makes it possible to add new kinds of displays to the system with the greatest ease. At the present time the need for dotted lines and circles, including center lines, dark lines, etc., and the need for a ratio type unitless scalar for representing angles and proportions is clear. Conic sections would be useful. What other kinds of things may become useful for special purposes is as yet unknown; Sketchpad attempts to be big enough to incorporate anything easily.

Chapter VI

RECURSIVE FUNCTIONS

In the process of making the Sketchpad system operate, a few very general functions were developed which make no reference at all to the specific types of entities on which they operate. These general functions give the Sketchpad system the ability to operate on a wide range of problems. The motivation for making the functions as general as possible came from the desire to get as much result as possible from the programming effort involved. For example, the general function for expanding instances makes it possible for Sketchpad to handle any fixed geometry subpicture. The rewards that come from implementing general functions are so great that the author has become reluctant to write any programs for specific jobs.

Each of the general functions implemented in the Sketchpad system abstracts, in some sense, some common property of pictures independent of the specific subject matter of the pictures themselves. For example, the instance expansion program is a representation of the fact that pictures from many fields contain subpictures with relatively fixed appearance. It is not claimed that the general functions described in this chapter form a complete set, that is, abstract all the common properties of pictures. There is a definite need for a general purpose function for making topological changes to a drawing. Such a general purpose system is necessary, for example, to put fillets and rounds on corners, or to be able to define a vocabulary of dotted lines which could be, "unreeled," as it were, to any desired length. Nevertheless, the power obtained from the small set of generalized functions in Sketchpad is one of the most important results of the research.

In order of historical development, the recursive functions in use in the Sketchpad system are:

1. Expansion of instances, making it possible to have subpictures within subpictures to as many levels as desired.

2. Recursive deletion, whereby removal of certain picture parts will remove other picture parts in order to maintain consistency in the ring structure.

3. Recursive merging, whereby combination of two similar picture parts forces combination of similarly related other picture parts, making possible application of complex definitions to an object picture.

4. Recursive moving, wherein moving certain picture parts causes the display of appropriately related picture parts to be regenerated automatically.

PUSH DOWN LISTS

A common method of keeping track of the recursion process is to use, a "push down list," a device much like a sinking table used in cafeterias to hold dishes so that as a dish is removed the next is ready. Each of the entries of a push down list references the next, so that if one is removed, the location of the next will be available. A peculiarity of the Sketchpad system is that these push down lists are formed directly in the data storage structure and not separately by the program. This guarantees that if the data storage structure fits in memory, it may be fully recursed without risk that the push down information overflow the space available for it. As far as possible, Sketchpad uses parts of the data structure otherwise used for other purposes to perform the push down function.

Chapter III and Appendix C described the ring structure used for primary picture storate in the Sketchpad system and showed the relationships between various kinds of blocks. In this section as little reference

as possible will be made to the exact nature of the blocks involved, because by avoiding reference to specific structure the functions considered may be made applicable to any specific structure. By way of example, however, some specific cases will be mentioned; bear in mind that these are meant only to be illustrative.

DEPENDENT AND INDEPENDENT ELEMENTS

Certain picture elements depend in a vital way for their existence, display, and properties on other elements. For example, a line segment must reference two end points between which it is drawn; a set of digits must reference a scalar which indicates the value to be shown. In three dimensions it might be that a surface is represented as connecting four lines which in turn depend on end points. If a particular thing depends on something else there will be in the dependent thing a reference by pointer to the thing depended upon. In the ring structure used in Sketchpad, there will be a ring with a "hen" pair in the thing depended on and at least one "chicken" pair in a dependent thing. For example, a ring will connect a point with all lines which use it as an end point; the chicken pairs of this ring, being in the blocks for the lines in question, point to the point as an end point of the lines.

Since there may be any number of rings passing through a given block, a particular block may depend on some other blocks and simultaneously be depended on by others. Such a block contains both hens and chickens. In particular, all blocks contain at least one chicken which indicates by a reference to a generic block the type of thing represented. Some things are otherwise totally depended upon, e.g. points, some things are totally dependent, e.g. lines, and some both depend and are depended on, e.g. instances.

RECURSIVE DELETING

Consistency is of course maintained if a single thing upon which no other thing depends is deleted. To accomplish this, all chicken pairs in its block are removed from their corresponding rings. The registers which comprised a deleted block are declared "free" by their addition to the FREES storage ring. In the Sketchpad system, line segments are entirely dependent and may be deleted without affecting anything else. However, deleting a line may leave end points on the drawing with no lines attached to them. A special button is provided for removing all such useless points from the drawing.

If a thing upon which other things depend is deleted, the dependent things must be deleted also. For example, if a point is to be deleted, all lines which terminate on the point must also be deleted. Otherwise, where would these lines end? Similarly, deletion of a variable requires deletion of all constraints on that variable; a constraint must have variables to act on. Three dimensional surfaces might be made to depend on lines which depend on points; if so, deletion of a point would require deletion of a line which would in turn require deletion of a surface. In Sketchpad, deleting a scalar forces deletion of all digits displaying its value, which will force deletion of all constraints holding the digits in position. Although the scalar-digits-constraint chain is the longest one in Sketchpad, the programs could handle longer chains if they existed.

The recursiveness of deletion brings with it the difficulty that one deletion may cause any number of deletions. It may therefore be difficult to follow the ring structure during deletions. For example, suppose that everything in a particular picture is to be deleted, a facility which is provided. The program applies the delete routine to

the first thing in the picture, say a point, and then to the next thing in the picture, say a line which terminated on the point. The normal macro mentioned in Chapter III for applying functions to all the members of a ring, LGORR, cannot be used, for at the time the next ring member is to be located, both it and the current ring member may be so much meaningless free storage. To delete everything in a picture, Sketchpad again and again deletes the first thing in the picture, thus chewing away until the picture is gone.

The push down list for recursive deletion is formed with the pair of registers which normally indicates what type of thing a block represents. As soon as it is found that a block must be deleted, it is declared "dead" by placing its TYPE pair in a generic ring called DEADS. The first dead thing is then examined to see if it forces other things to be declared dead, which is done until no more dead things are generated by the first dead thing. The first dead thing is then declared "free" and the new first dead thing is examined in exactly the same way until no more dead things exist. The DEADS ring, through registers which normally indicate type, serves as the push down list.

RECURSIVE MERGING

The single most powerful tool for constructing drawings, when combined with the definition copying function described in Chapter VII, is the ability to merge picture parts recursively. The recursive merge function makes it possible to make statements such as "this thing is to be related to that thing in such and such a way." The relationship may be treated as applying to things which it relates only indirectly. For example we shall soon see how one line may be made parallel to another even though the parallelism constraint applies only to the locations

of their end points. Similarly, a set of digits can be forced to display the length of a line, even though the constraint involved refers to the end points of the line and the value of the digits rather than to the line or the digits themselves. The recursive merge function makes it meaningful to combine anything with anything else of the same type regardless of whether the things are dependent on other things or depended on by others.

If two things of the same type which are independent are merged, a single thing of that type results, and all things which depended on either of the merged things depend on the result* of the merger. For example, if two points are merged, all lines which previously terminated on either point now terminate on the single resulting point. In Sketchpad, if a thing is being moved with the light pen and the termination flick of the pen is given while aiming at another thing of the same type, the two things will merge. Thus, if one moves a point to another point and terminates, the points will merge, connecting all lines which formerly terminated on either. This makes it possible to draw closed polygons.

If two things of the same type which do depend on other things are merged, the things depended on by one will be forced to merge, respectively, with the things depended on by the other. The result* of merging two dependent things depends respectively on the results* of the mergers it forces. For example, if two lines are merged, the resultant line must refer to only two end points, the results of merging the pairs of end points of the

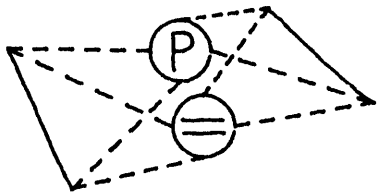
---

* The "result" of a merger is a single thing of the same type as the merged things.

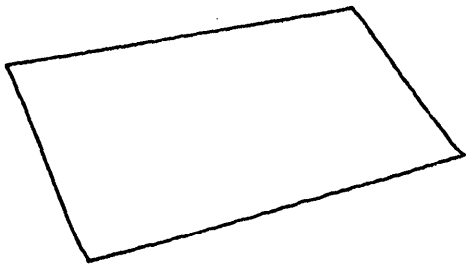
original lines. All lines which terminated on any of the four original end points now terminate on the appropriate one of the remaining pair. More important and useful, all constraints which applied to any of the four original end points now apply to the appropriate one of the remaining pair. This makes it possible to speak of line segments as being parallel even though (because line segments contain no numerical information to be constrained) the parallelism constraint must apply to their end points and not to the line segments themselves. If we wish to make two lines both parallel and equal in length, the steps outlined in Figure 6.1 make it possible. More obscure relationships between dependent things may as easily be defined and applied. For example, constraint complexes can be defined to make line segments be collinear, to make a line be tangent to a circle, or to make the values represented by two sets of digits be equal.

INSTANCES

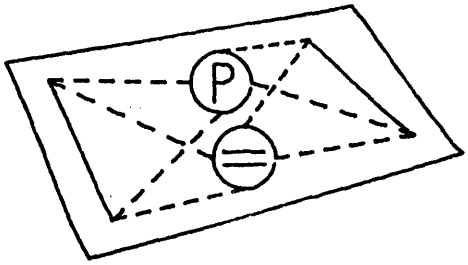
The most powerful tool provided in the Sketchpad system for creating large complex drawings quickly and easily is the instance. Instances are recursively expanded so that instances may contain other instances to give an exponential growth of picture produced with respect to effort expended. Instances may have attachment points and therefore may connect points topologically much as line segments do. For example, an instance of a resistor may connect two points both electrically and geometrically on the drawing. An instance also has the properties of a four component variable: numbers are stored in each instance block to indicate where, how big, and in what rotation that instance is to appear on the picture. It took some time to reconcile the topological properties of instances with their properties as variables.



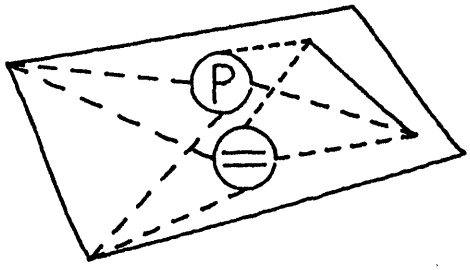
A. OPERATION DEFINITION



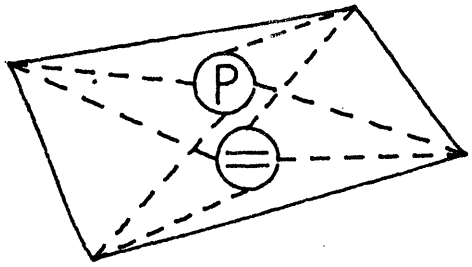
B. PICTURE TO CONSTRAIN



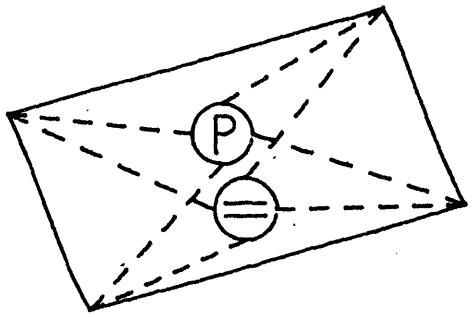
C. DEFINITION COPIED



D. FIRST LINE MERGED



E. SECOND LINE MERGED



F. CONSTRAINTS SATISFIED

FIGURE 6.1. APPLYING TWO CONSTRAINTS INDIRECTLY TO TWO LINES

P PARALLELISM = EQUAL LENGTH

The block of registers which represents an instance is remarkably small considering that it may generate a display of any complexity. For the purposes of display, the instance block makes reference to a picture by means of its chicken in a ring which ties a picture to all its instances. The instance will appear on the display as a figure geometrically similar to the picture of which it is an instance but at a location, size, and rotation indicated by the four numbers which constitute the "value" of the instance. An important omission as this is written is the ability to make mirror images. Right and left handed figures must now be treated separately, whereas the instance should indicate whether a right or left handed version of the master is to be shown.

INSTANCES AS VARIABLES

The four numbers which specify the size, rotation, and location of the instance are considered numerically as a four dimensional vector. In certain computations, the value of a variable is changed "as little as possible" if there is no need to change it further. The distance measured in the case of instances is the square root of the sum of the squares of the four components. For this reason, and for simplicity in the use of the fixed point arithmetic of the TX-2, it is important that the four numbers used to represent the vector be of about the same order of magnitude. The particular numbers chosen are the coordinates of the center of the instance and the actual size of the instance as it appears on the drawing times the sine and cosine of the rotation angle involved. In a typical drawing these four numbers have reasonably similar ranges of variation.

In our early work we attempted to use the position and the sine and cosine of the rotation angle times the reduction in size from the master picture in order to avoid the normalization of master picture size implicit in the above paragraph. This not only prevented having instances larger than their masters because of the fixed point arithmetic, but also made distance in the four dimensional space meaningless. No attempt was ever made to use the size and rotation numbers independently.

The transformations of coordinates represented by the above paragraphs are:

Poor

$$\begin{bmatrix} x_d \\ y_d \end{bmatrix} = \begin{bmatrix} i_1 & i_2 \\ -i_2 & i_1 \end{bmatrix} \begin{bmatrix} x_m \\ y_m \end{bmatrix} + \begin{bmatrix} i_3 \\ i_4 \end{bmatrix} \qquad (6\text{-}1)$$

Better

$$\begin{bmatrix} x_d \\ y_d \end{bmatrix} = \begin{bmatrix} i_1 & i_2 \\ -i_2 & i_1 \end{bmatrix} \begin{bmatrix} x_m/s_m \\ y_m/s_m \end{bmatrix} + \begin{bmatrix} i_3 \\ i_4 \end{bmatrix} \qquad (6\text{-}2)$$

where:

x_d, y_d = Display location in page coordinates.

x_m, y_m = Master location in page coordinates.

s_m = Size of master picture in page coordinates.

$i_1 \,..\, i_4$ = 4 vector in instance, $-1 < i_i < +1$.

RECURSIVE DISPLAY OF INSTANCES

In displaying an instance of a picture, reference must be made to the picture itself to find out what picture parts are to be shown. The picture referred to may contain instances, however, requiring further reference, and so on until a picture is found which contains no instances. A recursive program performs this function. At each stage in the recursion, any picture parts displayed must be relocated so that they will appear at the correct position, size and rotation on the display. Thus, at each stage of the recursion, some transformation of the form of Equation (6-2) is applied to all picture parts before displaying them. If an instance is encountered, the transformation represented by its value must be adjoined to the existing transformation for display of parts within it. When the expansion of an instance within an instance is finished, the transformation must be restored for continuation at the higher level.

To avoid the difficulties of taking an inverse transformation, the old transformation is saved in registers provided for that purpose in the picture block of the picture being expanded. Thus, the current transformation is stored in program registers and is being used, whereas the previous transformation is saved in the picture block currently being expanded. The push down list is provided also by indicating in the picture block being expanded the particular instance thereof which is responsible for this expansion of the picture. The first picture to be displayed starts with no transformation at all. Thus, if it contains itself as an instance, one recursion is possible, saving the old transformation in the picture block and saving the address of the instance responsible for the expansion in the picture block as well. Subsequent recursions will be prevented, however, because no instance is expanded

if the picture of which it is an instance already belongs on the push down list. It would be possible to expand such circular instances further by providing some suitable termination condition such as reaching a level too small to show on the display. However, since the instances might get larger rather than smaller, termination conditions are far from simple.

ATTACHERS AND INSTANCES

Many symbols used must be integrated into the rest of the drawing by attaching lines to the symbols at appropriate points, or by attaching the symbols directly to each other as if by zero length lines. For example, circuit symbols must be wired up, geometric patterns made by fitting shapes together, or mechanisms composed of links tied together appropriately. An instance may have any number of tie points, and, conversely, a point may serve as tie for any number of instances.

An "instance-point" constraint block is used to relate an instance to each of its tie points. An instance-point constraint is satisfied only when the point bears the same relationship to the instance that a point in the master picture for that instance bears to the master picture coordinate system. Instance-point constraints are treated as a special case when an instance is moved so that tie points always move with their instance, and lines terminating on the tie points move as well. Each instance-point constraint makes reference to both the instance and its tie points by means of chickens.

To use a point as an attacher of an instance, the point must be designated as an attacher in the master drawing of the instance. For example, when one first draws a resistor, the ends of the resistor must

be designated as attachers if wiring is to be attached. When an instance is created by pressing the "instance" button, toggle switches tell what picture the instance is to refer to. Along with the instance element are created a point and an instance-point constraint for each attacher. These points are bonifide points in the object picture but are not automatically attachers of the object picture. If they are to be used as attachers when the object picture is instanced, they must be designated anew. Thus of the three attachers of a transistor it is possible to select one or two to be the attachers of a flip-flop.

The entire internal structure of the instance is suppressed as far as the light pen is concerned except for the attachers. Thus even on a dense circuit drawing it is possible to connect elements with ease because at the highest level of instance only the designated attachers will hold the attention of the light pen program. Usually there are only a few attachers for each block no matter how complicated internally, and so it is generally obvious which one to use.

RECURSIVE MOVING

At first only variables could be moved. Moving a variable means to change somehow the numbers stored as the components of the variable, usually to make the display for the variable follow light pen motions. A moving point, for example, will be firmly attached to the pseudo pen position, while a moving piece of text faithfully follows light pen displacements so that the part of the text which was under the pen when the "move" button was pressed remains under the pen. For variables with more than two components, moving is partly controlled by the pen and partly by knobs. For example, the moving text can be made larger or

rotated by two of the knobs.

The advent of the recursive merging and the definition copying functions made it clear that one should be able to move anything regardless of whether or not it is variable. To move a non-variable, a recursive process is used to find whatever variables may be basic to the thing being moved. For example, if a line is to be moved, the end points on which it depends must be moved. All objects which are being moved are put in a ring whose hen is in the MOVINGS generic block. The object actually attached to the light pen is first in the ring. Upon termination only this first object in the MOVINGS ring may be merged with other objects.

The numerical operation of moving is accomplished by the standard transformation procedure. The small transformation due to light pen position change and knob rotation since the last program iteration is converted to the form of Equation (6-2) and placed in the standard location. Each object in the MOVINGS ring is transformed by it. The generic block for each type of object, of course, contains the subroutine to apply the transformation to such objects. The generic block for lines, for example, indicates that no transformation need be applied to the line because it contains no numerical values and will automatically be moved when its end points are moved.

Moving objects must be invisible to the light pen. Since the light pen aims at anything within its field of view, it would otherwise aim at a moving object and a jerky motion would result. Motion would only happen when the pen's field of view passed beyond the object being moved. Moreover, the display for moving objects must be recomputed regularly for the benefit of the human user, but the unmoving background need not be recomputed. The display spot coordinates for objects being recomputed

is placed last in the display file, above (in higher numbered registers) the fixed background display so that it may be recomputed without disturbing the rest of the display file. The light pen program rejects any spots seen by the pen which come from these high display file locations. Needless to say, the entire display file must be recomputed once to eliminate the former traces of the newly moving objects.

Chapter VII

BUILDING A DRAWING, THE COPY FUNCTION

As experimentation with drawing systems for the computer progressed, the basic drawing operations evolved into their present form. At the outset, the very general picture and relationship defining capability of the copy and recursive merging functions were unknown and so considerable power had to be built directly into the system. Now, of course, it would be possible to use much simpler atomic operations to draw simple pictures embodying many of the notions now treated as atomic.

DRAWING VS. MOVING

An idea that was difficult for the author to grasp was that there is no state of the system that can be called "drawing." Conventionally, of course, drawing is an active process which leaves a trail of carbon on the paper. With a computer sketch, however, any line segment is straight and can be relocated by moving one or both of its end points. In particular, when the button "draw" is pressed, a new line segment and two new end points are set up in storage, and one of the line's end points is left attached to the light pen so that subsequent pen motions will move the point. The state of the system is then no different from its state whenever a point is being moved.

Similarly, to draw a circle, one creates a center point when the button "circle center" is pressed, and creates in the ring structure a circle block and its start and end points when the button "draw" is pressed with a circle center defined. The end point of the circle arc

is left attached to the light pen to move with subsequent pen motions. Since the start and end points of a circle arc should be equidistant from its center point, an equal distance constraint is created along with the circle but could be subsequently deleted without deleting the circle.

ATOMIC OPERATIONS

In general, when creating new points to serve as the start of line segments and circle arcs or centers for circle arcs, an existing point is used if the pen is aimed at one when the new point would be generated. Thus, if one aims at the end of an existing line segment and presses "draw" the new line segment will use the existing point rather than setting up another point which has the same coordinates. Later motion of this point will move both lines attached to it; the ring structure storage reflects the intended topology of the drawing. Similarly, if one is moving a point and gives a termination signal while aiming at another point, these two points will be merged, again reflecting the intended drawing topology.

We have seen that a constraint is set up to indicate that the start and end points of a circle arc should be equidistant from its center whenever a new circle arc is drawn. Similarly, constraints to indicate that a point should lie on a line or circle are automatically set up if a point is either created while the pen is pointing to the line or circle or moved onto the line or circle. The constraints, of course, do not apply to the line or circle itself but to the points on which it depends. If the light pen is aimed at the intersection of line segments,

two "point-on-line" constraints will be set up for a point created or left there, one for each intersecting line. Three or more line segments may be forced to pass through a single point by moving that point onto them successively to set up the appropriate constraints. Constraint satisfaction will then move the lines so that all of them pass through the point. In order to avoid cluttering up the ring structure with redundant constraints, the point-on-line and point-on-circle constraints are set up only if the point is not already so constrained.

GENERALIZATION OF ATOMIC OPERATIONS

The atomic operations described above make it possible to create in the ring structure new picture components and relate them topologically. The atomic operations are, of course, limited to creating points, lines, circles, point-on-line and point-on-circle constraints. (The point-on-circle constraint is the same type as used to keep the circle's start and end points equidistant from its center.) Since implementation of the copy function it has become possible to create <u>any</u> combination of picture parts and constraints in the ring structure. The recursive merging function makes it possible to relate this set of picture parts to any existing parts. For example, if a line segment and its two end points are copied into the object picture, the action of the "draw" button may be exactly duplicated in every respect. Along with the copied line, however, one might copy as well a constraint to make the line horizontal, or two constraints to make it both horizontal and three inches long, or any other variation one cares to put into the ring structure to be copied.

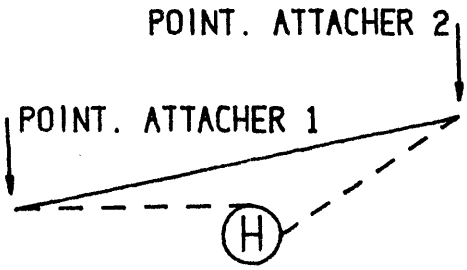
When one draws a definition picture to be copied, certain portions of it to be used in relating it to other object picture parts are designated as "attachers". Anything at all may be designated: for example, points, lines, circles, text, even constraints! The rules used for combining points when the "draw" button is pressed are generalized so that:

> For copying a picture, the last-designated attacher is left moving with the light pen. The next-to-last-designated attacher is recursively merged with whatever object the pen is aimed at when the copying occurs, if that object is of like type. Previously designated attachers are recursively merged with previously designated object picture parts, if of like type, until either the supply of designated attachers or the supply of designated object picture parts is exhausted. The last-designated attacher may be recursively merged with any other object of like type when the termination flick is given.

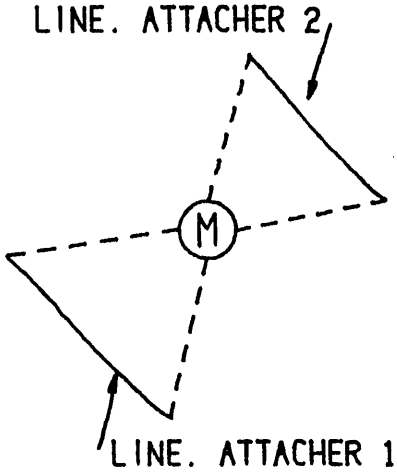
Normally only two designated attachers are used because it is hard to keep track of additional ones. The order in which attachers are designated is important because it is in this order that they will be treated. If a mistake is made in ordering the attachers, redesignation of an attacher puts it last in the order. As this is written there is no way to undesignate an attacher, except by deleting it, an oversight which should be corrected.

If the definition picture to be copied consists of a line segment with end points as attachers and a horizontal constraint between the end points, as shown in Figure 7.1A, pressing the "copy" button will appear to the user exactly like pressing the "draw" button. One end point of the line will be left behind and one will follow the light pen. Subsequent constraint satisfaction will, however, make the line horizontal.

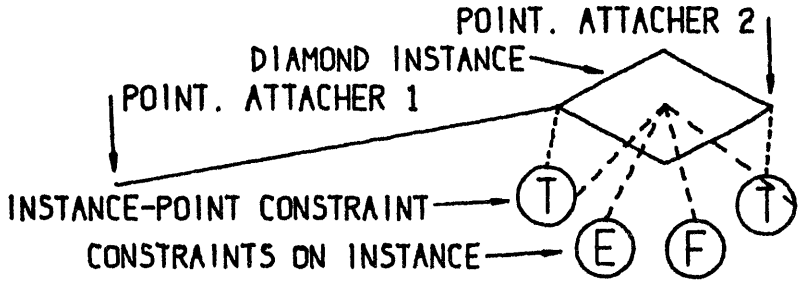
If the definition picture consists of two line segments, their four end points, and a constraint on the points which makes the lines equal



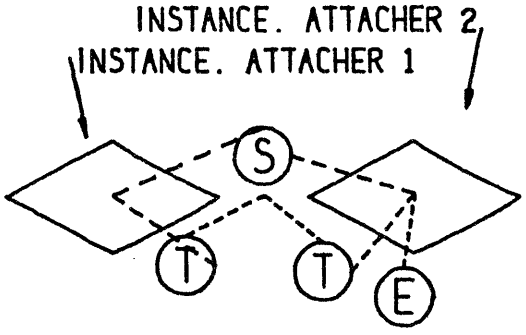
A. HORIZONTAL LINE



B. EQUAL LENGTH LINES



C. PARTLY FLEXIBLE ARROW



D. PRE-JOINED INSTANCES

FIGURE 7.1. DEFINITIONS TO COPY

in length, with the two lines designated as attachers as shown in Figure 7.1B, copying enables the user to make any two lines equal in length. If the pen is aimed at a line when "copy" is pushed, the first of the two copied lines merges with it, (taking its position and never actually being seen). The other copied line is left moving with the light pen and will merge with whatever other line the pen is aimed at when termination occurs. Since merging is recursive, the copied equal-length constraint will apply to the desired pair of object picture lines. If no lines are aimed at, of course, the copied picture parts are seen at once with the scale factor so reduced that the entire copied picture takes up about 1/16 of the display area.

If the picture to be copied consists of the erect constraint and the full size constraint, both applying to a single dummy variable which is the attacher, copying produces a useful constraint complex attached to the pen for subsequent application to any desired instance. With only one attacher, the instance constrained is the one the pen is aimed at when termination occurs.

COPYING INSTANCES

As we saw in Chapter VI the internal structure of an instance is entirely fixed. The internal structure of a copy, however, is entirely variable. An instance always retains its identity as a single part of the drawing; one can only delete an entire instance. Once a definition picture is copied, however, the copy loses all identity as a unit; individual parts of it may be deleted at will.

One might expect that there was intermediate ground between the fixed-internal-structure instance and the loose-internal-structure copy.

One might wish to produce a collection of picture parts, some of which were fixed internally and some of which were not. The entire range of variation between the instance and the copy can be constructed by copying instances.

For example, the arrow shown in Figure 7.1C can be copied into an object picture to result in a fixed-internal-structure diamond arrowhead with a flexible tail. As the definition in Figure 7.1C is set up, drawing diamond-arrowheaded lines is just like drawing ordinary lines. One aims the light pen where the tail is to end, presses "copy" and moves off with an arrowhead following the pen. The diamond arrowhead in this case will remain horizontal.

Copying pre-joined instances can produce vast numbers of joined instances very easily. For example the definition in Figure 7.1D, when repetitively copied, will result in a row of joined, equal size diamonds. In this case the instances themselves are attachers. Although each press of the "copy" button copies two new instances into the object picture, one of these is merged with the last instance in the growing row. In the final row, therefore, each instance carries all the constraints which were applied to either of the instances in the definition. This is why only one of the instances in Figure 7.1D carries the erect constraint. Notice also that although the diamond is normally a two-attacher instance, each of the diamonds in Figure 7.1D carries only one attacher. The other has been deleted so that each instance in the final row of diamonds will obtain only one right and one left attacher, one from each of the copied instances.

THE MECHANICS OF COPYING

Needless to say, when a piece of ring structure is copied the definition picture used is not destroyed; the copying procedure reproduces its ring structure elsewhere in memory. However, the reproduction is not just a duplication of the numbers in some registers. The parts of the definition drawing to be copied may be topologically related, and the parts of the copy must be related to each other in the same way rather than to the parts of the master. Worse yet, some parts of the definition may be related to things which are not being copied. For example, an instance is related to the master picture of which it is an instance, and the copy of the instance must be related to the same master picture, not to a copy of it.

To copy a picture, space to duplicate all the elements of the picture is allocated in the free registers at the end of the ring structure. Each of the new elements is tied into its appropriate generic block ring by its TYPE component. Each new element is placed in this ring adjacent to the element it is a copy of. That is, for each element in the master a duplicate element is set up adjacent to it in the generic ring for that type of element. Appropriate scaled values are given to copied variables. The various references in the definition elements are then examined to see whether they refer to things that have been copied. If they do, the corresponding components of the copied elements are made to refer to the appropriate copied elements. On the other hand, if a definition element refers to something which has not been copied, its copy refers to the same element that its definition does.

When the complete copy has been made, the copies of all but the last-designated of the attachers are recursively merged with the designated

portions of the object picture. The last-designated attacher is fastened to the light pen with the recursive moving function. The last-designated attacher may later on merge with another picture part.

Chapter VIII

CONSTRAINT SATISFACTION

The major feature which distinguishes a Sketchpad drawing from a paper and pencil drawing is the user's ability to specify to Sketchpad mathematical conditions on already drawn parts of his drawing which will be automatically satisfied by the computer to make the drawing take the exact shape desired. For example, to draw a square, any quadralateral is created by sloppy light pen manipulation, closure being assured by the pseudo light pen position and merging of points. The sides of this quadralateral may then be specified to be equal in length and any angle may be required to be a right angle. Given these conditions, the computer will complete a square. Given an additional specification, say the length of one side, the computer will create a square of the desired size.

The process of fixing up a drawing to meet new conditions applied to it after it is already partially complete is very much like the process a designer goes through in turning a basic idea into a finished design. As new requirements on the various parts of the design are thought of, small changes are made to the size or other properties of parts to meet the new conditions. By making Sketchpad able to find new values for variables which satisfy the conditions imposed, it is hoped that designers can be relieved of the need of much mathematical detail. The effort expended in making the definition of constraint types as general as possible was aimed at making design constraints as well as geometric constraints equally easy to add to the system. To date,

however, Sketchpad is more of a model of the design process than a complete designer's aid both because it is limited to two dimensions and because little advanced application has as yet been made of it.

The work on constraint satisfaction has been successful as far as it has been taken. The constraint definition and satisfaction programs generalize easily to three dimensions; in fact, constraint satisfaction for instances is even now treated as a four dimensional problem. The high speed maze solving technique for constraint satisfaction described below works well where constraints have been specified unredundantly. There is much room for improvement in the relaxation process and in making the "intelligent" generalizations that permit humans to capitalize on symmetry and eliminate redundancy.

DEFINITION OF A CONSTRAINT TYPE

Each constraint type is entered into the system as a generic block indicating the various properties of that particular constraint type. Generic blocks for constraints need not be given symbolic programming names since virtually no reference is made to particular constraint types in the program. The generic block tells how many variables are constrained, which of these variables may be changed in order to satisfy the constraint, how many degrees of freedom are removed from the constrained variables, and a code letter for human reference to this constraint type.

Any number of variables may be related by a constraint, but the display for constraints (see Chapter V) will be ambiguous if more than four variables are indicated, and so no constraints relate more than four variables. Of these variables, some may be referenced only.

The routine which satisfies the constraint by changing the values of some of the variables is forbidden to satisfy the constraint by changing a "for reference only" variable. For example, a constraint could be implemented which would make its first variable equal to its second by changing the first to match the second, but not the reverse. This kind of one-way constraint is useful because it speeds up the relaxation procedure by forcing re-evaluation of variables in a specified order. For example, the constraint which makes the value of a number equal to the change in length of a bridge beam, thus indicating the force carried by the beam, is one way. It would be pointless to have an erroneous value of the indicator affect in any way the relaxation procedure for the bridge. Again, the constraint which relates a point to an instance in such a way that the point maintains the same relationship to the instance that an original point in the master picture had to the master picture, uses the original point "for reference only" to discover just what the correct relationship is. Thus the end terminal on a resistor will always stay at the end of the resistor. It would be out of keeping with the fixed geometry nature of instances to have the internal details of the instance changed to make it fit into some awkward position.

The one-way type constraint, however, can lead to instabilities in the constraint satisfaction procedure. For example, if two scalars were each specified to be twice the value of the other, with reference only made to the smaller,

$$\begin{aligned} A &\rightarrow 2B \\ B &\rightarrow 2A, \end{aligned} \qquad (8\text{-}1)$$

both variables would grow without bound, assuming, each iteration, values four times as big as before. If, however, a similar condition were set up with normal two-way constraints, the values of the variables would approach zero, a correct and stable result. Since the number of one-way constraints is small and they are designed for and used in special applications only, very little instability trouble of this kind has been observed. Future users who add one-way constraints, however, are warned to be cautious of the instabilities which may result.

NUMERICAL DEFINITION OF CONSTRAINTS

After the first stumblings of trying to define a constraint type in terms of the equations of lines along which the constrained variables should lie to satisfy the constraint, the numerical definition of constraints directly in terms of an error was devised. By using an error definition and considering the square of the error as an energy, one not only reflects directly the intent of the relaxation process, but also makes it easy to write the defining subroutines for new constraint types.

The defining subroutine for a constraint type is a subroutine which will compute, for the existing values of the variables of a particular constraint of that type, the error introduced into the system by that particular constraint. For example, the defining subroutine for making points have the same x coordinate (to make a line between them vertical) computes the difference in their x coordinates. What could be simpler? The computed error is a scalar which the constraint satisfaction routine will attempt to reduce to zero by manipulation of the constrained variables. The computation of the error may be non-linear or time dependent, or it may involve parameters not a part of the drawing such

as the setting of toggle switches, etc. The flexibility of computation subroutines for defining constraints is tremendous.

In order to avoid overflow difficulties, the partial derivative of the error with respect to the value of any of the components of a constrained variable must be less than two. In order to make the constraints work well together, it is necessary that they be balanced, that is that the partial derivative of error with respect to displacement be nearly equal for all constraint types. I have arbitrarily tried to make the error subroutines compute an error about proportional to the distance by which a variable is removed from its proper position. In other words, many of the existing constraint computation subroutines make the partial derivative about unity.

LINEARIZATION OF CONSTRAINTS

The method of finding the least mean squares fit to a group of constraints described below requires that a linear equation be given for each constraint. To find the linear equation which best approximates the possibly non-linear constraint for the present values of the variables, the error computed by the subroutine is noted for several slightly different values of the variables. The equation,

$$\sum \frac{\Delta E}{\Delta x_i} (x_i - x_{io}) = -Eo, \qquad (8\text{-}2)$$

where x_i are the components of the variable, E is the computed error, and subscript o denotes intial value, is used as the linear best fit. Actually, the coefficients computed are 1/2 the values shown in equation (8-2) to permit error to be equal to displacement without generating overflow.

Some constraints may remove more than one degree of freedom from the variables constrained. For example, the constraint which locates one thing exactly mid-way between two others removes two degrees of freedom. Such constraints must have as many error computation subroutines as there are degrees of freedom lost, since each subroutine results in a single linear equation. A subroutine which computes the distance from a variable to its correct location without regard to the number of degrees of freedom being removed will cause erratic results. A correct subroutine pair for constraining one thing to lie between two others computes both how far out of line the center thing is and, separately, 1/2 the difference in the distances from the center object to the two outer ones (1/2 is put in to meet the maximum derivative requirement).

THE RELAXATION METHOD

When the one pass method of satisfying constraints to be described later on fails, the Sketchpad system falls back on the reliable but slow method of relaxation to reduce the errors indicated by the various computation subroutines to smaller and smaller values. For simple constructions such as the hexagon illustrated in Figure 1.5, page 15 the relaxation procedure is sufficiently fast to be useful. However, for complex systems of variables, especially directly connected instances, relaxation is unacceptably slow. Fortunately, it is for just such directly connected instances that the one pass method shows the most striking success.

The relaxation method of satisfying conditions is as follows:

> Choose a variable. Re-evaluate it to reduce the total error introduced by all constraints in the system. Choose another variable and repeat.

Note that since each step makes some net reduction of total error, there will be monotonic decrease of error and thus stability is assured. Since re-evaluating a variable will change only the error introduced by the constraints which apply to that variable, only the changes in the errors introduced by these constraints need be considered. Other variables and therefore the errors of constraints applying only to them will remain constant. Sketchpad's ring structure makes it easy to consider all constraints applying to a particular variable since all such constraints are collected together in a ring whose "hen" is in the variable.

It is important in the relaxation method that, at each step, the very latest computed values of all variables be used for error computations. From the point of view of the program, this means that only one value for each variable need be stored, each being updated in turn. Former values not only may, but must be discarded. It is also important that the change in error obtained by completely satisfying a constraint by moving one of its variables be identical to to the change in error to be obtained by completely satisfying it by moving another of its variables. The error computing subroutine definition for a constraint computes the same error for a constraint no matter which of its variables is to be moved. My original instability troubles with constraint satisfaction came from insufficient care in meeting this condition.

LEAST MEAN SQUARES FIT TO LINEARIZED CONSTRAINTS

In implementing the relaxation method above, it is important to be able to find quickly a new value for a variable which reduces the total error introduced by the constraints on that variable. In particular, the linearized form of the constraints results in a set of linear equations

for the variable each of which must be met as closely as possible. Unfortunately, there may be any number of linear equations applying to a particular variable and these may be either independent but incomplete, independent and complete, or redundant and overdefining. A general arithmetic macro, SOLVE, for finding the best value for a set of equations has been devised.

SOLVE converts the given equations into an independent set of equations whose solution will be a point of minimum mean squared error for the original set. It is not always possible to solve the independent set of equations uniquely, and if it is not, SOLVE finds that solution which results in the minimum change from the existing value of the variable. The mathematical discussion pertinent to SOLVE is given in Appendix F. I am indebted to Lawrence G. Roberts for providing me with the basic SOLVE program.

Seen from the outside, then, the linearization program and SOLVE make it possible for Sketchpad to find a new value for any variable to more closely meet the conditions indicated by constraints. Repeated application of these programs to variables, in sequence, implements the relaxation process. Application of these programs to selected variables to detect the number and degree of independence of constraints is used as an important part of the one pass constraint satisfaction method.

ONE PASS METHOD

Sketchpad can often find an order in which the variables of a drawing may be re-evaluated to completely satisfy all the conditions on them in just one pass. For the cases in which the one pass method works, it is far better than relaxation: it gives correct answers at once; relaxation

may not give a correct solution in any finite time. Sketchpad can find an order in which to re-evaluate the variables of a drawing for most of the common geometric constructions. Ordering is also found easily for the mechanical linkages illustrated in the last chapter. Ordering cannot be found for the bridge truss problems illustrated in the last chapter.

The way in which the one pass method works is simple in principle and was easy to implement as soon as the nuances of the ring structure manipulations were understood. To visualize the one pass method, consider the variables of the drawing as places, and the constraints relating variables as passages through which one might pass from one variable to another. Variables are adjacent to each other in the maze formed by the constraints if there is a single constraint which constrains them both. Variables are totally unrelated if there is no path through the constraints by which one could pass from one to the other.

Suppose that some variable can be found which has so few constraints applying to it that it can be re-evaluated to completely satisfy all of them. Such a variable we shall call a "free" variable. As soon as a variable is recognized as free, the constraints which apply to it are removed from further consideration, because the free variable can be used to satisfy them. Removing these constraints, however, may make adjacent variables free. Recognition of these new variables as free removes further constraints from consideration and may make other adjacent variables free, and so on throughout the maze of constraints. The manner in which freedom spreads is much like the method used in Moore's algorithm[7] to find the shortest path through a maze. Having found that a collection of variables is free, Sketchpad will re-evaluate

them in the reverse order, saving the first-found free variable until last. In re-evaluating any particular free variable Sketchpad uses only those constraints which were present when that variable was found to be free.

In the ring structure representation of the drawing all variables found to be free are placed in a special ring called the FREEDOMS ring. (Note that the FREE ring is used for empty spaces in storage and has nothing to do with freedom in the present sense.) Each variable placed on the FREEDOMS ring has associated with it, by extra ties, those constraints which it will be used to satisfy. In what order variables should appear in the FREEDOMS ring need only be computed when the constraint conditions change. For a given set of conditions the same ordering will serve for finding many satisfactory values. For example, as part of a linkage is moved with the light pen, the ordering first set up for the linkage serves until the conditions change.

Chapter IX

EXAMPLES AND CONCLUSIONS

In the first chapter we saw, as an introduction to the system, some simple examples of Sketchpad drawings. In the body of this report we have seen many drawings, all of which, except the drawing of the light pen, Figure 4.2, were drawn with Sketchpad especially to be included here. In this chapter we shall consider a wider variety of examples in somewhat more detail. The examples in this chapter were all taken from the library tape and thus serve to illustrate not only how the Sketchpad system can be used, but also how it actually has been used so far.

We conclude from these examples that Sketchpad drawings can bring invaluable understanding to a user. For drawings where motion of the drawing, or analysis of a drawn problem is of value to the user, Sketchpad excells. For highly repetitive drawings or drawings where accuracy is required, Sketchpad is sufficiently faster than conventional techniques to be worthwhile. For drawings which merely communicate with shops, it is probably better to use conventional paper and pencil.

PATTERNS

The instance facility outlined in Chapter I enables one to draw any symbol and duplicate its appearance anywhere on an object drawing at the push of a button. The symbols drawn can include other symbols and so on to any desired depth. This makes it possible to generate huge numbers of identical shapes; if at each stage two of the previous symbols are combined to double the number of basic shapes present, in twenty steps one million objects are produced.

The hexagonal pattern we saw in Figure 1.1, p. 10, is one example of a highly repetitive drawing. The hexagonal pattern was first drawn in response to a request for hexagonal "graph" paper. About 900 hexagons were plotted on a single 30 x 30 inch plotter page. It took about one half hour to generate the 900 hexagons, including the time taken to figure out how to do it. Plotting them takes about 25 minutes. The drafting department estimated it would take them two days to produce a similar pattern.

The instance facility also made it easy to produce long lengths of the zig-zag pattern shown in Figure 9.1. As the figure shows, a single "zig" was duplicated in multiples of five and three, etc. Five hundred zigs were generated in a single row. Four such rows were plotted one half inch apart to be used for producing a printed circuit delay line. Total time taken was about 45 minutes for constructing the figure and about 15 minutes to plot it.

In both the zig-zag pattern of Figure 9.1 and in the hexagonal pattern of Figure 1.1 the various subpictures were fastened together by attachment points. In the hexagonal pattern, each corner of the basic hexagon was attached to the corners of adjacent hexagons. The position of any hexagon was then completely determined by the position of any other. In the zig-zag pattern of Figure 9.1, however, only a single attachment was made between adjacent zig-zags. Additional constraints were applied to each instance to keep them erect and of the same size.

A somewhat less repetitive pattern to be used for encoding the time in a digital clock is shown in Figure 9.2. Each cross in the figure marks the position of a hole. The holes will be placed so that a binary coded decimal (BCD) number will indicate the time.

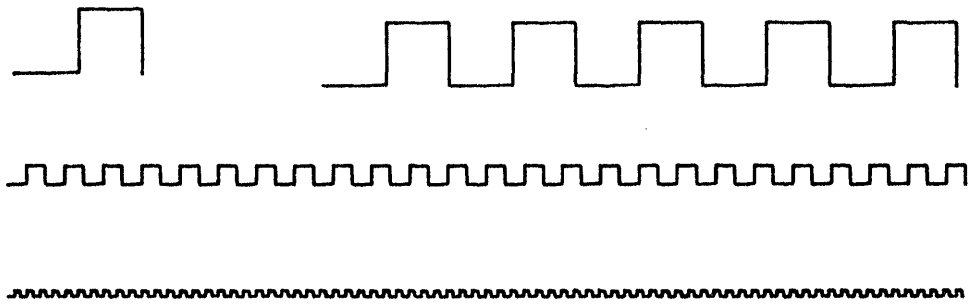


FIGURE 9.1.
ZIG-ZAG FOR DELAY LINE

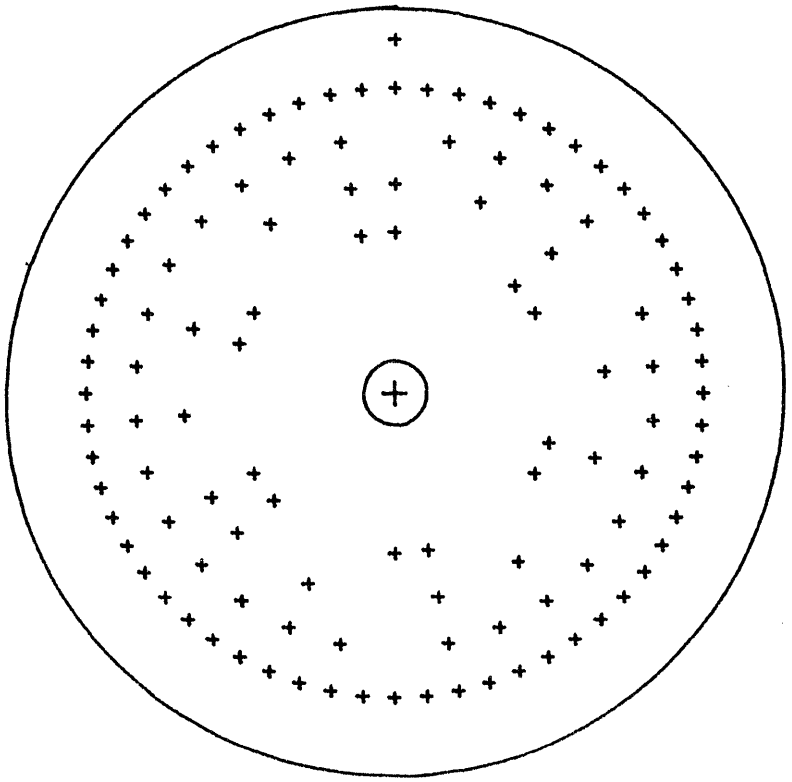


FIGURE 9.2.
BCD ENCODER FOR CLOCK

Sketchpad was first used in the BCD clock project to produce 60 radial lines at equal 6° spacing. To do this a single 6° wedge was produced by first trisecting a right angle to obtain a 30° wedge and then cutting the 30° wedge into five parts. The relaxation procedure was used in each case to make three or five sketched-in chords equal in length. Making the 6° wedge took a brand new user less than one half hour including instruction time. The author has constructed other wedges as small as 1/128 of a circle in five minutes. All such wedges become a part of the library.

The 6° wedge has three attachment points. By attaching five of the wedges together, and then attaching three groups of five, a quadrant is constructed. Fitting together four quadrants gives a complete circle based entirely on the single 6° wedge. The advantage of constructing a full circle composed of 60 wedges is that any lines drawn in the original 6° wedge will appear 60 times around the circle with no further effort on the part of the user. Sixty radial lines were produced in this way.

Using the sixty radial lines plotted for him the BCD clock designer then marked with pencil approximately where the crosses should be placed to obtain BCD coding. Returning to Sketchpad we put a pattern of dots in the 6° wedge so that in the full circle, rings of dots appeared which could be aimed at with the light pen. It was then an easy matter to place a cross exactly on each of the desired dots. Total time for placing crosses was 20 minutes, most of which was spent trying to interpret the sketch.

LINKAGES

By far the most interesting application of Sketchpad so far has been drawing and moving linkages. We saw in Chapter I the straight line linkage of Peaucellier, Figure 1.6, p.20. The ability to draw and then move linkages opens up a new field of graphical manipulation that has never before been available. It is remarkable how even a simple linkage can generate complex motions. For example, the linkage of Figure 9.3 has only three moving parts. In this linkage a central ⊥ link is suspended between two links of different lengths. As the shorter link rotates, the longer one oscillates as can be seen in the multiple exposure. The ⊥ link is not shown in Figure 9.3 so that the motion of four points on the upright part of the ⊥ may be seen. These are the four curves at the top of the figure.

To make the three bar linkage, an instance shaped like the ⊥ was drawn and given 6 attachers, two at its joints with the other links and four at the places whose paths were to be observed. Connecting the ⊥ shaped subpicture onto a linkage composed of three lines with fixed length created the picture shown. The driving link was rotated by turning a knob below the scope. Total time to construct the linkage was five minutes, but over an hour was spent playing with it.

Sketchpad can make linkages that one would hardly think of constructing out of actual links and pins. For example, a Sketchpad sliding joint is ideal, whereas to actually build a sliding joint is relatively difficult. Again, it is possible to make two widely separated links be of equal length by applying an appropriate constraint, but to build such a linkage would be impossible.

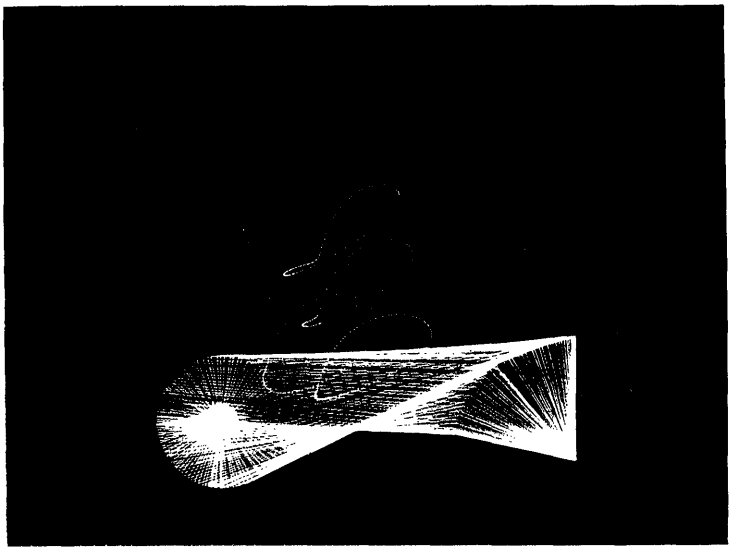


FIGURE 9.3. THREE BAR LINKAGE
The paths of four points on the central link are traced. This is a 15 second time exposure of a moving Sketchpad drawing.

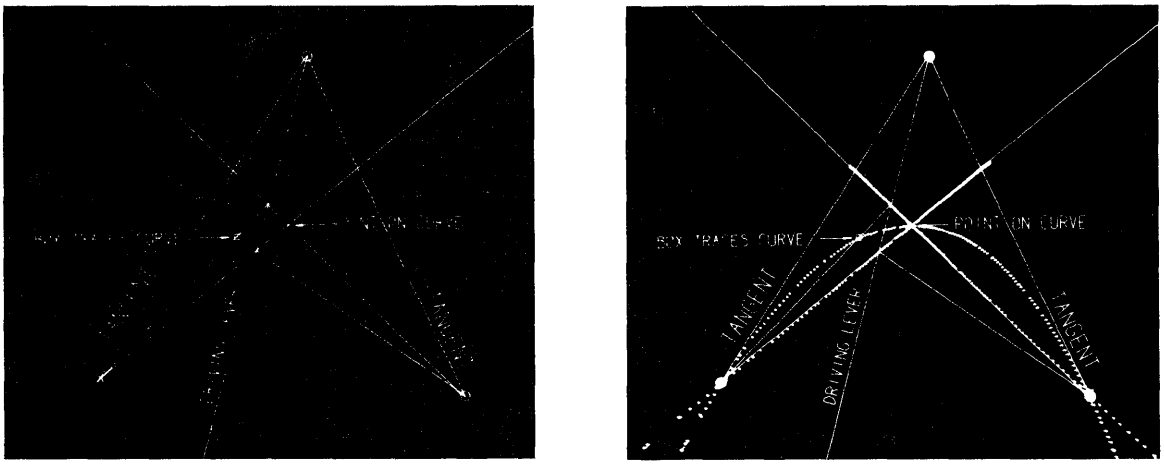


FIGURE 9.4. CONIC DRAWING LINKAGE
As the "driving lever" is moved, the point shown with a box around it traces a conic section. This conic can be seen in the time exposure.

A linkage that would be difficult to build physically is shown in Figure 9.4. This linkage is based on the complete quadrilateral. The three circled points and the two lines which extend out of the top of the picture to the right and left are fixed. Two moving lines are drawn from the lower circled points to the intersections of the long fixed lines with the driving lever. The intersection of these two moving lines (one must be extended) has a box around it. It can be shown theoretically that this linkage produces a conic section which passes through the place labeled "point on curve" and is tangent to the two lines marked "tangent." Figure 9.4B shows a time exposure of the moving point in many positions. The straight dotted lines are the paths of other, less interesting points. At first, this linkage was drawn and working in fifteen minutes. Since then we have rebuilt it time and again until now we can produce it from scratch in about three minutes.

DIMENSIONING OF DRAWINGS

It is important that a Sketchpad drawing be made in the correct size for many applications. For example, the BCD clock pattern shown in Figure 9.2 was plotted exactly 12 inches in diameter for the actual application. In fact, the precision of the plotter is such that its plotted output can be used directly as a layout in many cases. But the size of a drawing as seen on the computer display is variable. To make it possible to have an absolute scale in drawings, a constraint is provided which forces the value displayed by a set of digits to indicate the distance between two points on the drawing. The distance is indicated in thousandths of an inch for "full size" plotted output.

This distance indicating constraint is used to make the number in a dimension line. Many other constraints are used to make the arrowheads at the end of the line be "parallel" to the dimension line and to make enough space in the line for the dimension number. In some sense the dimension line is a complicated linkage; like a linkage it can be moved around while retaining its properties. For example, the arrowheads stay the same size even when the dimension line is made longer. A dimension line with small arrowheads is a part of the library. This line is suitable for dimensions of the order of a few inches. A three-four-five triangle dimensioned with this line is shown in Figure 9.5.

To produce the three-four-five triangle of Figure 9.5, three vertical and four horizontal line segments were made to be the same length. After erasing these lines, the three correctly positioned corners of the triangle were dimensioned. Putting in a dimension line is as easy as drawing any other line. One points to where one end is to be left, copies the definition of the dimension line by pressing the "copy" button, and then moves the light pen to where the other end of the dimension line is to be. The size of the three-four-five triangle was adjusted so that even dimensions appeared. At other sizes, of course, the ratio of the dimensions was correct but not so easy to recognize at a glance. Total time to produce dimensioned three-four-five triangle was three minutes, exclusive of time taken to produce the library version of the dimension line. The first dimension line took about fifteen minutes to construct, but that need never be repeated.

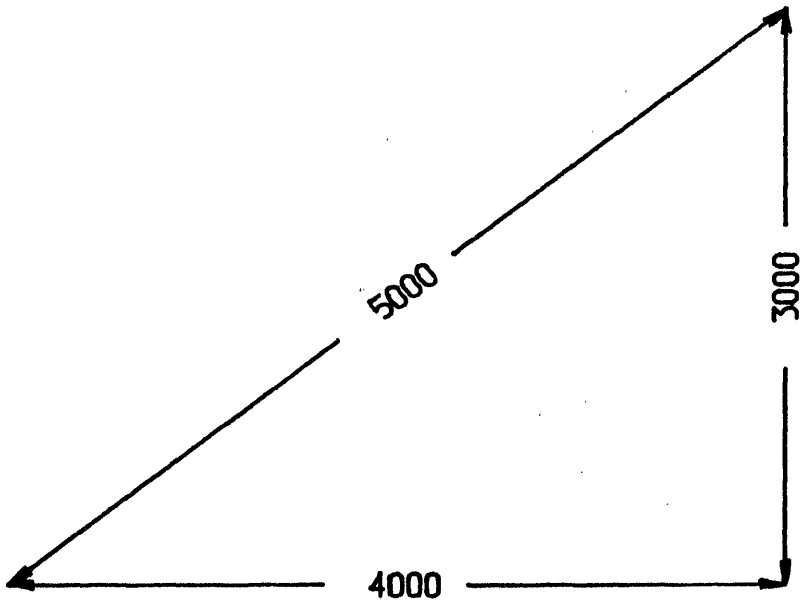


FIGURE 9.5. DIMENSION LINES

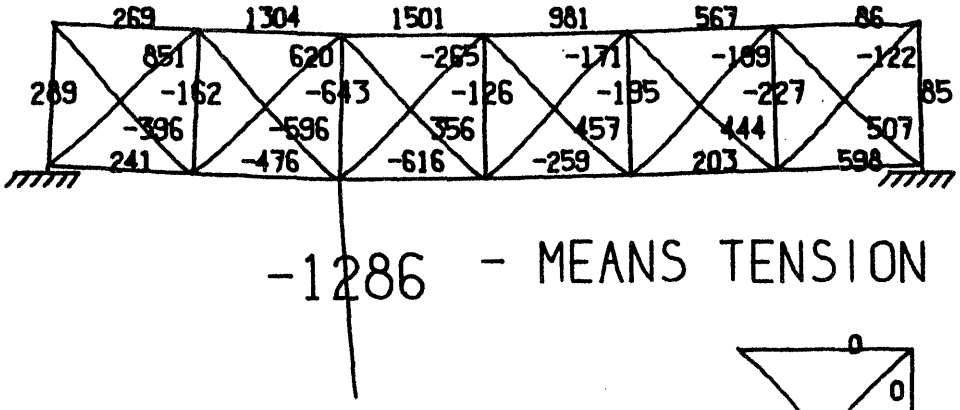


FIGURE 9.6.
TRUSS UNDER LOAD

BRIDGES

One of the largest untapped fields for application of Sketchpad is as an input program for other computation programs. The ability to place lines and circles graphically, when coupled with the ability to get accurately computed results pictorially displayed, should bring about a revolution in computer application. With Sketchpad we have a powerful graphical input tool. It happened that the relaxation analysis built into Sketchpad is exactly the kind of analysis used for many engineering problems. By using Sketchpad's relaxation procedure we were able to demonstrate analysis of the force distribution in the members of a pin connected truss. We do not claim that the analysis represented in the next series of illustrations is accurate to the last significant digit. What we do claim is that a graphical input coupled to some kind of computation which is in turn coupled to graphical output is a truly powerful tool for education and design.

In Figure 9.6 is shown a truss bridge supported at each end and loaded in the center. To draw this figure, one bay of the truss (shown below the bridge) was first drawn with enough constraints to make it geometrically accurate. These constraints were then deleted and each member was made to behave like a bridge beam. A bridge beam is constrained to maintain constant length, but any change in length is indicated by an associated number. Under the assumption that each bridge beam has a cross-sectional area proportional to its length, the numbers represent the forces in the beams. The basic bridge beam definition (consisting of two constraints and a number) may be copied and applied to any desired line in a bridge picture. Each desired bridge member was

changed from a line into a full bridge beam by pointing to it and pressing the "copy" button.

Using the bridge bay six times we construct the complete bridge. The loading line and the one missing end member are put in separately. The six-bay unloaded truss bridge is part of the library. It took less than ten minutes to draw completely. Applying a load where desired and attaching supports, one can observe the forces in the various members. It takes about 30 seconds for new force values to be computed. The bridge shown in Figure 9.6 has both outside lower corners fixed in position. Normally, of course, a bridge would be fixed only at one end and free to move sideways at the other end.

Having drawn a basic bridge shape, one can experiment with various loading conditions and supports to see what the effect of making minor modifications is. For example, an arch bridge is shown in Figure 9.7 supported both as a three hinged arch (two supports) and as a cantilever (four supports). For nearly identical loading conditions the distribution of forces is markedly different in these two cases.

ARTISTIC DRAWINGS

Sketchpad need not be applied only to engineering drawings. The ability to put motion into the drawings suggests that it would be exciting to try making cartoons. The capability of Sketchpad to store previously drawn information on magnetic tape means that every cartoon component ever drawn is available for future use. If the almost identical but slightly different frames that are required for making a motion picture cartoon could be produced semi-automatically, the entire Sketchpad system could justify itself economically in yet another way.

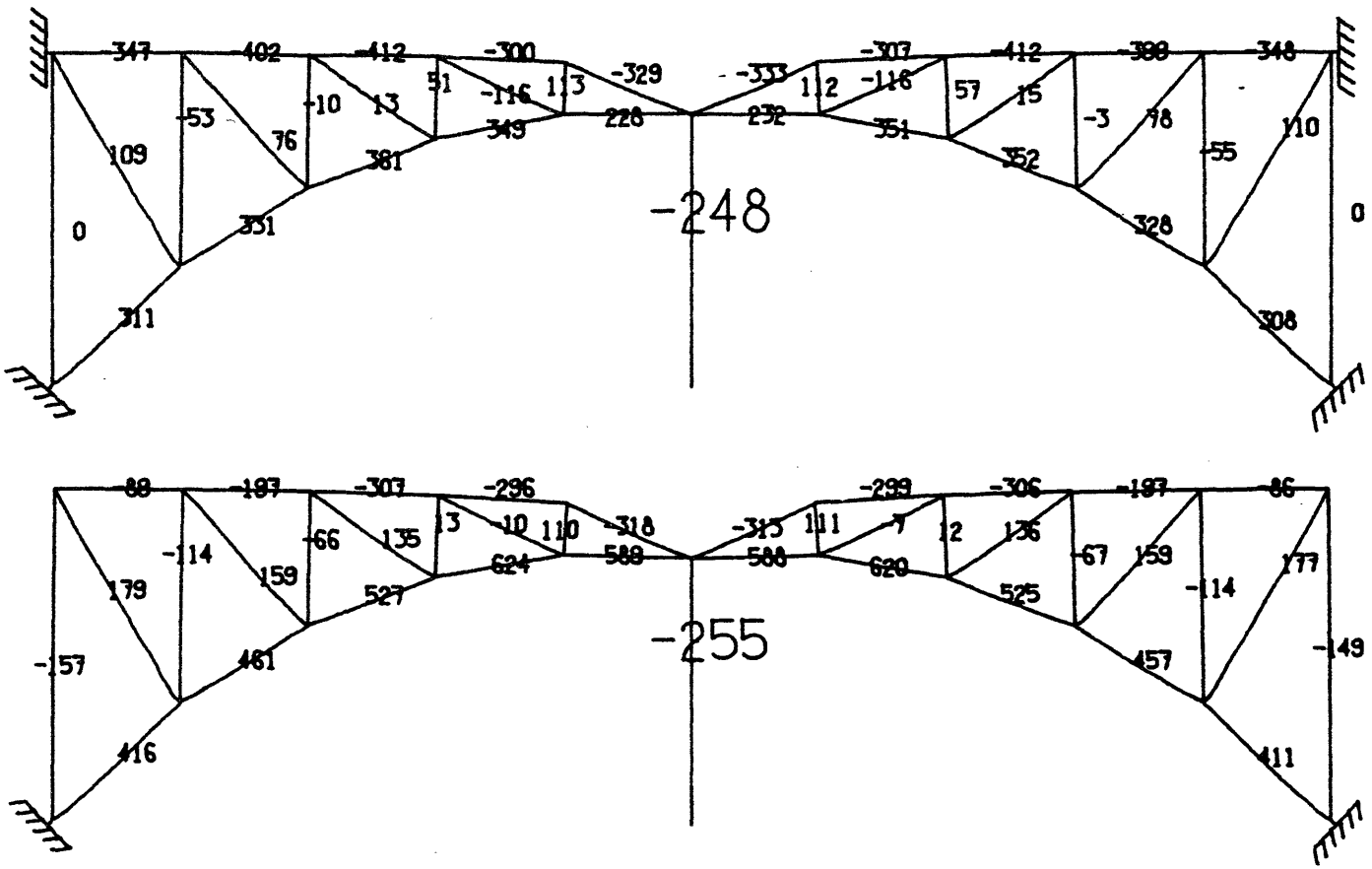


FIGURE 9.7. CANTILEVER AND ARCH BRIDGES

One way of cartooning is by substitution. For example, the girl "Nefertite" shown in Figure 9.8 can be made to wink by changing which of the three types of eyes is placed in position on her otherwise eyeless face. Doing this on the computer display has amused many visitors.

A second method of cartooning is by motion. A stick figure could be made to pedal a bicycle by appropriate application of constraints. Similarly, Nefertite's hair could be made to swing. This is the more usual form of cartooning seen in movies.

Aside from its economics as a teaching or amusement device, cartooning can bring the insights which are the prime value of Sketchpad drawings. The girl seen in Figure 9.9 was traced from a photograph into the Sketchpad system. The photograph was read into the computer by a facsimile machine used in another project[8] and shown in outline on the computer display. This outline was then traced with wax pencil on the display face. Later, with Sketchpad in the computer, the outline was made into a Sketchpad drawing by tracing the wax line with the light pen.

Once having the tracing on magnetic tape many things can be done with it. In particular, the eyes and mouth were erased to leave the featureless face which may also be seen in Figure 9.9. Returning to the tracing and erasing everything except the mouth and then everything except an eye we obtained features. In refitting the features to the blank face we discovered that, although the original girl was a sweet looking miss, an entirely different character appears if her mouth is made larger as in Figure 9.10. Using a computer to partially automate an artistic process has brought me, a non-artist, some understanding of

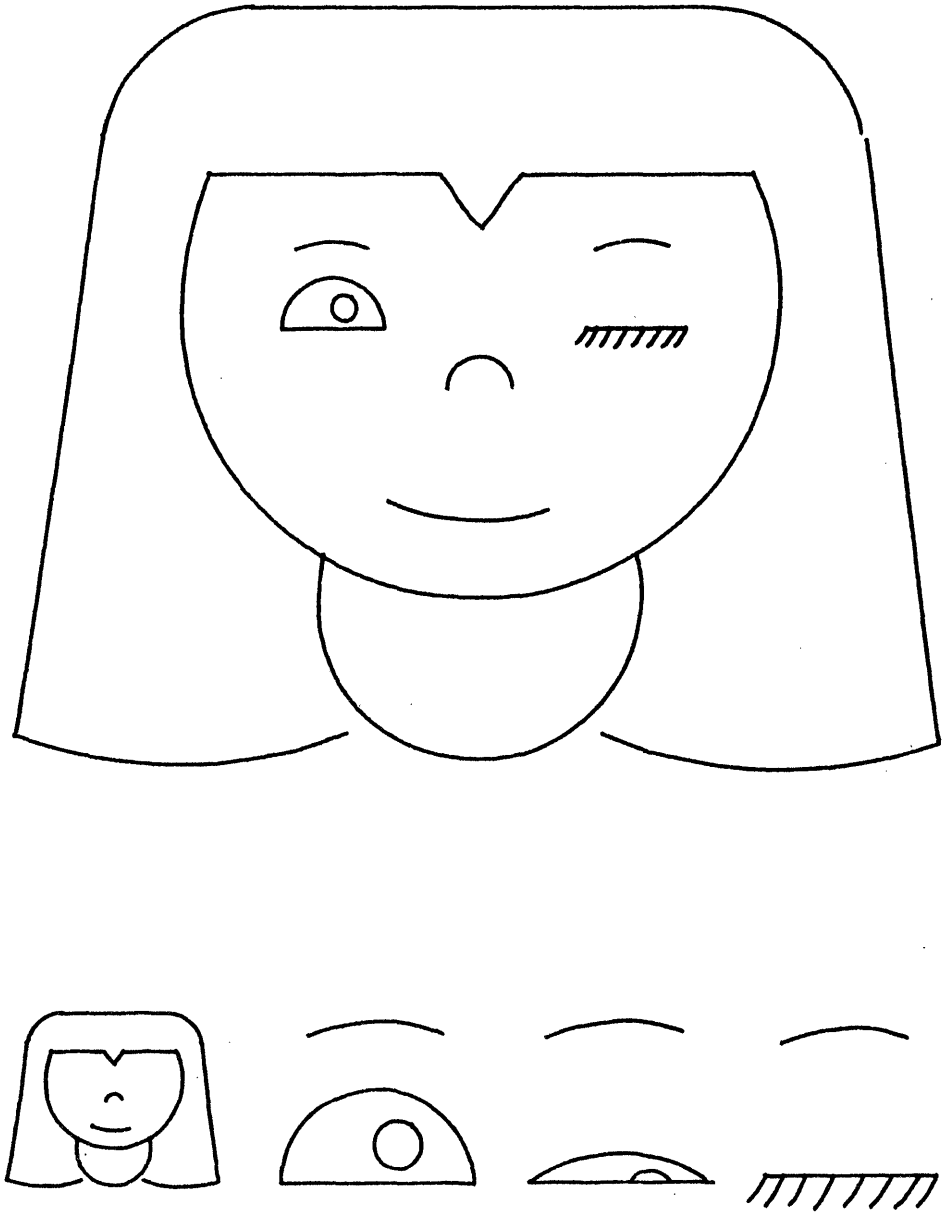


FIGURE 9.8.

WINKING GIRL AND COMPONENTS

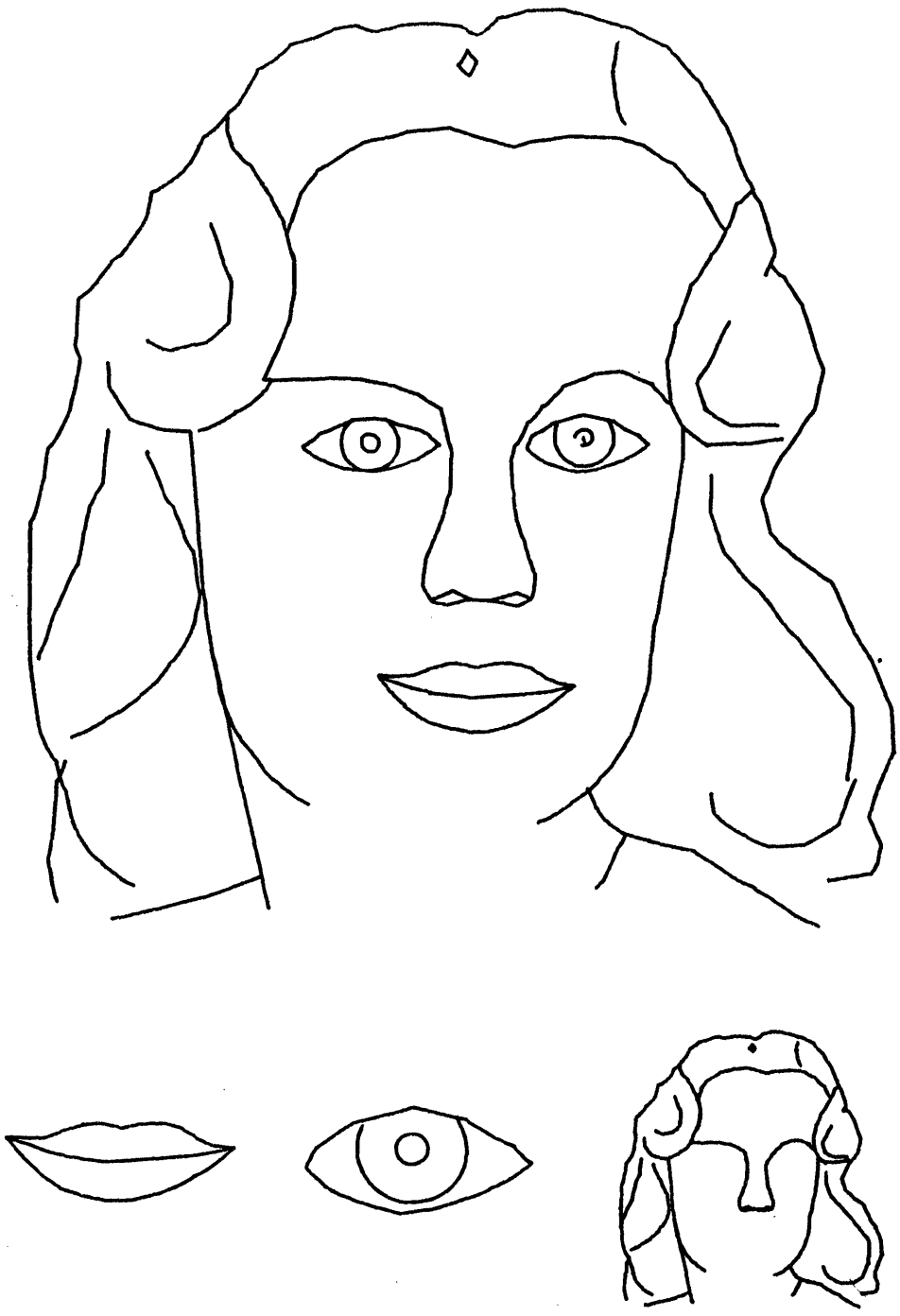


FIGURE 9.9.
GIRL TRACED FROM PHOTOGRAPH

FIGURE 9.10.
GIRL WITH FEATURES CHANGED

the effect of certain features on the appearance of a face. It is the understanding that can be gained from computer drawings that is more valuable than mere production of a drawing for shop use.

ELECTRICAL CIRCUIT DIAGRAMS

Electrical engineers are, of course, interested in making circuit diagrams. It is not surprising that Sketchpad should be applied to this task. Unfortunately, electrical circuits require a great many symbols which have not yet been drawn properly with Sketchpad and are not therefore in the library. After some time is spent working on the basic electrical symbols it may be easier to draw circuits. So far, however, circuit drawing has been a big flop.

The circuits of Figure 9.11 are parts of an analog switching scheme. You can see in the figure that the more complicated circuits are made up of simpler symbols and circuits. It is very difficult, however, to plan far enough ahead to know what composits of circuit symbols will be useful as subpictures of the final circuit. The simple circuits shown in Figure 9.11 were compounded into a big circuit involving about 40 transistors. Including much trial and error, the time taken by a new user (for the big circuit not shown) was ten hours. At the end of that time the circuit was still not complete in every detail and he decided it would be better to draw it by hand after all.

CONCLUSIONS

The circuit experience points out the most important fact about computer drawings. It is only worthwhile to make drawings on the

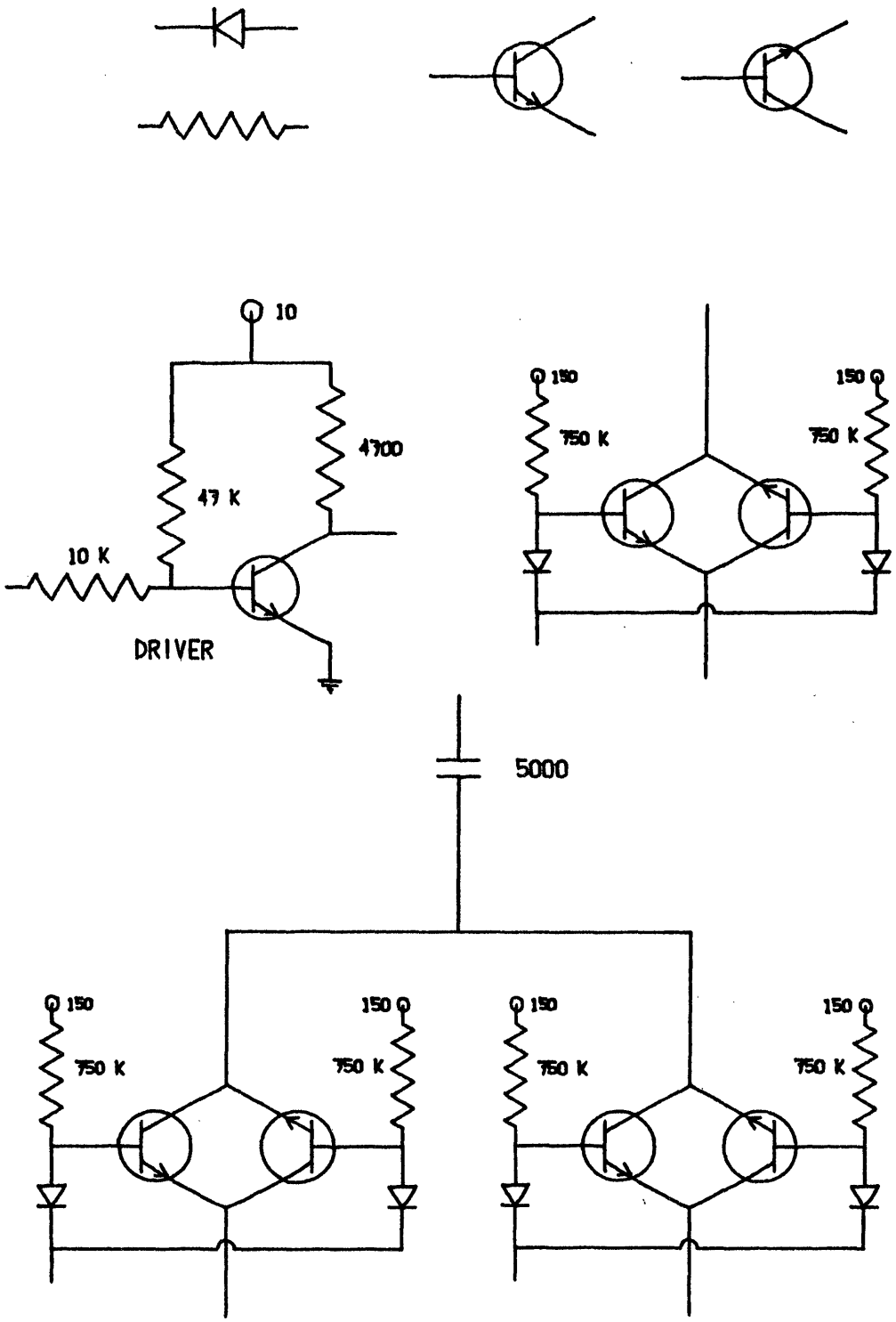


FIGURE 9.11.
CIRCUIT DIAGRAMS

computer if you get something more out of the drawing than just a drawing. In the repetitive patterns we saw in the first examples, precision and ease of constructing great numbers of parts were valuable. In the linkage examples, we were able to gain an understanding of the behavior of a linkage as well as its appearance. In the bridge examples we got design answers which were worth far more than the computer time put into them. If we had had a circuit simulation program connected to Sketchpad so that we would have known whether the circuit we drew worked, it would have been worth our while to use the computer to draw it. We are as yet a long way from being able to produce routine drawings with the computer.

FUTURE WORK

The methods outlined in this report generalize nicely to three dimensional drawing. In fact, work has already been begun to make a complete "Sketchpad Three" which will let the user communicate solid objects to the computer. A forthcoming thesis by Timothy Johnson of the Mechanical Engineering Department will describe this work. When Johnson is finished it should be possible to aim at a particular place in the three dimensional drawing through two dimensional, perspective views presented on the display. Johnson is completely bypassing the problem of converting several two dimensional drawings into a three dimensional shape. Drawing will be directly in three dimensions from the start. No two dimensional representation will ever be stored.

Work is also proceeding on direct conversion of photographs into line drawings. Roberts reports a computer program[8] able to recognize

simple objects in photographs well enough to produce three dimensional line drawings for them. Roberts is storing his drawings in the ring structure described in Chapter III so that his results will be compatible with the three dimensional version of Sketchpad.

Much room is left in Sketchpad itself for improvements. Some improvements are minor, such as including mirror image subpictures. Some improvements should be made to suit Sketchpad to particular uses that come up. For example, it is so interesting to study the path of particular points on a linkage that Sketchpad should be able to store and later display the path of chosen points.

More major improvements of the same order and power as the existing definition copying capability can be forseen. At present Sketchpad is able to add defined relationships to an existing object drawing. A method should be devised for defining and applying changes which involve removing some parts of the object drawing as well as adding new ones. Such a capability would permit one to define what rounding off a corner means. Then, by pointing at any corner and applying that definition, one could round off any corner. Sketchpad cannot now do this because rounding off a corner involves disconnecting the two lines which form the corner from the corner point and then putting a small circular arc between them.

HARDWARE

Sketchpad has pointed out some weaknesses in present computer hardware. A proposal for a line drawing display which would greatly surpass the capability of the spot display now in use is given in

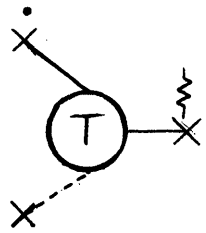
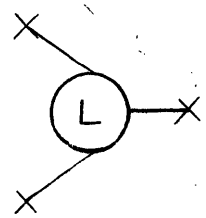
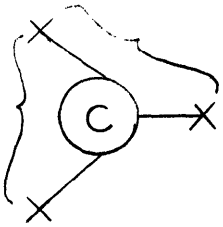
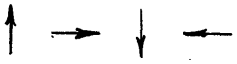
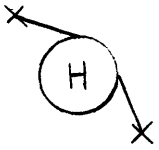
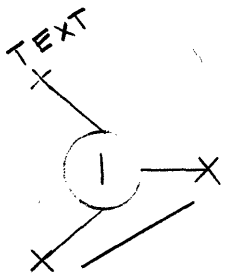
Appendix E. Such a display would not only provide flicker free display to the user, but also would relieve the computer of the burden it now carries in computing successive spots in the display.

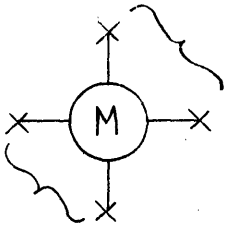
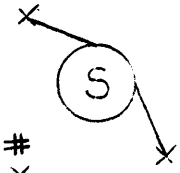
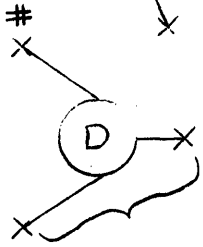
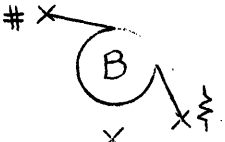
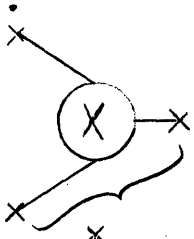
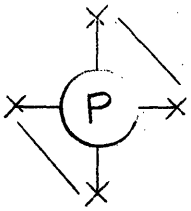
There are two conflicting demands made by Sketchpad on the light pen. On the one hand, the pen must have a fairly large field of view for ease of tracking. On the other hand, it should have a small field of view for aiming at objects. It should be possible to build a pen with two concentric fields of view which would report to the computer separately.

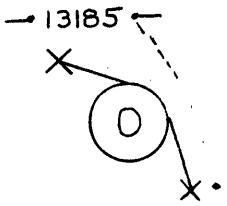
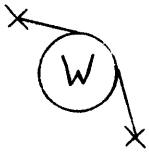
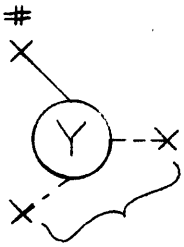
The arithmetic element of the computer is not used in doing the ring structure processing which forms a large part of Sketchpad. On the other hand, the index registers and their associated arithmetic are extensively used. This suggests that several users could share an arithmetic element if sufficiently powerful index arithmetic were made available to each of them.

Appendix A

CONSTRAINT DESCRIPTIONS

	code	variable types	description
	43 T	point instance (point)	Point bears same relation to instance that (point) bears to its picture. GENERATED AUTOMATICALLY WITH INSTANCES
	33 L	p thing p thing p thing	Three things are collinear. Note: no distinction made about ordering of variables. GENERATED AUTOMATICALLY WHEN POINTS ARE CREATED ON LINES
	22 C	p thing p thing p thing	Distance from first to second is equal to distance from first to third. (First is circle center.) GENERATED AUTOMATICALLY WHEN POINTS ARE CREATED ON CIRCLES
	24 E	4 thing	Thing is erect or on its side. 
	27 H	p thing p thing	First thing is directly above or below, or directly beside second thing. (Horizontal or vertical line.) GENERATED AUTOMATICALLY FOR ANY LINE BY HORV BUTTON
	30 I	4 thing p thing p thing	4 thing is "parallel" to line between p things. Parallel to horizontal line means upright. (To set angle of text.)

	code	variable types	description
	34 M	p thing p thing p thing p thing	Distance from first thing to second is 1/3,1/2,1,2,3, times distance from third to fourth.
	42 S	4 thing 4 thing	First thing is 1/3,1/2,1,2,3 times size of second thing.
	23 D	scalar p thing p thing	Value of scalar equals distance between things in inches.
	21 B	scalar 4 thing	Value of scalar equals size of thing in inches.
	25 F	instance	Instance is full size, i.e. the same size as its master picture.
	47 X	p thing p thing p thing	First thing is at mid point of other two, e.g. dimension in dimension line is at center of line.
	06 6	4 thing	Thing is 1/32,1/16,1/8,1/4,1/2 or 1 inch in overall size.
	37 P	p thing p thing p thing p thing	Line from first to second would be parallel or perpendicular to line from third to fourth. (Lines need not be there.)

	code	variable types	description
	36 O	4 thing p thing	p thing will be next to 4 thing with enough space for 5 digit number, e.g. to create space in dimension line.
	46 W	p thing p thing	Distance between things is maintained what it was last time meta of tog 22 was down. USES META OF TOG 22. e.g. for bridges and linkages.
	50 Y	scalar (p thing) (p thing)	Value of scalar is equal to change in distance between p things since meta of tog 22 was down, sign considered. e.g. to display forces in beams. USES META OF TOG 22.

Appendix B

PUSH BUTTON CONTROLS

BUTTON NAME	BIT NUMBER	FUNCTION
Draw	1.8	Create a new straight line segment or circle arc. End of line or arc left attached to light pen.
Circle center	1.7	Center of circle is left where pen is pointing. Next thing drawn will be circle arc.
Move	2.1	Object pointed at moves with light pen.
Delete	1.3	Object pointed at removed from drawing.
Instance	2.4	Instance of picture whose number is in toggle register 25 is created.
Copy 20 Copy 21 Copy 22 Copy 23	3.6 3.1 2.5 1.9	Four buttons. Copy definition picture indicated in toggle registers 20 to 23 respectively. These buttons can be set up to create equal length lines, dimension lines, etc. Any four functions can be available at once.
Stop	1.6	Leave moving object wherever it is. Merge moving object if aiming at object of like type. Same as termination flick of the pen.
Text	4.3	Create line of text consisting only of the letter X. Typing while a piece of text is moving adds to the text displayed.
Number	3.7	Create a new set of digits and a scalar which is its value. Digits left moving.
Hold	4.9	Following pen flick not to be taken as termination signal. Used to set pen aside for typing text.
Garbage	1.1	If pen is tracking, recenter picture so that place pen is pointing at will be in the center. If pen not tracking, compact ring structure by removing garbage.

BUTTON NAME	BIT NUMBER	FUNCTION
Constraint	2.8	Create a new constraint of the type numbered in toggle register 25. Dummy variables are created. Constraint left moving.
Horv	2.9	Apply horizontal or vertical constraint to line aimed at. Choice is based on 45^{o} cutoff.
Designate	2.2	Designate object. For copying a definition picture with three or more ties.
Tie	2.6	Object pointed at is an attacher of this picture.
Fix	3.3	This object must not move during constraint satisfaction. Moving an object with the light pen unfixes it.
Unfix	2.7	All fixed and designated objects unfixed and undesignated.
IBM	4.3	Read tape record. Number of record on tape given in toggle 26. Typewriter confirms successful reading or writing.
Library	3.9	Read a record from the TX-2 library tape. Address of record given in toggle register 27. Typewriter confirms.
Library write	Special start point	Write a record on library tape. Typewriter confirms.
Change instance	2.3	Moving instance or instance pointed at is changed to type indicated in Toggle register 25. Can change resistor into diode, etc.
Dismember	4.4	Instance pointed at is reduced one level, i.e., its internal structure on the next level becomes usable.
Order	4.6	Lines are put in better order for plotting.
Disorder	4.5	Lines are put in worst order for plotting.

BUTTON NAME	BIT NUMBER	FUNCTION
Punch	4.7	Punch plotter tape for object picture.
Plot	4.8	Plot object picture.

The following dangerous functions only operate if "meta" button (4.10) is pressed as well.

Delete constraints	1.2	All constraints in object picture are deleted.
Delete points	1.4	All unattached points in object picture are deleted.
Delete picture	1.5	Entire object picture is deleted.
IBM	4.3	Write IBM tape record. Typewriter confirms.

STRUCTURE OF STORAGE BLOCKS

(C) = Chicken (S) = Start of subroutine . = Spare register
(H) = Hen - = Ring part of component {} = Data part of block

TYPE OF BLOCK	STRUCTURE		REMARKS
Universe Variables Holders Constraints Topos Frees Deads Movings Curpics Freedoms Fixeds Desigs Mergers Works	TYPE - SPECB - {NAME .}	(C) (H)	All these short generic blocks use the same format. TYPE is a chicken (C) which connects the block to its next higher level in the generic structure, see Figure 3.8. SPECB is the hen (H) collecting the TYPE blocks in the next lower level. TYPE and SPECB serve this purpose in all blocks where they appear. NAME contains a four letter typewriter code name for each generic block. Counting lines, one finds that TYPE = 0, SPECB = 2, and NAME = 4.
Lines Circles Pictures	TYPE - SPECB -	(C) (H)	(Generic blocks for lines, circles and picture blocks.)
	NAME DISPLAY HOWBIG . MOVIT SIZE . KIND	 (S) (S) (S) 	 Display subroutine. Fit scope around this thing. Apply transformation to this thing (Degenerate) Length of line, circle and picture blocks. Put these in PPART or PICBLKS of a picture block.
Scalars Points Instances Texts Digits Dummies	TYPE - SPECB -	(C) (H)	(Generic blocks for various kinds of variables.)
	NAME DISPLAY HOWBIG . MOVIT SIZE WHERE KIND TUPLE VARLOC	 (S) (S) (S) (S) 	 Apply transformation to this thing. Find position of thing on display. Number components in vector. Location of first vector component in block.

Hov	TYPE	(C)	Generic blocks for various constraint
Porp	-		types.
etc.	SPECB	(H)	
etc.	-		
	NAME		
	DISPLAY	(S)	
	HOWBIG	(S)	Degenerate. (Does nothing.)
	.		
	MOVIT	(S)	Degenerate.
	SIZE		
	CONLET		Letter to appear in display.
	KIND		
	.		
	.		
	COMP	(S)	Error computing subroutine.
	NCON		Number degrees of freedom removed.
	CHVAR		Number of changeable variables.
Picture	TYPE	(C)	(Specific picture block.)
	-		
	PICBLKS	(H)	Abstractions in picture. KIND of generic
	-		block tells if a thing is an abstraction.
	PPART	(H)	Picture parts. Lines, Circles, Instances,
	-		Texts, and Digits in picture.
	PWHOS	(C)	Put into SPECB of Curpics ring if this
	-		is current picture.
	PPARTM	(H)	Moving parts of picture.
	-		
	PATAP	(H)	Attachers of this picture.
	-		
	PINS	(H)	Instances of this picture.
	-		
	PSIZE		Overall size of this picture.
	PNAME		36 bit "name" for this picture.
	PSAVE		Space to save transformation when recursively
	"		expanding instances.
	"		
	"		
	"		
	.		
Line	TYPE	(C)	(Specific line block.)
	-		
	ATATAP	(C)	Put into PATAP of picture if this line
	-		is an attacher.
	BWHOS	(C)	Which picture this thing belongs to.
	-		
	VORD	(C)	Put into SPECB of Movings if this line
	-		is moving.
	LSP	(C)	Start point of line. Goes into PLS ring
	-		of point.
	LEP	(C)	End point of line.
	-		

Circle	TYPE	(C)	(Specific circle block.)
	-		
	ATATAP	(C)	
	-		
	BWHOS	(C)	
	-		
	VORD	(C)	
	-		
	CSP	(C)	Start point of circle arc.
	-		
	CEP	(C)	End point of circle arc.
	-		
	CIRCEN	(C)	Center point of circle.
	-		
	CVAL		Angle of circle arc (to avoid ambiguity).
	"		Radius of Circle (to save recomputation).
Point	TYPE	(C)	(Specific point block.)
	-		
	ATATAP	(C)	
	-		
	BWHOS	(C)	
	-		
	VORD	(C)	Put in SPECB of Freedoms during maze-solving constraint satisfaction.
	-		
	VFLW	(H)	Constraints which this variable will be used to satisfy.
	-		
	VCON	(H)	Constraints on this variable.
	-		
	PLS	(H)	Lines and Circles on this point
	-		
	IPCOTP	(H)	Instance-point constraints which use this point for reference only.
	-		
	PVAL		X coordinate of point.
	"		Y coordinate of point.
Instance	TYPE	(C)	Specific instance block. Size of instance is half size of enclosing box.
	-		
	ATATAP	(C)	
	-		
	BWHOS	(C)	
	-		
	VORD	(C)	
	-		
	VFLW	(H)	
	-		
	VCON	(H)	
	-		
	IWHAT	(C)	What picture this is an instance of.
	-		
	IVAL		Size times cosine of rotation.
	"		Size times sine of rotation.
	"		X coordinate.
	"		Y coordinate.

Text	TYPE	(C)	Particular lines of text. Size of text is half height of letters. Position of text is center of first letter in the line.
	-		
	ATATAP	(C)	
	-		
	BWHOS	(C)	
	-		
	VORD	(C)	
	-		
	VFLW	(H)	
	-		
	VCON	(H)	
	-		
	TVAL		Size times cosine of rotation.
	"		Size times sine of rotation.
	"		X coordinate.
	"		Y coordinate.
	TXTS		Text to be shown, four letters per register, typewriter codes.
	"		
	"		
	"		
	"		
	"		
	"		
	"		
	"		
	"		
Dummy	TYPE	(C)	(Particular dummy variable.)
	-		
	ATATAP	(C)	
	-		
	BWHOS	(C)	
	-		
	VORD	(C)	
	-		
	VFLW	(H)	
	-		
	VCON	(H)	
	-		
	TPVAL		X coordinate.
	"		Y coordinate.

Digits	TYPE	(C)	(A particular set of digits. Size of digits is half height of figures.)
	-		
	ATATAP	(C)	
	-		
	BWHOS	(C)	
	-		
	VORD	(C)	
	-		
	VFLW	(H)	
	-		
	VCON	(H)	
	-		
	NTOSHOW	(C)	Scalar whose value is to be shown.
	-		
	NVAL		Size times cosine of rotation.
	"		Size times sine of rotation.
	"		X position.
	"		Y position.
Scalar	TYPE		(A particular scalar block.)
	-		
	ATATAP	(C)	
	-		
	BWHOS	(C)	
	-		
	VORD	(C)	
	-		
	VFLW	(H)	
	-		
	VCON	(H)	
	-		
	SSHOW	(H)	Digits showing this scalar's value.
	-		
	SVAL		Value of scalar.
	.		
Constraint	TYPE	(C)	(All constraint blocks have same format. If fewer than four variables, block will be shorter and VARIATION will be moved up.)
	-		
	ATATAP	(C)	
	-		
	BWHOS	(C)	
	-		
	CVTS,VORD	(C)	Variable used to satisfy this constraint in maze-solving method.
	-		
	VAR1		First constrained variable.
	-		
	VAR2		Second constrained variable.
	-		
	VAR3		
	-		
	VAR4		
	-		
	VARIATION		Code for variations within a constraint type. e.g., horizontal or vertical.
	-		

RING OPERATION MACRO INSTRUCTIONS

The macro instructions listed in this appendix are used to implement the basic ring operations listed in Chapter III. Only the format is given here since to list the machine instructions generated would be of value only to persons familiar with the TX-2 instruction code. In each case the macro name is followed by dummy variables separated by non-alphabetic symbols. The dummy variables XR and XR2 refer to index registers which contain the address of the block which contains the ring element being worked on. The terms N of XR or N×XR mean the Nth element of the block pointed to by index register XR, for example, the LSP (line start point) register of the line block pointed to by index register α.

LTAKE≡N×XR

Take N of XR out of whatever ring it is in. The ring is reclosed. If N of XR is not in a ring, LTAKE does nothing. N of XR must not be a hen with chickens.

PUTL≡N×XR→M×XR2

PUTR≡N×XR→M×XR2

Put N of XR into the ring of which M of XR2 is a member. N of XR is placed to the left (PUTL) or right (PUTR) of M of XR2. M of XR2 may be either a hen or a chicken. N of XR must not already belong to a ring.

MOVEL≡N×XR→M×XR2

MOVER≡N×XR→M×XR2

Combination of LTAKE and PUTL (PUTR). Assumes that both N of XR and M of XR2 are in the same ring. Intended for reordering a ring.

CHGRL≡N×XR→M×XR2

CHGRR≡N×XR→M×XR2

Combination of LTAKE and PUTL (PUTR). N of XR and M of XR2 may be in different rings.

LGORR≡N×XR=XR2→SUBR→LEXIT

LGORL≡N×XR=XR2→SUBR→LEXIT

Go around the ring of which N of XR is the hen. Exit to subroutine SUBR once for each ring member. The address of the top of the block to which each ring member belongs is put in XR before starting the subroutine. XR2 is used as a working index register. The subroutine may destroy the contents of both XR and XR2. The subroutine may delete individual members of the ring provided recursive deletion does not delete additional ring members. The subroutine must not generate new ring members. Jump to LEXIT when finished with the ring. Go around the ring to the right (LGORR) or left (LGORL).

LGORRI≡N×XR=XR2→SUBR→LEXIT

LGORLI≡N×XR=XR2→SUBR→LEXIT

Same as LGORR except that the subroutine may generate new members in the ring. The subroutine must not delete the current member of the ring. New members will be visited if they are put in the ring later in sequence.

COMBHR≡N×XR→M×XR2

COMBHL≡N×XR→M×XR2

The members of the ring whose hen is at N of XR are placed in the ring of which M of XR2 is a member. N of XR must not be empty. The new members are placed to the right (COMBHR) or left (COMBHL) of M of XR2. M of XR2 may be either a hen or a chicken. N of XR is left empty.

Appendix E

PROPOSAL FOR AN INCREMENTAL CURVE DRAWING DISPLAY

In the course of the work with Sketchpad it has become all too clear that the spot-by-spot display now in use too slow for comfortable observation of reasonable size drawings. Moreover, having the central machine compute and store all the spots for the display is a waste of general purpose capacity that might better be applied to other jobs. As a solution to these difficulties I propose that a special purpose incremental computer be used to generate the successive spots of the display at high speed. The central machine would provide only a minimum of information about each curve to be drawn; e.g., end points of lines; start, center and arc length of circle arcs.

The technology of incremental computers is well developed, but so far as I know, no one has yet applied them directly to the problem of computer display systems. Basically the incremental computer works by adding one register to another successively and detecting any overflows or underflows which may be generated. Certain registers are incremented conditionally on the result of overflow or underflow generation.

In the system of Figure E.1, the x and y increment registers are added to the x and y remainder registers and overflows or underflows (dotted lines) are used to increment the beam position of the display. A counter (not shown) is provided to limit the length of the straight line generated. The unit would request more information from the computer after the appropriate number of additions. For drawing straight lines on a 1024 x 1024 raster display, the increment registers should

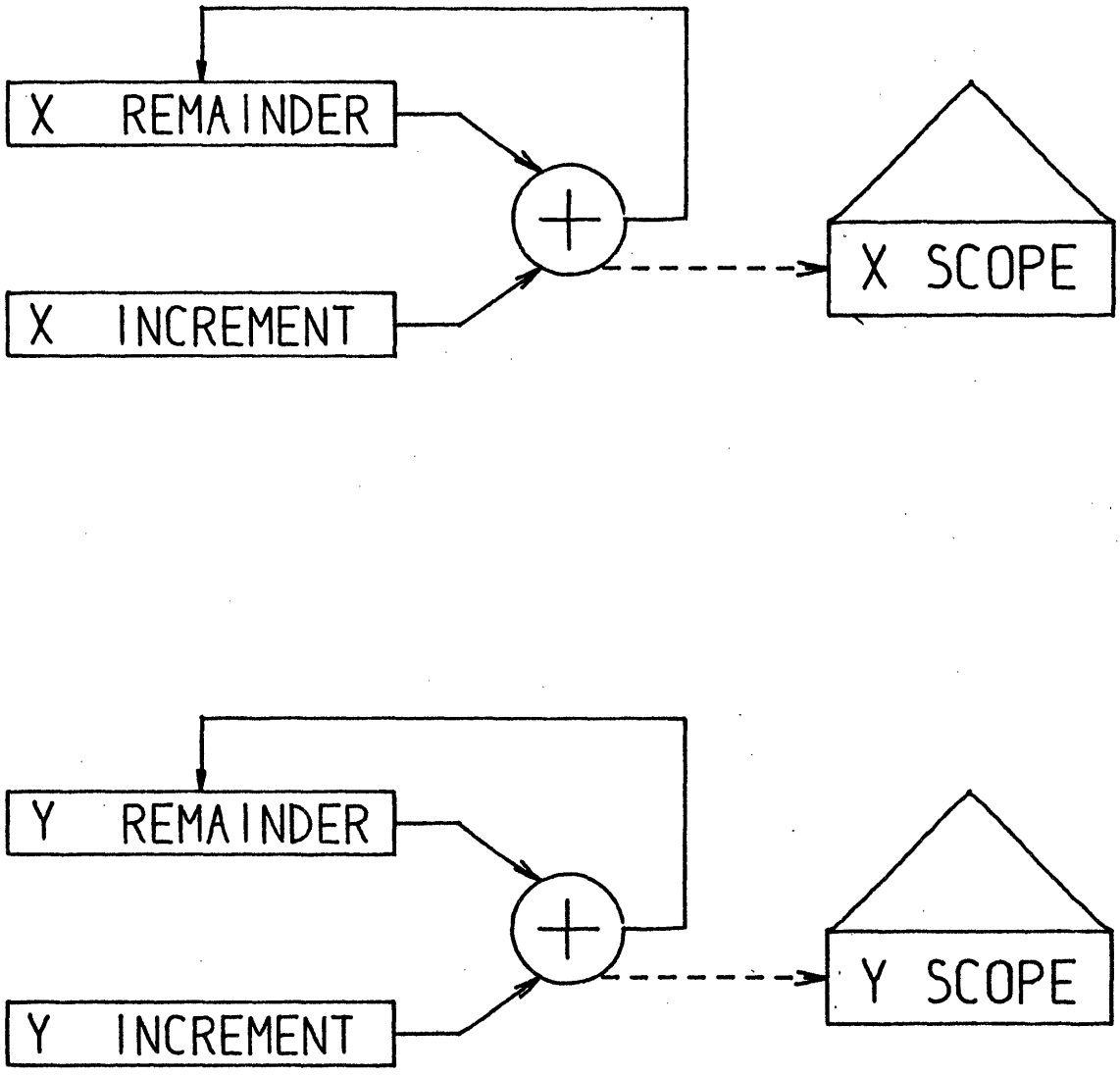


FIGURE E.1.
DDA FOR DRAWING LINES

contain 10 bits plus sign, 11 bits in all each; the remainder registers should contain 10 bits with no sign; and the counter should contain 10 bits.

To understand how the system of Figure E.1 operates consider that its x increment register contains the largest possible positive number and that its y increment contains one half that value. The x addition would result in overflow nearly every iteration, whereas the y addition would result in overflow only on alternate additions, and so a line would be drawn up and to the right with a slope of 1/2.

The usual practice in incremental computers is to be able to step the increment registers by a single unit up or down according as overflow or underflow is produced in another addition. In the system of Figure E.2, the (+?) is an adder-subtractor which can increase or decrease the increment register by the amount stored in the curvature register. The (+?) adds or subtracts if overflow or underflow is generated in the other addition. Overflow or underflow is signalled to the (+?) adder along the dotted paths in Figure E.2.

Use of the conditional adder permits a curvature to be specified so that curves can be drawn. The system of Figure E.2 will draw straight lines if the numbers in the curvature registers are zero, circles if the numbers are equal and opposite in sign, ellipses if the numbers are unequal and unlike in sign, and hyperbolas if the numbers are like in sign. The ellipses and hyperbolas are generated, however, with axes parallel to the coordinate axes of the display.

Theory and simulation show that just as in the incremental equation used for generating circles (see Chapter V), the latest value of increment must be used if the curve is to close. Therefore, the additions

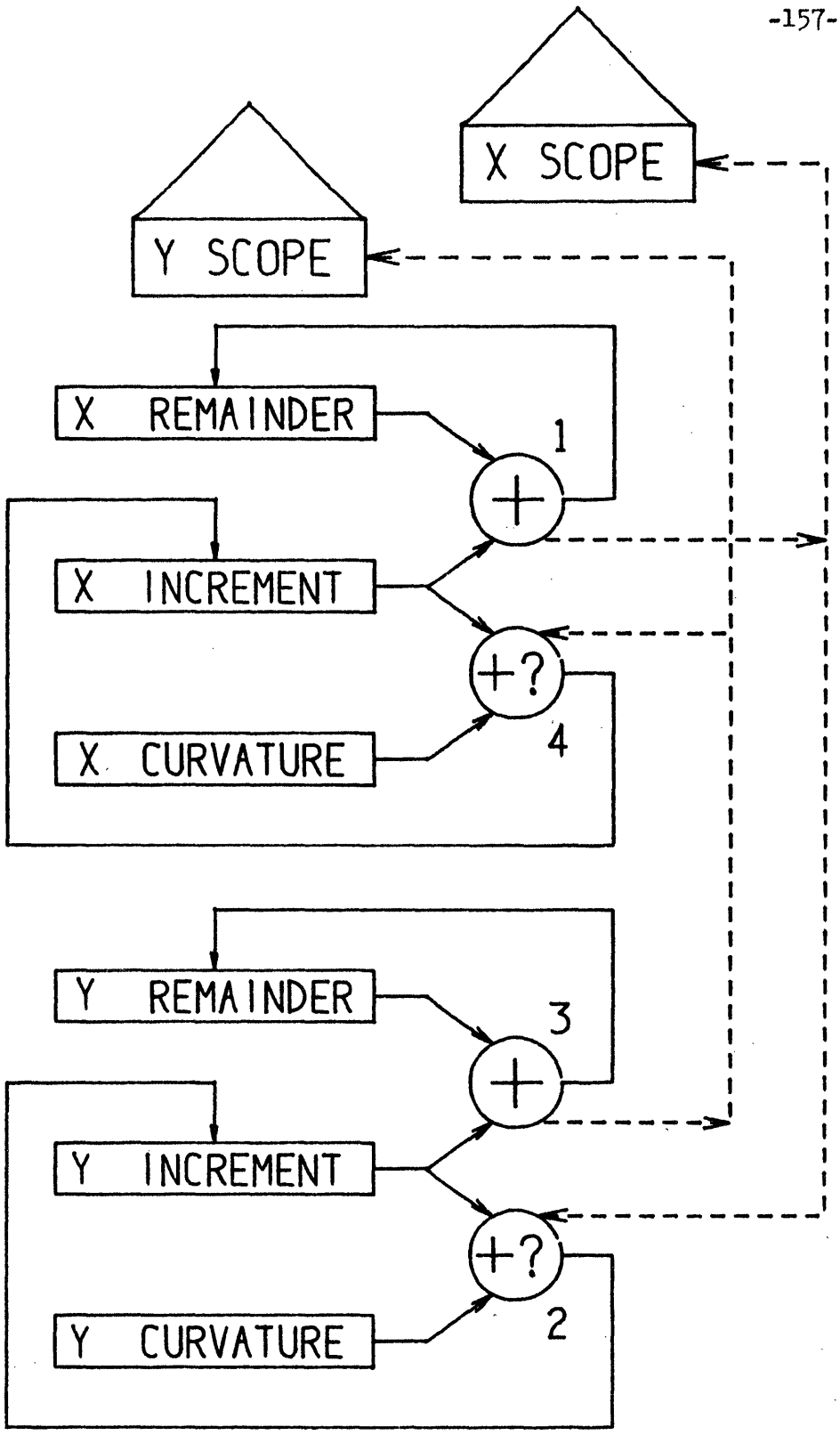


FIGURE E.2.
DDA FOR UPRIGHT CONICS

cannot all occur at once; the order shown in Figure E.2 by the numbers 1-4 next to the adders makes the circles and ellipses close. In a serial device it is possible to do the four additions in just two add times by having only a one bit time delay between the two additions for each coordinate, i.e., ⊕? just before ⊕.

Circles can be drawn with radii from about one scope unit to a straight line according to the numbers put in the curvature registers. Simulation shows that if the increment and curvature registers contain 17 bits plus sign, 18 bits each in all, and the remainder contains 17 bits without sign, the largest radius circle that can be drawn is just noticeably different from a straight line after having passed fully across a 1024 x 1024 raster display. The simulation program for this test is less than 100 instructions long and requires, of course, no multiply or divide. Simulation of larger incremental computers on small general purpose digital computers should be a powerful way to get complex numerical answers quickly and easily.

If the system of Figure E.2 is duplicated twice as shown in Figure E.3, a general Conic Section drawing capability is obtained. I am indebted to Larry M. Delfs for pointing out that the display incrementing outputs of the two systems should be added together. The full system of Figure E.3 can draw not only arbitrary conic sections but a host of interesting cycloidal curves. For drawing the simple straight lines and circles, the two halves of the system would be loaded with identical numbers to gain a two-fold speed advantage.

A trial design using 20 megacycle serial logic and 36 bit delay lines available commercially showed that the full system would be able

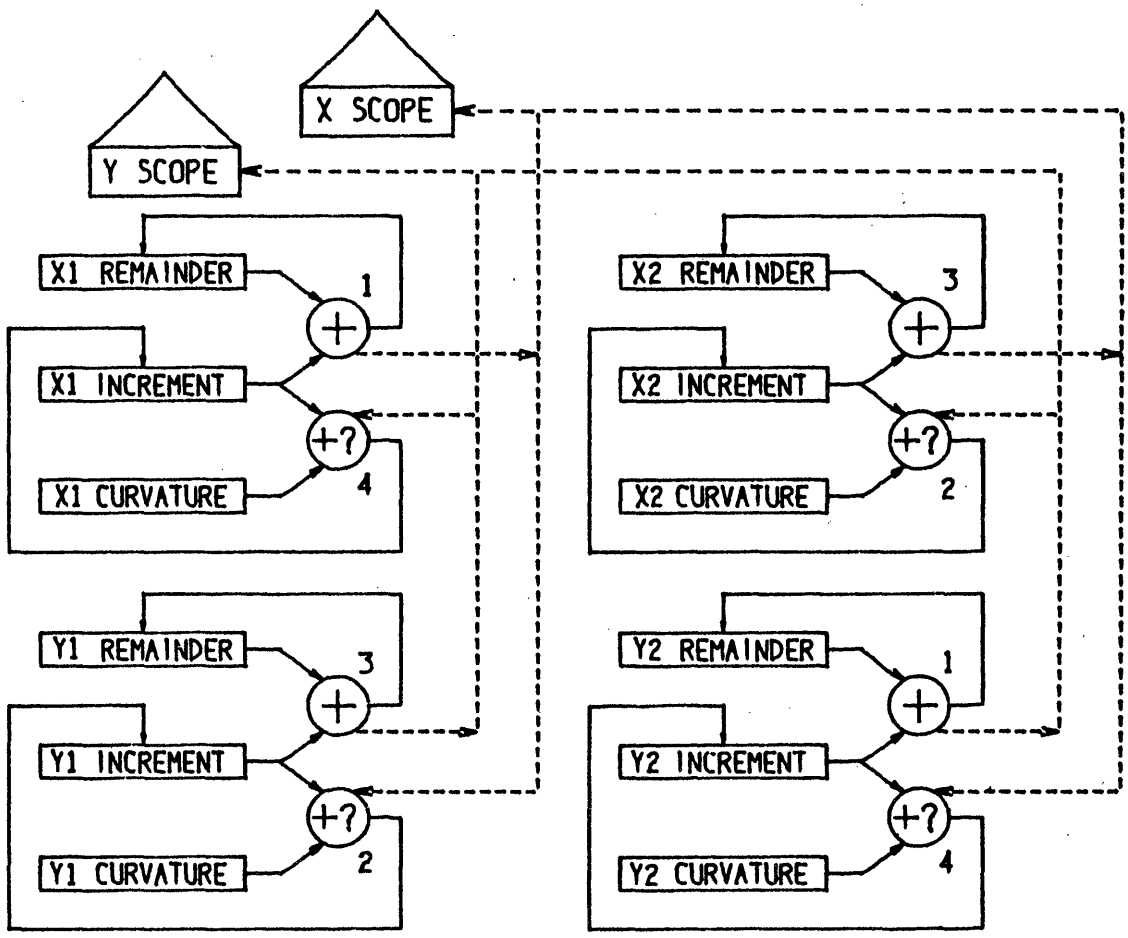


FIGURE E.3.
DDA FOR THE GENERAL CONIC

to generate new display points at 0.9 microseconds each for lines and circles and slightly slower (but not half speed) for complicated conics. This corresponds to a writing rate of about 10,000 inches per second. Some saving in cost could be expected if longer delay lines were used and a correspondingly slower operation speed were tolerated. It appears possible to get similar performance from a parallel scheme.

Appendix F

MATHEMATICS OF LEAST MEAN SQUARE FIT

The result quoted in this appendix is well known and is repeated here only for reference.

Suppose we have P equations in N unknowns:

$$\sum_{j=1}^{N} a_{ij}\, x_j = c_i \quad 1 \leq i \leq P; \qquad \text{or} \qquad AX = C\,. \tag{F-1}$$

If P is larger than N there would in general be no exact solution. We wish to find the values for the unknowns which minimize the sum of the squared errors of the equations. The error in the i^{th} equality is given by:

$$E_i = \sum_{j=1}^{N} (a_{ij}\, x_j - c_i), \tag{F-2}$$

and the total squared error,

$$E_t = \sum_{i=1}^{P} \left[\sum_{j=1}^{N} (a_{ij}\, x_j) - c_i \right]^2. \tag{F-3}$$

We wish to minimize E_t, and so we take partials with respect to each x_j and set all these equal to zero. For a particular x_j called x_k,

$$\frac{\partial E_t}{\partial x_k} = \frac{\partial}{\partial x_k} \sum_{i=1}^{P} \left[\sum_{j=1}^{N} (a_{ij}\, x_j) - c_i \right]^2. \tag{F-4}$$

Since the partial of a sum is equal to the sum of the partials,

$$\frac{\partial E_t}{\partial x_k} = \sum_{i=1}^{P} \frac{\partial}{\partial x_k} \left[\sum_{j=1}^{N} (a_{ij}\ x_j) - c_i \right]^2, \qquad \text{(F-5)}$$

or since

$$\frac{\partial}{\partial x} (Q)^2 = 2Q \frac{\partial}{\partial x} Q,$$

$$\frac{\partial E^2}{\partial x_k} = \sum_{i=1}^{P} 2 \left[\sum_{j=1}^{N} (a_{ij}\ x_j) - c_i \right] \frac{\partial}{\partial x_k} \left[\sum_{j=1}^{N} (a_{ij}\ x_j) - c_i \right]. \qquad \text{(F-6)}$$

Now the last part of (F-6) is a sum of terms like $a_{12}x_2$ ... only one of which involves x_k at all, namely $a_{ik}x_k$. Therefore,

$$\frac{\partial E^2}{\partial x_k} = \sum_{i=1}^{P} 2 \left[\sum_{j=1}^{N} (a_{ij}\ x_j) - c_i \right] (a_{ik}), \qquad \text{(F-7)}$$

which, when set equal to zero gives:

$$0 = \sum_{i=1}^{P} \left[\sum_{j=1}^{N} (a_{ik}\ a_{ij}\ x_j) - a_{ik}\ c_i \right], \qquad \text{(F-8)}$$

or

$$\sum_{i=1}^{P} \sum_{j=1}^{N} a_{ik}\ a_{ij}\ x_j = \sum_{i=1}^{P} a_{ik}\ c_i. \qquad \text{(F-9)}$$

Changing the order os summation,

$$\sum_{j=1}^{N} \left(\sum_{i=1}^{P} a_{ik}\ a_{ij} \right) x_j = \left(\sum_{i=1}^{P} a_{ik}\ c_i \right), \qquad \text{(F-10)}$$

which in matrix notation becomes:

$$A^T A X = A^T C. \qquad \text{(F-11)}$$

A^TA is a square matrix of order N. Thus a system of any number of linear equations can be reduced to a simpler system whose solution is the value of the variables for least square fit to the original set of equations.

If the original equations are equations in two unknowns, a plot of (F-2) with error squared in the upward direction is a parabolic valley. Since any vertical section of a parabolic valley will be a parabola, and the sum of any two parabolas in likewise a parabola, a plot of (F-3) can at most be an eliptic paraboloid. The Equations (F-10) and (F-11) resulting from the method described here represent the locus of locations where contour lines of the eliptic paraboloid are parallel to the axes. The intersection of these loci, the solution of (F-11), is the lowest point in the eliptic paraboloid, the least mean squares fit to (F-1).

Appendix G

A BRIEF DESCRIPTION OF TX-2*

At first glance, TX-2 is an ordinary single-address, binary digital computer with an unusually large memory. It is an experimental machine—many of its in-out devices are not commercially available. On closer inspection, one finds it has some important innovations—at least they were innovations at the time TX-2 was built (1956).

The distinctive features of TX-2 are:

1. Simultaneous use of in-out machines through interleaved programs.
2. Flexible, "configured" data processing.

Some other virtues include:

1. Automatic memory and arithmetic overlap.
2. A "bit" sensing instruction (i.e., the operand is one bit!).
3. Addressable arithmetic element registers.
4. Especially flexible in-out.
5. 64 index registers.
6. Indirect—i.e. deferred addressing.
7. Magnetic Tape Auxiliary Storage

IN-OUT

The phrase "simultaneous use of in-out machines" should be taken quite literally. It does not mean simultaneous control. Each unit has

---

* By Alexander Vanderburgh

its own buffer register and only one of these can be processed by TX-2 at any given instant. It is the relative speed that is important. For example, the in-out instruction that "fills" the display scope buffer takes no more than 10 microseconds, but the display itself takes from 20 to 100 microseconds, i.e., up to ten times as long. While the display is busy, the computer can compute the next datum of course, but it can also initiate other in-out transfers. In practice, since most in-out units are much slower than their associated programs, the computer spends a significant percentage of the time just waiting (in "Limbo"), even when several devices are in use. Interleaved initiation of in-out data transfers is partly automatic and partly program controlled. Each in-out routine is independently coded and is operated by TX-2 according to its "priority." Each unit has a "Flag Flip-Flop" to indicate to control that it is ready for further attention. When a unit is ready for further attention its routine will be operated unless another unit of higher priority also needs attention. An index register is reserved for each in-out unit and is used as a "place-keeper" when its routine is not being operated. The sharing among in-out routines of storage, index memory, and the arithmetic element is the programmer's responsibility.

"CONFIGURED" DATA PROCESSING

The "normal" word length for TX-2 is 36 bits. For many applications 18 or 9 bits would suffice, and in some cases each piece of data requires the same processing. Configuration control permits "fracture" of the normal word into two 18 bit pieces, four 9 bit pieces, or one 27

bit and one 9 bit. These "subwords" are completely independent—for example, there are separate overflow indicators. In addition to "fracture" there is "activity" and "quarter permutation". Any quarter word can be made "inactive" i.e., inoperative. The 9 bit quarters of a datum from memory may be rearranged (permuted) before use. There are 8 standard permutations—for example, the right half of memory can be used with the left half of the arithmetic element. Nine bits are required for complete configuration specification. Since only 5 bits are available for bit thin film memory is addressed by each instruction word, a special 32 word, 9 bit thin film memory is addressed by each instruction that processes data directly. A complete change to any of 32 configurations is therefore possible from instruction to instruction.

THE SMALLER VIRTUES

Overlap: TX-2 has two core memories—"S" memory, a vacuum tube driven 65,536 word core memory, and "T" memory, a transistor driven 4096 word core memory about 20% faster. Instruction readout can be done concurrently with the previous data readout if program and data are in separate memories.

The use of the arithmetic element is also overlapped. Instructions that follow a multiply or divide operation will be done during the arithmetic time if they make no reference to the arithmetic element. The overlap is entirely automatic and may be ignored if the programmer chooses. A careful programmer can gain speed by doing indexing after multiply or divide and by putting program and data in separate memories.

Bit Sensing Instruction: One instruction--SKM--uses a single bit of any memory word as its operand. Control bits provide 32 variations of skipping setting, clearing, and/or complementing the selected bit. This instruction can also cycle the whole word right one place if desired.

Addressable Arithmetic Element: Seventeen bits of the TX-2 instruction word are reserved for addressing an operand. This would allow a 131,072 word memory. TX-2 has only 69,632 registers of core storage. The toggle switch and plugboard memories, the real time clock register, the knob register (shaft encoder), and the arithmetic element registers use 55 of the remaining addressing capability. The arithmetic element registers are therefore part of the memory system and can be addressed, e.g., one can add the accumulator to itself.

Flexible In-Out: The TX-2 user must program each and every datum transfer. The lack of complex automatic in-out controls may seem to be a burden, but the simplicity of the system gives the programmer much more precise and variable control than automatic systems provide. For example, coordination of separate in-out units such as display and light pen is possible. Moreover, it is relatively easy to attach new in-out machines as they become available.

Index Memory and Indirect Addressing: Of the 64 index registers, one must devote a few to each in-out unit's program. With all 21 in-out devices concurrently in use, each program would have two index registers for normal programming use. In practice, one seldom uses more than half a dozen in-out units, and each routine would then have 9--clearly a luxury. Indirect addressing provides a means for indexing normally nonindexable instructions, or for double indexing normal instructions.

Magnetic Tape Auxiliary Storage: Each TX-2 magnetic tape unit stores about 70 million bits, 34 times the capacity of the core memory system. Like a magnetic drum the tape is addressable. It can be read in either direction at any speed from 60 to 600 ips, and can be searched at a maximum of 1200 ips. It is used at present primarily for program storage. "Turn around time"—i.e. the time required to save one program and read-in a different one is seldom more than 2 minutes and often less than 30 seconds. (The read-in time, once the desired section of the tape is found, is about 12 seconds for 69,632 words.) A standard IBM 729 tape unit is also available.

SUMMARY OF VITAL STATISTICS—TX-2—DECEMBER 1962

Word Length:	36 bits, plus parity bit, plus debugging tag bit		
Memory:	256 x 256 core	65,536 words	6.0 μsec cycle time
	64 x 64 core	4,096 words	4.4 μsec cycle time
	Toggle switch	16 words	
	Plugboard	32 words	
Auxiliary Memory:	Magnetic Tape 2+ million words, 70+ million bits per unit (2 units in use, total of 10 planned)		
Tape Speeds:	selectable 60-300 inches/sec, search at 1000 inches/sec (i.e. about 1600 to 8000 36 bit words/sec)		

IN-OUT EQUIPMENT

Input:

Paper Tape Reader: 400-2000 6 bit lines/sec

2 keyboards—Lincoln writer 6-bit codes

Input:

Random number generator—average 57.6 μsec per 9 bit number

IBM Magnetic Tape (Model 729 M6)

Miscellaneous pulse inputs—9 channels—push buttons or other source

Analog input—Epsco Datrac—nominal 11 bit sample
—27 kilocycle max. rate

2 light pens—work with either scope or both on one

Special memory registers:

Real time clock

4 shaft encoder knobs, 9 bits each

592 toggle switches (16 registers)

37 push buttons—any or all can be pushed at once

Output:

Paper tape punch—300 6 bit lines/sec

2 typewriters—10 characters per second

IBM Magnetic Tape (729 M6)

Miscellaneous pulse/light/relay contacts—9 channels (low rates)

Xerox printer—1300 char. sec

2 display scopes—7 x 7 inch usable area, 1024 x 1024 raster

Large board pen and ink plotter—29" x 29" plotting area 15 in/sec slew speed. Off line paper tape control as well as direct computer control.

4-thing — A four component variable: text, digits, or instance.

Aim — To place the light pen so that light from the picture part aimed at falls on the photocell and so that the center of the light pen field of view is sufficiently close to the picture part.

Atomic — Axiomatic, fundamental, built in. The atomic constraints are listed in Appendix A. The atomic operations are each controlled by a push button listed in Appendix B.

Attacher — For instances, a particular point designated in the master for which in the instance the light pen will have a particular affinity. Also the related point created in the picture containing the instance when the instance was created.

For copying, any drawing part designated in the definition picture. Attachers may be recursively merged with object picture parts when the definition is copied.

Balance — The property of equal weight among constraints obtained by making error in a constraint equal to displacement.

Block — A set of consecutive registers used to represent a picture part. An n-component element.

Chicken — A subordinate ring member, composed of two registers one of which references the block containing the hen for this ring, the other references the next and previous chickens in the ring.

Circle — A circle arc. A full circle is a circle arc 360° or more in length.

Constraint — A specific storage representation of a relationship between variables which limits the freedom of the variables, i.e., reduces the number of degrees of freedom of the system. Also, constraint is sometimes used to mean a type of constraint, as in "there are seventeen atomic constraints."

Constraint satisfaction — The process of moving variables so that all the conditions on them embodied in the constraints are met. It is not always possible.

Copying — Duplication in storage the ring structure of a definition picture. A copy is not to be confused with an instance. Any instance may be changed into a copy by dismembering.

Definition — A master picture. Especially a picture to be used for copying, usually containing a combination of atomic constraints. Also the error computation routine associated with a constraint.

Delete — To erase. Deleted blocks become garbage.

Digits — A set of five decimal digits plus sign, leading zeros suppressed. As a variable digits may be moved, rotated, or made larger on the display. The particular value displayed is that of an associated scalar and may be changed only by moving the scalar.

Dismembering — The process of changing an instance into a copy by creating in the ring structure a duplicate of the internal structure of the instance's master and removing the instance. A dismembered instance becomes a group of lines, etc., which may be individually moved, deleted, etc. Dismembering peels off only one layer of instance at a time.

Dummy variable — A particular two component variable used to locate the arms of a constraint when it is first created. Dummy variables may merge with any other kind of variable leaving any attached constraints applying to that variable. Display for a dummy variable is a ×.

Error — The number computed by the definition subroutine for a constraint. Error is zero if the constraint is satisfied and grows monotonically as the constrained variables are moved.

File — A storage structure. A file may be in either list form or table form. Also a collection of magnetic tape records.

Free — A variable which has so few constraints on it that it may be moved to satisfy all of them. Such a variable will be in the FREEDOMS ring.

Garbage — Free storage inside the range of storage addresses being used to represent the drawing.

Hen — A pair of registers in a block used to indicate the first and last references made to that block by the chickens belonging in the hen's ring. Also called a key.

Instance — A fixed geometry subpicture represented very compactly in storage by reference to a master and indication by four numbers of the size, rotation, and location of the subpicture. Internal structure of an instance is visible and may contain other instances, but since it is identical in appearance to the master it cannot be changed without changing the master. Except for size, rotation, and location, all instances of one master look the same.

Key — See hen.

Line — A line segment. No representation for an infinite length line exists in Sketchpad.

Line segment — A topological thing connecting two points. Contains no numerical information. Sometimes called a line.

List — A particular form of storage structure in which each element stores not only the information pertinent to it but also the address of the next element. Not to be confused with a table.

Location — A position in the coordinate system represented by a pair of coordinates. Not to be confused with a point which has a location. Also the address of a particular piece of information in storage.

Master — A picture which is used to define the visible internal structure of an instance.

Merging — Combination of two storage blocks to identify two picture parts, which must be of like type, permanently. The result of a merger of variables takes on the value of the historically older variable. In the ring structure, merging makes one block out of two, reducing the other to garbage. In certain cases merging is recursive.

Moving — Changing the numerical information stored in a variable. Moving a point stores a new coordinate location over the previous one. Moving an instance, text, or digits includes size change and rotation. Moving a scalar implies changing its value but does not change the position of its display. Moving is also the state a thing is in when it is attached to the light pen; it may be stationary on the display. Moving is not to be confused with relocating.

N-component element — A particular form of storage in which various properties of each object represented are stored in consecutive registers. Also the block of registers representing an object.

Numbers — See scalars and digits. Number often refers to digits and scalars collectively. Also the binary numbers stored for a variable.

Object picture — A particular picture currently being worked on. Especially a complicated picture of particular interest to a user as opposed to a definition or master picture which is to be used as a portion of the object picture.

Older — The older of two blocks is the one with the lowest numbered address, illustrated higher on the page. Since new blocks are taken from the free space in addresses higher numbered than the drawing storage, an older block was usually created sooner.

Picture — A storage device to collect together related drawing parts. A "sheet of paper". Also the lines, points, instances, and constraints, etc., that are drawn in the picture, collectively. Pictures are numbered so that any one may be called to appear on the display. Within the limits of storage, as many pictures as desired may be set up and used.

Point — A specific representation in the ring structure used as an end point for a line segment. Not to be confused with location or spot. Also as a verb, to aim at something with the light pen.

Pointer — A storage register which contains the location of another storage register rather than numerical data. Such a register is said to point to the register whose address it contains.

Pseudo pen location — A location near the axis of the light pen which is used as the "point of the pencil". The pseudo pen location lies exactly on an existing point or line or circle or at the intersection of lines if the pen is aimed at them.

Relocating — Changing the address at which a particular block is stored in memory. Not to be confused with moving.

Result — The single thing which remains after two things have been merged.

Ring — A set of pointers which closes on itself. In Sketchpad all rings point both forward and back. A ring is composed of one hen and many chickens.

Ring structure — The type of storage structure used to represent the drawing's topology. See ring.

Satisfy — See constraint satisfaction.

Scalar — A one component vector whose value can be displayed by a set of digits. For display of the scalar itself a # is used.

Spot — One of the bright dots on the display. Not to be confused with point or location.

Table — A form of storage structure in which successive pieces of information are stored in successive registers in memory. Tables are the "conventional" form of storage. See also list and ring structure.

Termination — The process of taking things out of the moving state. Termination is usually done by giving a flick of the light pen. Pressing "stop" also terminates. Upon termination, merging may take place.

Texts — Lines of textual material typed in and appearing in a standard type style on the picture. Text is treated as a four component variable.

Tie — An attacher.

Value — The particular information stored in the numerical portion of a variable. E.g., the location of a point. Especially the value of a scalar as opposed to the location of the set of digits displaying this value.

Variable — A picture part which contains numerical information. Scalars, points, instances, texts, digits and dummy variables are the only variables at present. Also used to denote a type of variable.

BIBLIOGRAPHY

1. Clark, W. A., Frankovich, J. M., Peterson, H. P., Forgie, J. W. Best, R. L., Olsen, K. H., "The Lincoln TX-2 Computer," Technical Report 6M-4968, Massachusetts Institute of Technology, Lincoln Laboratory, Lexington, Mass., April 1, 1957, Proceedings of the Western Joint Computer Conference, Los Angeles, California, February, 1957.

2. Coons, S. A., Notes on Graphical Input Methods, Memorandum 8436-M-17, Dynamic Analysis and Control Laboratory, Massachusetts Institute of Technology, Department of Mechanical Engineering, Cambridge, Mass., May 4, 1960.

3. Johnston, L. E., A Graphical Input Device and Shape Description Interpretation Routines, Memorandum to Prof. Mann, Massachusetts Institute of Technology, Department of Mechanical Engineering, Cambridge, Mass., May 4, 1960.

4. Lickleder, J. C. R., "Man-Computer Symbiosis," I.R.E. Trans. on Human Factors in Electronics, vol. HFE, pp. 4-10, March 1960.

5. Lickleder, J. C. R., and Clark, W., "On-Line Man-Computer Communication," Proceedings of the Spring Joint Computer Conference, San Francisco, California, May 1-3, 1962, vol. 21, pp. 113-128.

6. Loomis, H. H. Jr., Graphical Manipulation Techniques Using the Lincoln TX-2 Computer, Group Report 51G-0017, Massachusetts Institute of Technology, Lincoln Laboratory, Lexington, Mass., November 10, 1960.

7. Moore, E. F., "On the Shortest Path Through a Maze," Proceedings of the International Symposium on the Theory of Switching, Harvard University, Harvard Annals, vol. 3, pp. 285-292, 1959.

8. Roberts, L. G., Machine Perception of Three Dimensional Solids, Ph.D. Thesis, Massachusetts Institute of Technology, Electrical Engineering Department, Cambridge, Mass., February, 1963.

9. Southwell, R. V., Relaxation Methods in Engineering Science, Oxford University Press, 1940.

10. Vanderburgh, A. Jr., TX-2 Users Handbook, Lincoln Manual No. 45, Massachusetts Institute of Technology, Lincoln Laboratory, Lexington, Mass. July, 1961.

11. Walsh, J. F., and Smith A. F., "Computer Utilization," Interim Engineering Report 6873-IR-10 and 11, Electronic Systems Laboratory, Massachusetts Institute of Technology, Cambridge, Mass., pp. 57-70, November 30, 1959.

12. Handbook for Variplotter Models 205S and 205T, PACE, Electronic Associates Incorporated. Long Branch, New Jersey, June 15, 1959.

Biographical Note

Ivan Edward Sutherland was born on May 16, 1938 in Hastings, Nebraska. After an early childhood near Chicago, he moved to Scarsdale, New York where he graduated from Scarsdale High School. Mr. Sutherland was a George Westinghouse Scholar during his four years at Carnegie Institute of Technology, Pittsburgh, Pennsylvania where he received the Bachelor of Science degree in Electrical Engineering in June 1959. While at Carnegie he twice won the American Institute of Electrical Engineers Student Prize Paper Contest for District 2 (1958 and 1959). As a graduate student he held a National Science Foundation Fellowship for three years (1959 to 1962). He received the Master of Science degree in Electrical Engineering from California Institute of Technology, Pasadena, California in June 1960. From September 1960 to December 1962, Mr. Sutherland was associated with the Research Laboratory of Electronics at Massachusetts Institute of Technology first as a full-time doctoral student and then as a research assistant during the fall semester of 1962. During the summers of 1960, 1961 and 1962 he was a Staff Member of the MIT Lincoln Laboratory.

Mr. Sutherland is a coauthor of "An Electro-Mechanical Model of Simple Animals," (Computers and Automation, February 1958) and is the author of "Stability in Steering Control," (Electrical Engineering, April 1960). He is a member of Sigma Xi, Tau Beta Pi, Eta Kappa Nu, and Pi Mu Epsilon. Mr. Sutherland belongs to the Institute of Electrical and Electronics Engineers and the American Society of Mechanical Engineers.