MASS. INST. OF TECHNOLOGY
APR 25 1963
LIBRARY

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 redacted

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

Signature redacted

Certified by.....
Thesis Supervisor

Signature redacted

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

Meta Platforms, Inc.
Exhibit 1140
Page 001

038

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

Meta Platforms, Inc.
Exhibit 1140
Page 002

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

Meta Platforms, Inc.
Exhibit 1140
Page 003

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.

Meta Platforms, Inc.
Exhibit 1140
Page 004

TABLE OF CONTENTS

Abstract 2

Acknowledgements 4

Table of Contents 5

List of Figures 6

Chapter I. Introduction 8

Chapter II. History of Sketchpad 24

Chapter III. Ring Structure 34

Chapter IV. Light Pen 54

Chapter V. Display Generation 67

Chapter VI. Recursive Functions 87

Chapter VII. Building a Drawing, The Copy Function 102

Chapter VIII. Constraint Satisfaction 110

Chapter IX. Examples and Conclusions 120

Appendix A. Constraint Descriptions 141

B. Push Button Controls 144

C. Structure of Storage Blocks 147

D. Ring Operation MACRO Instructions 152

E. Proposal for an Incremental Curve Drawing Display .. 154

F. Mathematics of Least Mean Square Fit 161

G. A Brief Description of TX-2 164

Glossary 170

Bibliography 175

Biographical Note 176

Meta Platforms, Inc.
Exhibit 1140
Page 005

LIST OF FIGURES

Figure 1.1. Hexagonal Pattern 10

1.2. TX-2 Operating Area - Sketchpad in Use 11

1.3. Plotter Used with Sketchpad 12

1.4. Line and Circle Drawing 14

1.5. Illustrative Example 15

1.6. Four Positions of Linkage 20

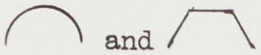
1.7.  and on Same Lattice 20

Figure 3.1. N-Component Elements 36

3.2. Basic Ring Structure 41

3.3. Line Segment and End Points 43

3.4. Zero and One Member Rings 43

3.5. Fresh Point Block 46

3.6. Compacting the Ring Structure 47

3.7. Instances Generic Block 51

3.8. Generic Structure 52

Figure 4.1. Light Pen 56

4.2. Construction of Light Pen 56

4.3. Predictive Pen Tracking 58

4.4. Displays for Pen Tracking 58

4.5. Address in Display Register 61

4.6. Operation of Pseudo Pen Location 61

Meta Platforms, Inc.
Exhibit 1140
Page 006

Figure 5.1. Twinkled Display 69
5.2. Coordinate Systems 73
5.3. Display of Constraints 84
5.4. Display of Scalar and Digits 84

Figure 6.1. Applying Two Constraints Indirectly to Two Lines 94

Figure 7.1. Definitions to Copy 106

Figure 9.1. Zig - Zag for Delay Line 122
9.2. BCD Encoder for Clock 122
9.3. Three Bar Linkage 125
9.4. Conic Drawing Linkage 125
9.5. Dimension Lines 128
9.6. Truss Under Load 128
9.7. Cantilever and Arch Bridges 131
9.8. Winking Girl and Components 133
9.9. Girl Traced from Photograph 134
9.10. Girl with Features Changed 135
9.11. Circuit Diagrams 137

Figure E.1. DDA for Drawing Lines 155
E.2. DDA for Upright Conics 157
E.3. DDA for the General Conic 159

Meta Platforms, Inc.
Exhibit 1140
Page 007

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

Meta Platforms, Inc.
Exhibit 1140
Page 008

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.

Meta Platforms, Inc.
Exhibit 1140
Page 009

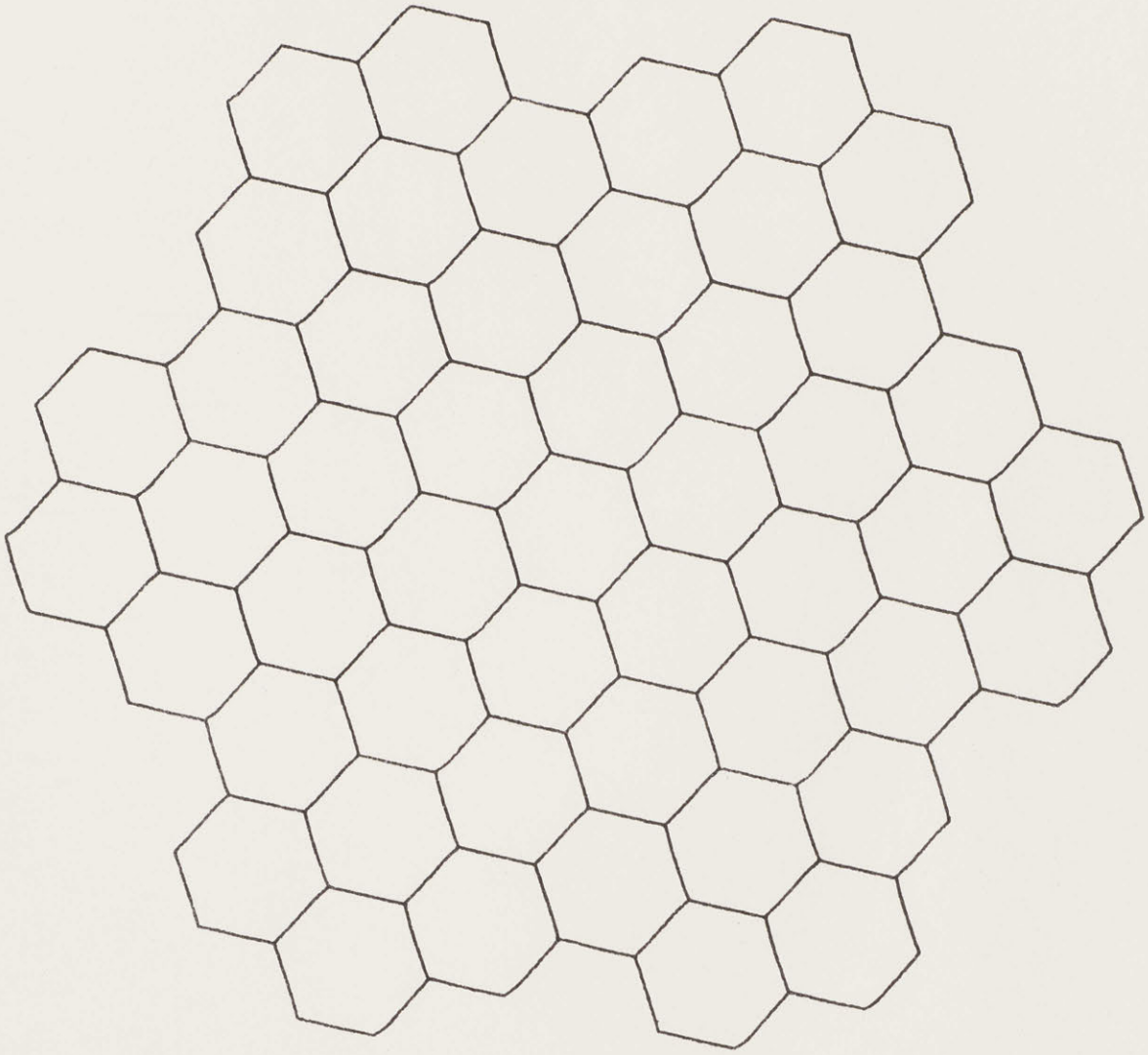


FIGURE 1.1. HEXAGONAL PATTERN

Meta Platforms, Inc.
Exhibit 1140
Page 010

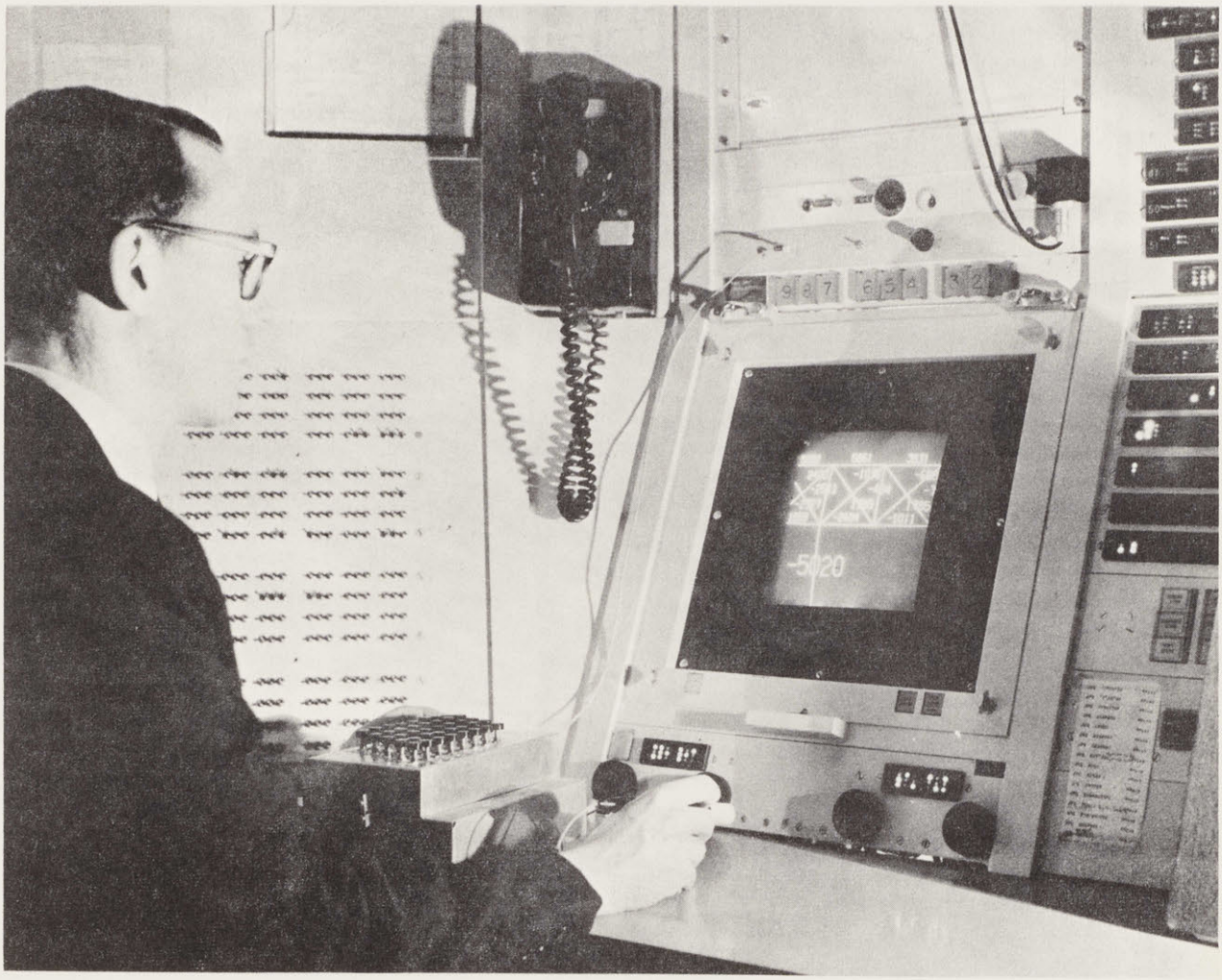


FIGURE 1.2. TX-2 OPERATING AREA - SKETCHPAD IN USE.
On the display can be seen part of a bridge similar to that of Figure 9.6. The Author is holding the Light pen. The push buttons used to control specific drawing functions are on the box in front of the Author. Part of the bank of toggle switches can 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.

Meta Platforms, Inc.
Exhibit 1140
Page 011

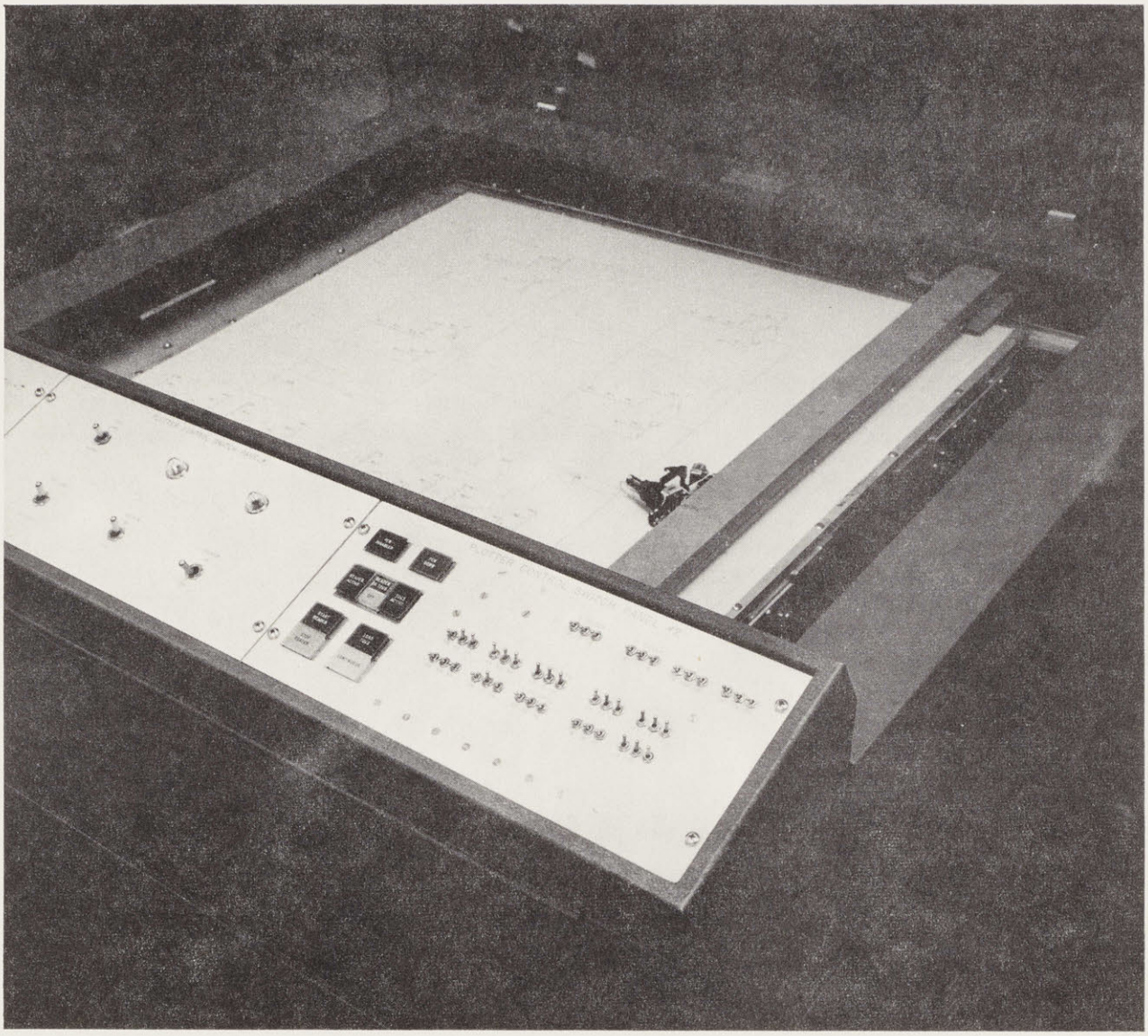


FIGURE 1.3. PLOTTER USED WITH SKETCHPAD
A digital and analog control system makes the plotter draw straight lines and circles either under direct control of the TX-2 or off-line from punched paper tape.

Meta Platforms, Inc.
Exhibit 1140
Page 012

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.

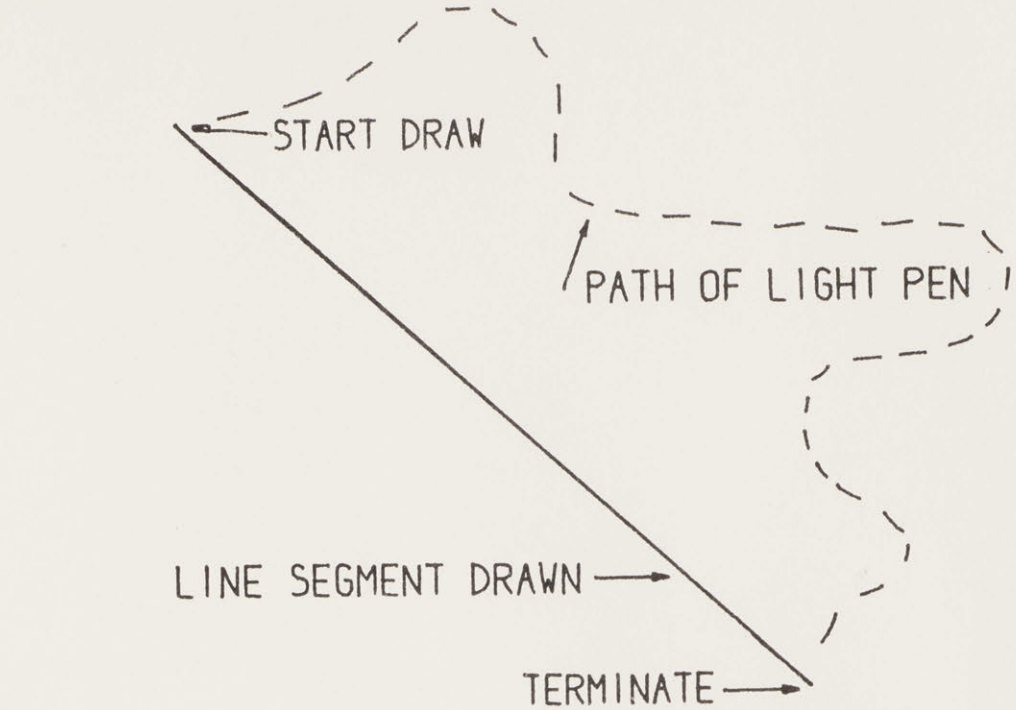
To make the hexagon regular, we can inscribe it in a circle. To draw the circle we place the light pen where the center is to be and press the button "circle center", leaving behind a center point. Now, choosing a point on the circle (which fixes the radius,) we press the button "draw" again, this time 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.

Meta Platforms, Inc.
Exhibit 1140
Page 013



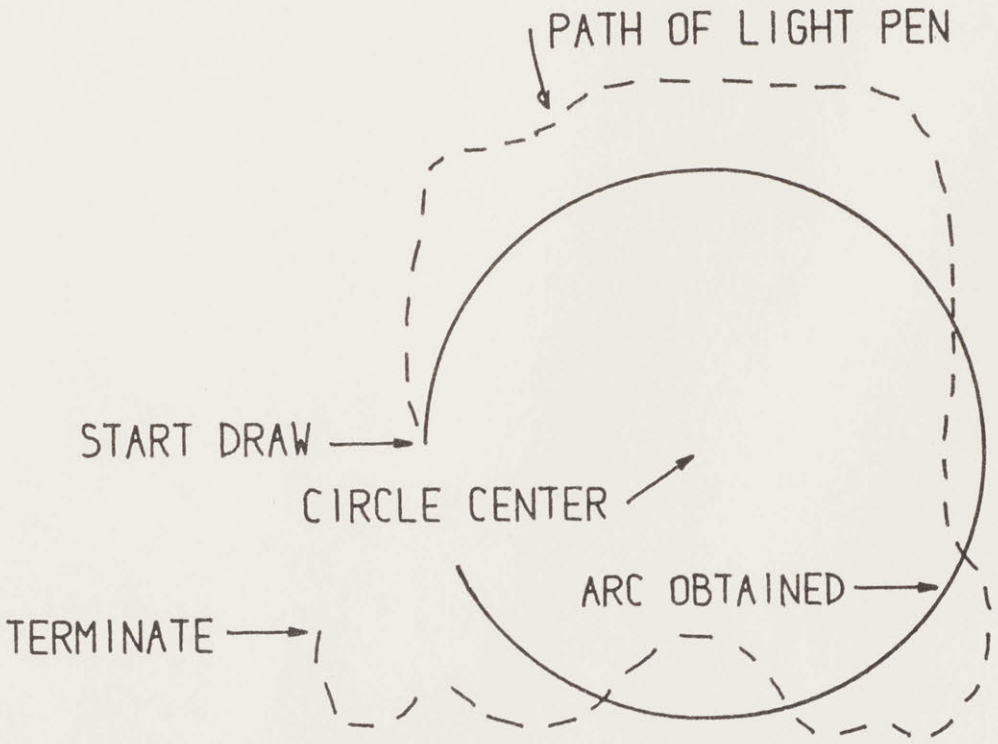
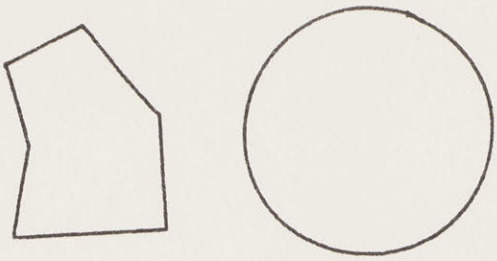


FIGURE 1.4.
LINE AND CIRCLE DRAWING

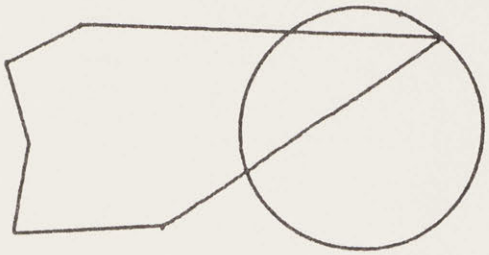
Meta Platforms, Inc.
Exhibit 1140
Page 014



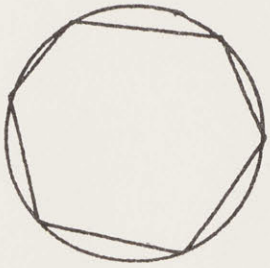
A. SIX SIDED FIGURE



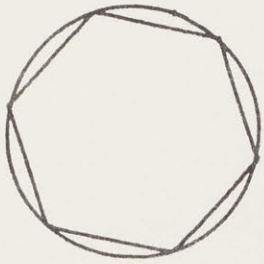
B. TO BE INSCRIBED IN CIRCLE



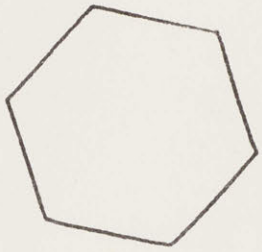
C. BY MOVING EACH CORNER



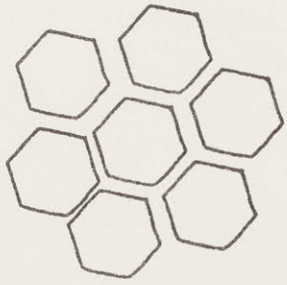
D. ON TO CIRCLE



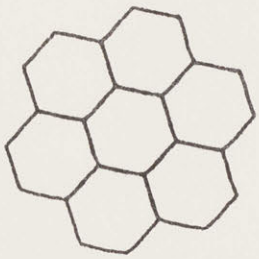
E. MAKE SIDES EQUAL



F. ERASE CIRCLE



G. CALL 7 HEXAGONS



H. JOIN CORNERS

FIGURE 1.5. ILLUSTRATIVE EXAMPLE

Meta Platforms, Inc.
Exhibit 1140
Page 015

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

Meta Platforms, Inc.
Exhibit 1140
Page 016

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.

Meta Platforms, Inc.
Exhibit 1140
Page 017

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

Meta Platforms, Inc.
Exhibit 1140
Page 018

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.

Meta Platforms, Inc.
Exhibit 1140
Page 019

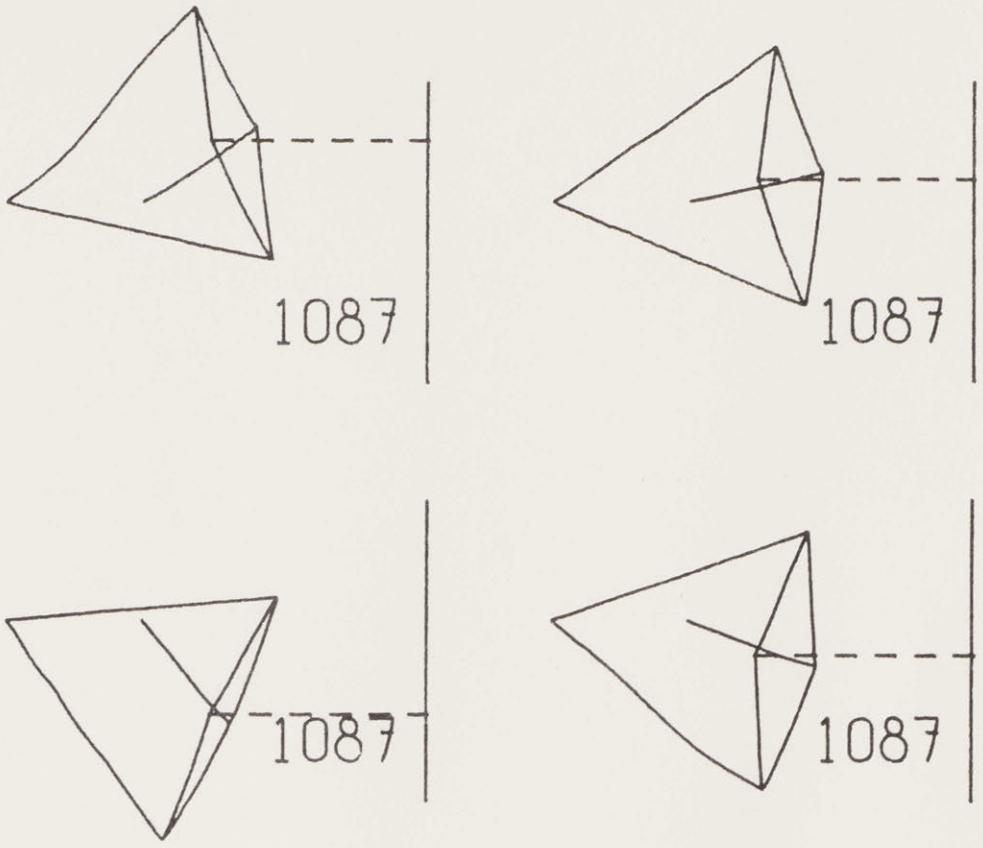


FIGURE 1.6.
FOUR POSITIONS OF LINKAGE
NUMBER SHOWS LENGTH OF DOTTED LINE

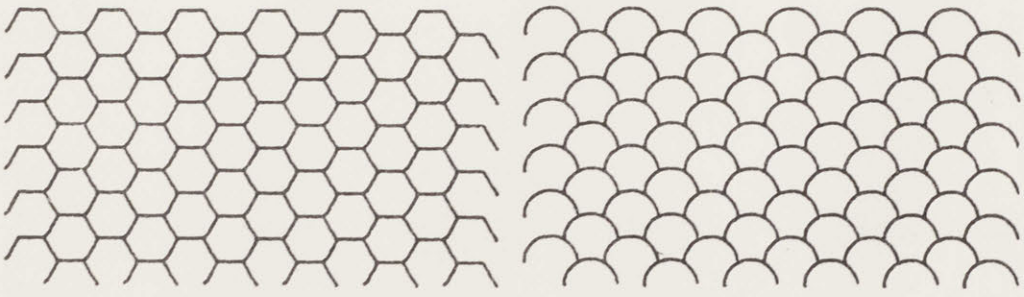


FIGURE 1.7.
⌒ AND ⌒ ON SAME LATTICE

Meta Platforms, Inc.
Exhibit 1140
Page 020

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

Meta Platforms, Inc.
Exhibit 1140
Page 021

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

Meta Platforms, Inc.
Exhibit 1140
Page 022

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.

Meta Platforms, Inc.
Exhibit 1140
Page 023

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

Meta Platforms, Inc.
Exhibit 1140
Page 024

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

Meta Platforms, Inc.
Exhibit 1140
Page 025

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.

Meta Platforms, Inc.
Exhibit 1140
Page 026

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.

Meta Platforms, Inc.
Exhibit 1140
Page 027

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

Meta Platforms, Inc.
Exhibit 1140
Page 028

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

Meta Platforms, Inc.
Exhibit 1140
Page 029

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, $-1 \geq x \geq 1$, 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

Meta Platforms, Inc.
Exhibit 1140
Page 030

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

Meta Platforms, Inc.
Exhibit 1140
Page 031

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

Meta Platforms, Inc.
Exhibit 1140
Page 032

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.

Meta Platforms, Inc.
Exhibit 1140
Page 033

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

Meta Platforms, Inc.
Exhibit 1140
Page 034

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 PVALth and PVAL+1st (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

Meta Platforms, Inc.
Exhibit 1140
Page 035

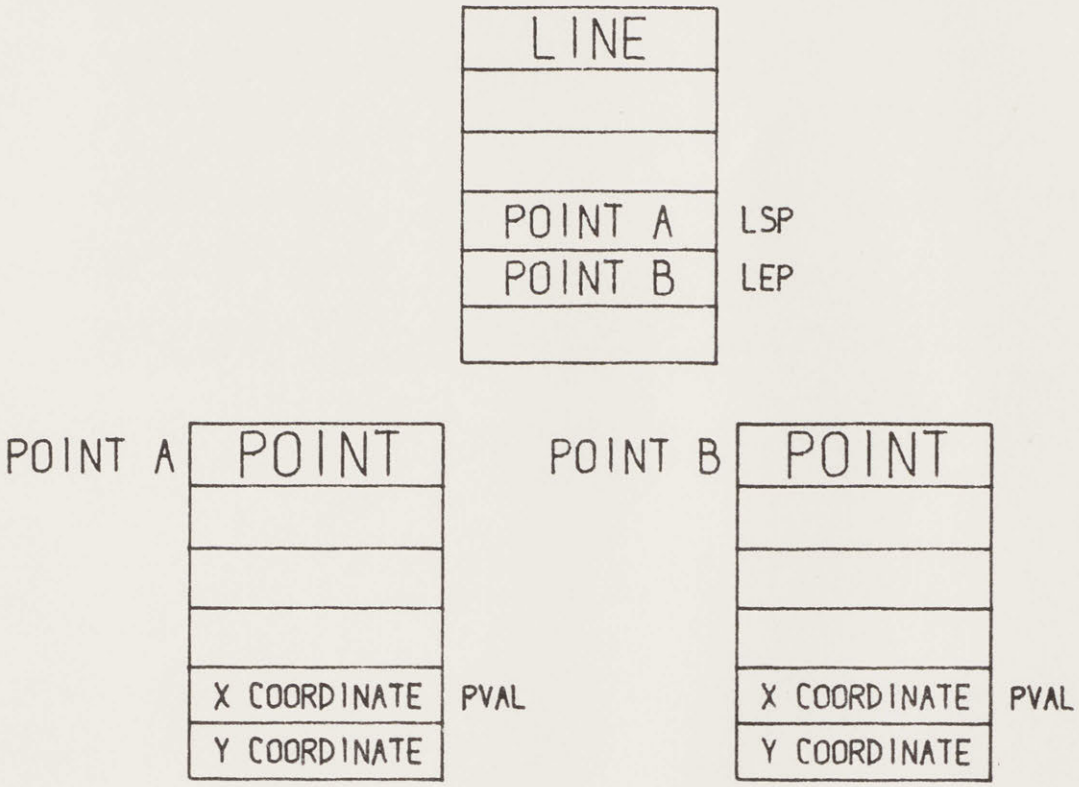


FIGURE 3.1. N-COMPONENT ELEMENTS

Meta Platforms, Inc.
Exhibit 1140
Page 036

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.

Meta Platforms, Inc.
Exhibit 1140
Page 037

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

Meta Platforms, Inc.
Exhibit 1140
Page 038

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

Meta Platforms, Inc.
Exhibit 1140
Page 039

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.

Meta Platforms, Inc.
Exhibit 1140
Page 040

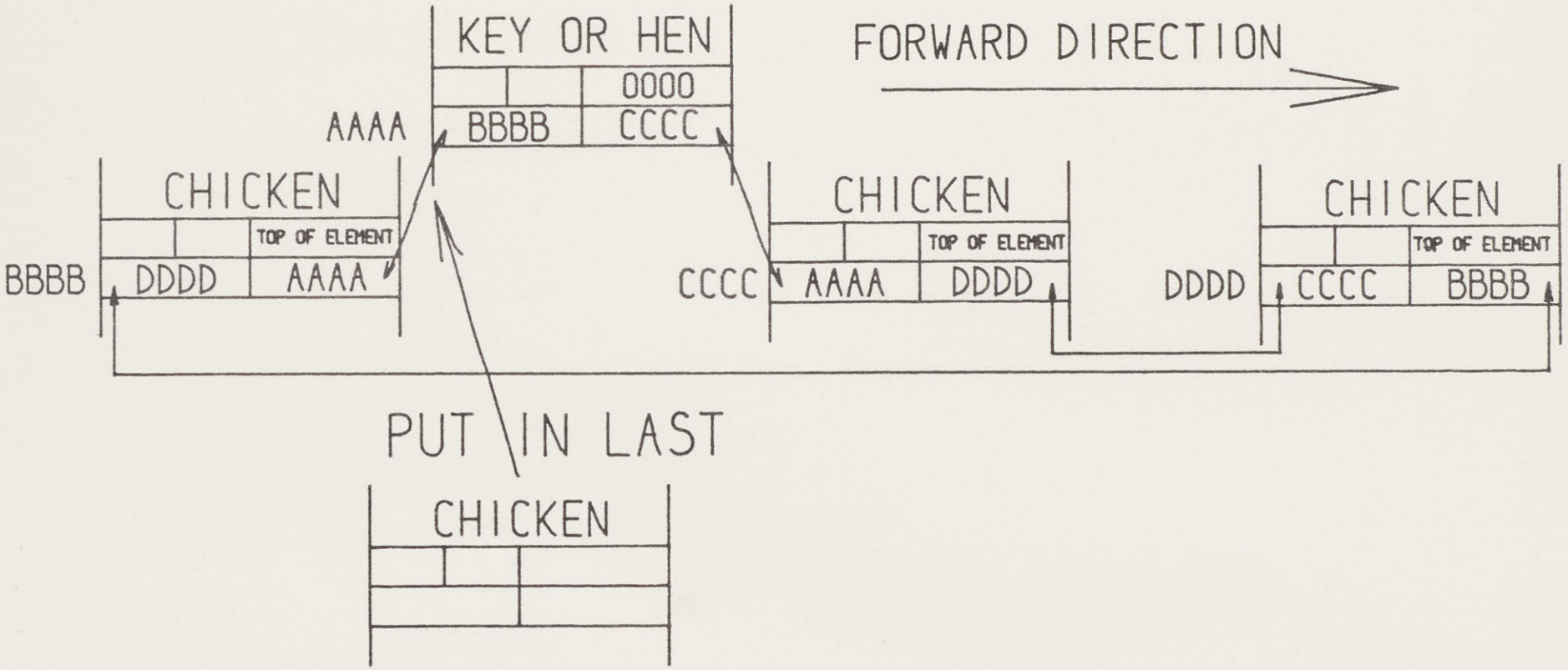


FIGURE 3.2. BASIC RING STRUCTURE

Meta Platforms, Inc.
Exhibit 1140
Page 041

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."

Meta Platforms, Inc.
Exhibit 1140
Page 042

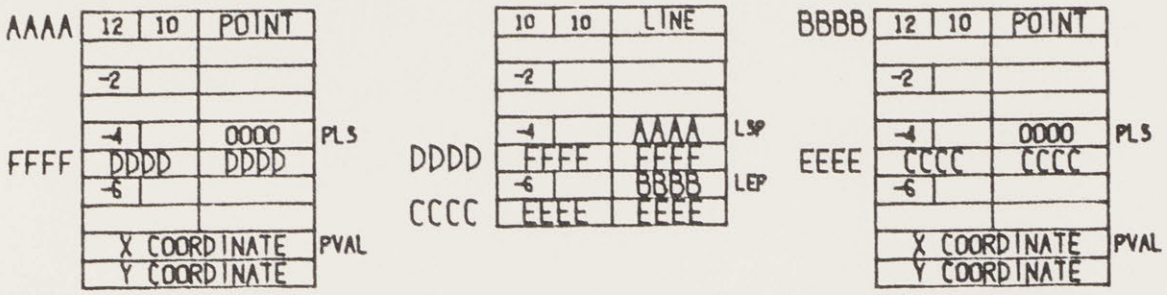
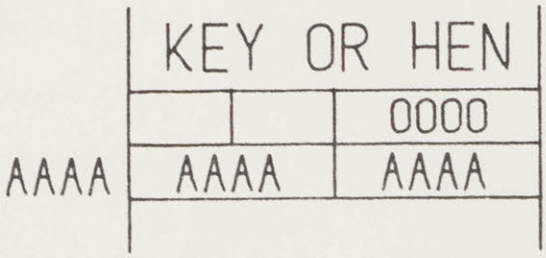


FIGURE 3.3.
LINE SEGMENT AND END POINTS
IN RING STRUCTURE NOTATION



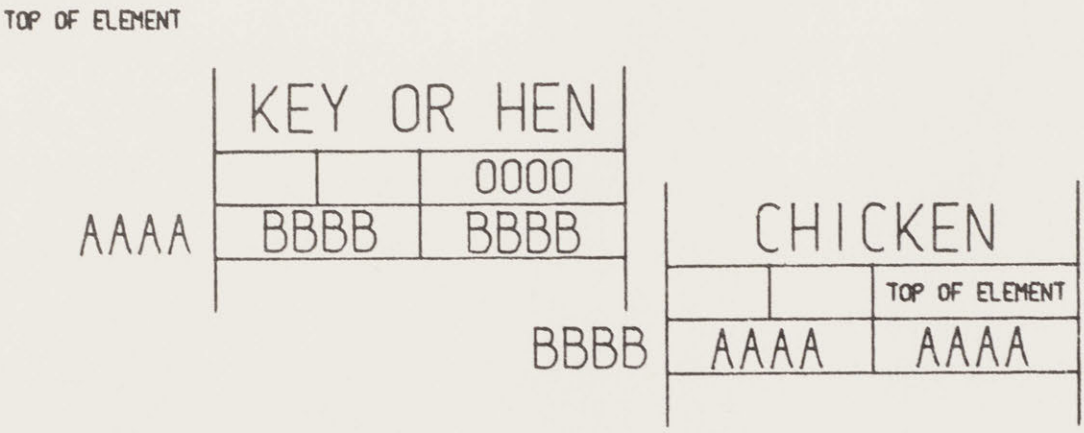


FIGURE 3.4.
ZERO AND ONE MEMBER RINGS

Meta Platforms, Inc.
Exhibit 1140
Page 043

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

Meta Platforms, Inc.
Exhibit 1140
Page 044

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.

Meta Platforms, Inc.
Exhibit 1140
Page 045

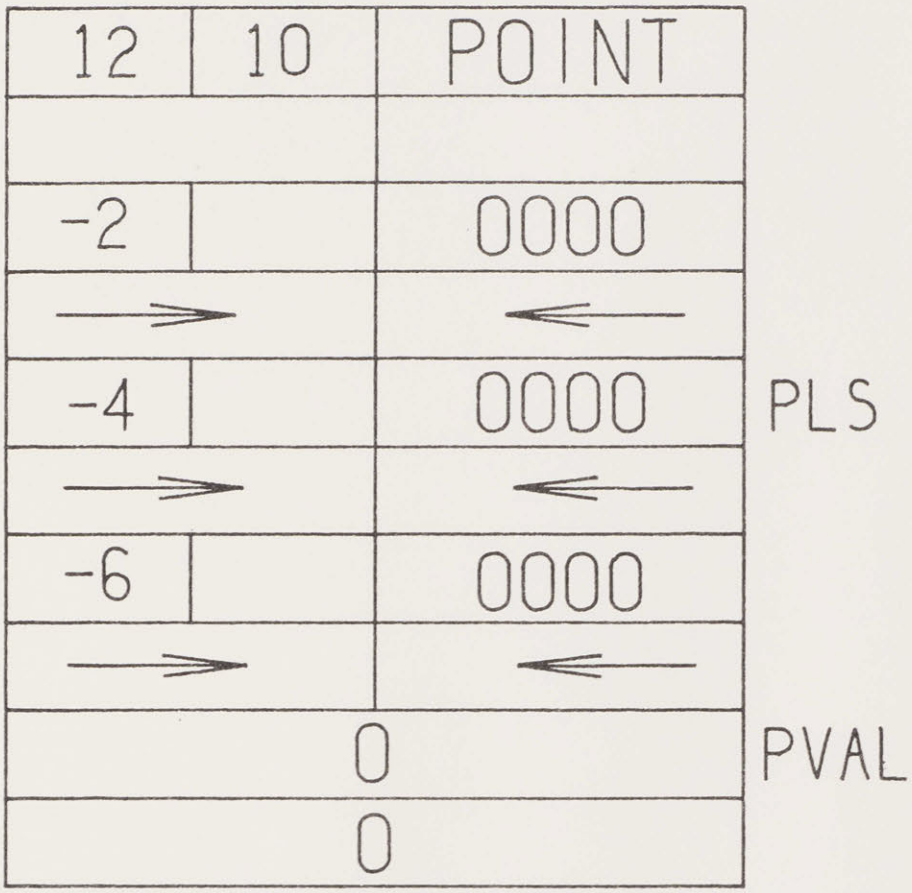


FIGURE 3.5. FRESH POINT BLOCK

Meta Platforms, Inc.
Exhibit 1140
Page 046

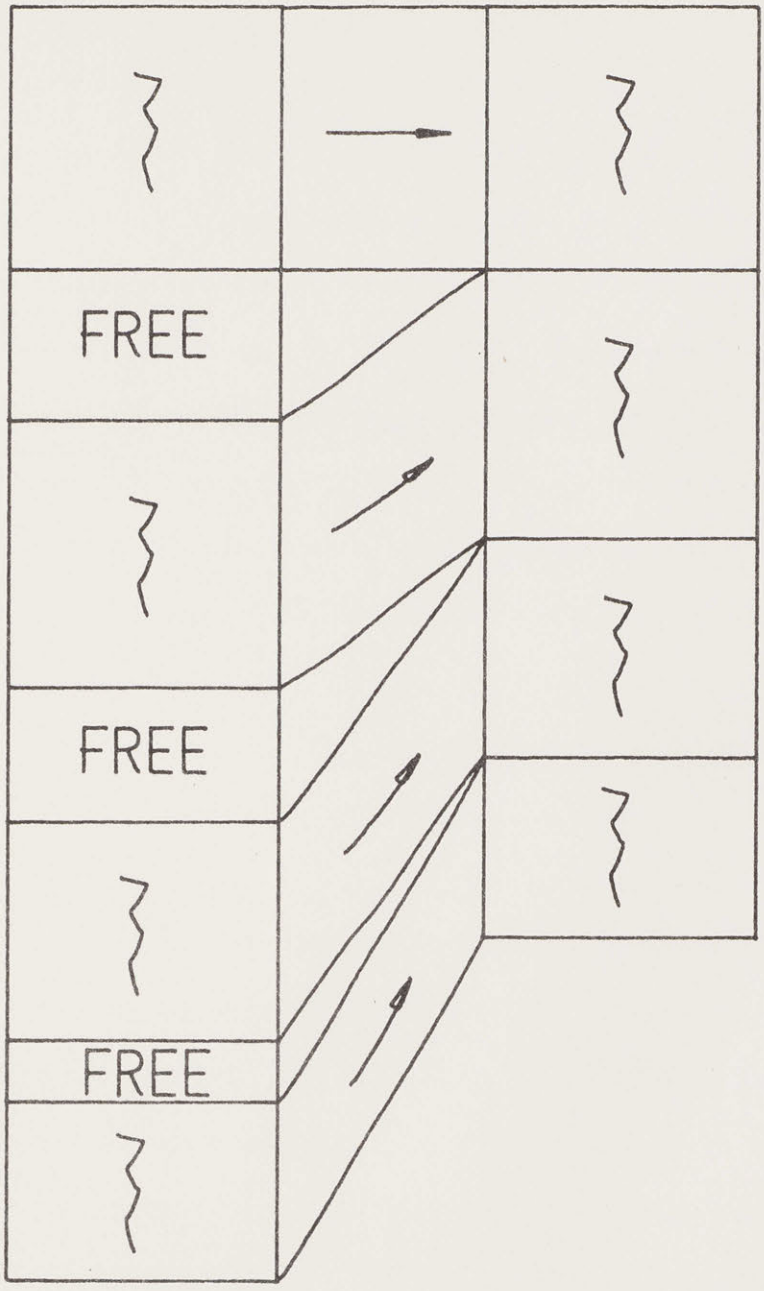


FIGURE 3.6.
COMPACTING THE RING STRUCTURE

Meta Platforms, Inc.
Exhibit 1140
Page 047

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.

Meta Platforms, Inc.
Exhibit 1140
Page 048

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

Meta Platforms, Inc.
Exhibit 1140
Page 049

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

Meta Platforms, Inc.
Exhibit 1140
Page 050

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

Meta Platforms, Inc.
Exhibit 1140
Page 051

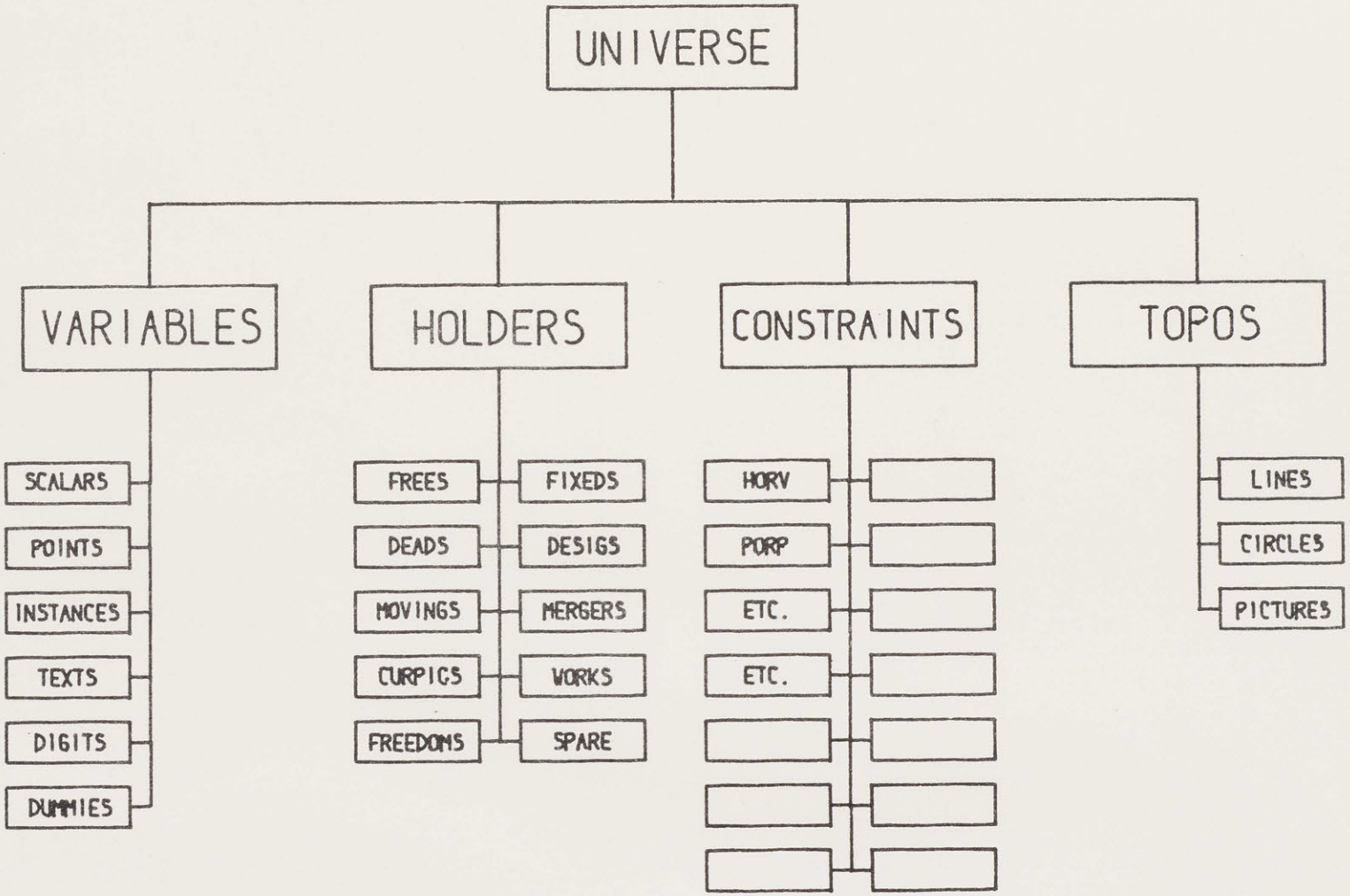


FIGURE 3.8. GENERIC STRUCTURE

Meta Platforms, Inc.
Exhibit 1140
Page 052

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.

Meta Platforms, Inc.
Exhibit 1140
Page 053

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

Meta Platforms, Inc.
Exhibit 1140
Page 054

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.

Meta Platforms, Inc.
Exhibit 1140
Page 055

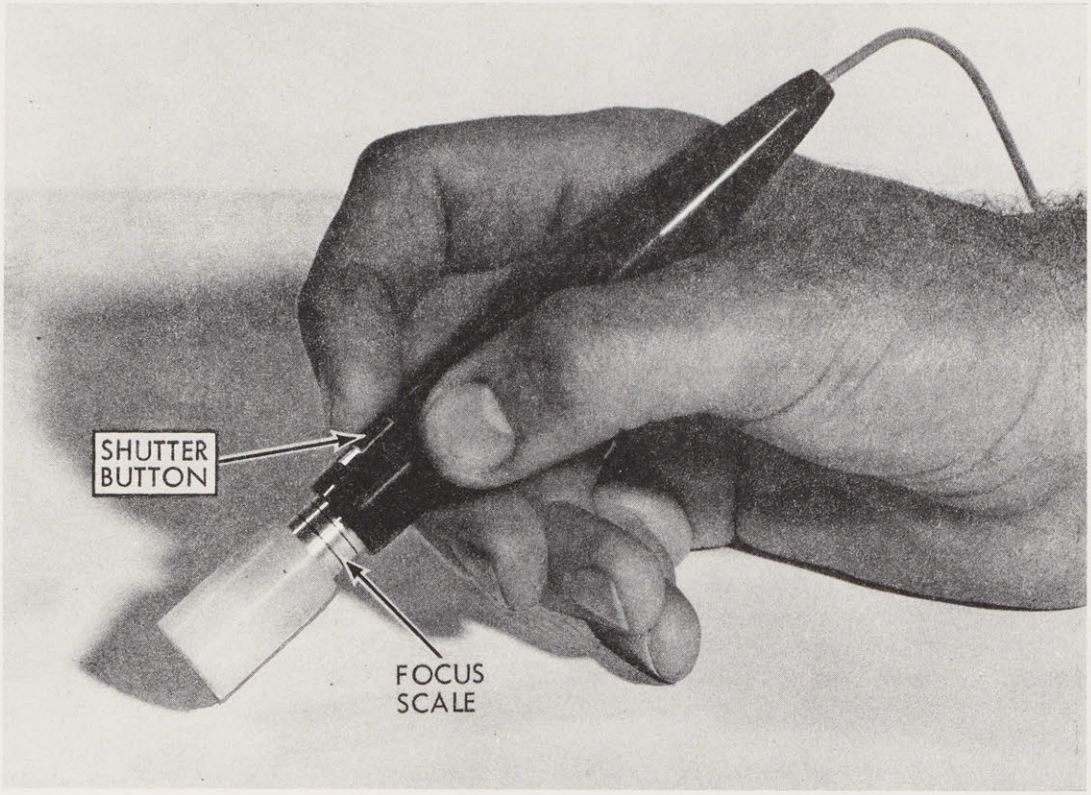


FIGURE 4.1. LIGHT PEN.
Courtesy of MIT
Electronic Systems Laboratory.

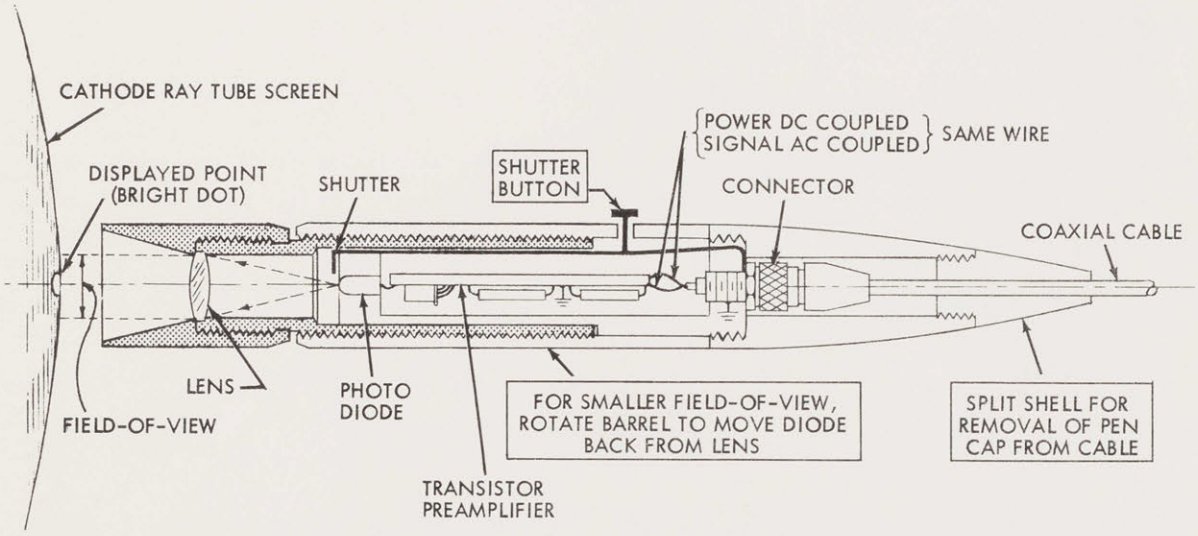


FIGURE 4.2. CONSTRUCTION OF LIGHT PEN.
Drawing courtesy of Electronic Systems Laboratory. This drawing was made by conventional methods.

Meta Platforms, Inc.
Exhibit 1140
Page 056

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.

Meta Platforms, Inc.
Exhibit 1140
Page 057

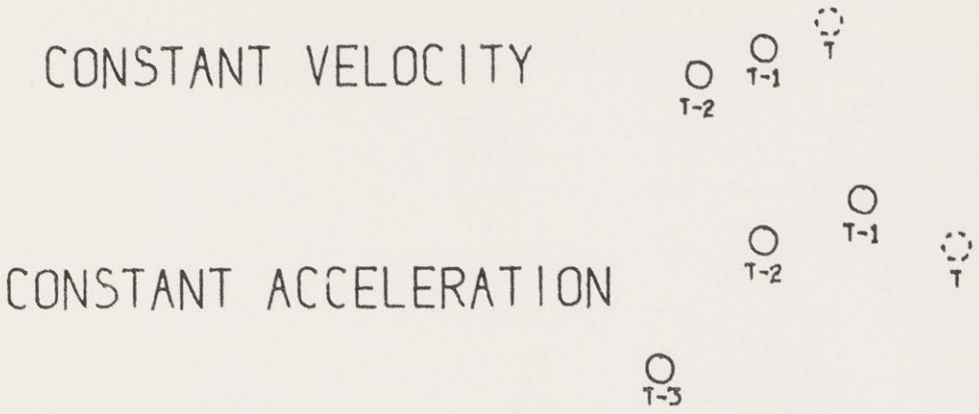


FIGURE 4.3.
PREDICTIVE PEN TRACKING



FIGURE 4.4.
DISPLAYS FOR PEN TRACKING

Meta Platforms, Inc.
Exhibit 1140
Page 058

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

Meta Platforms, Inc.
Exhibit 1140
Page 059

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

Meta Platforms, Inc.
Exhibit 1140
Page 060

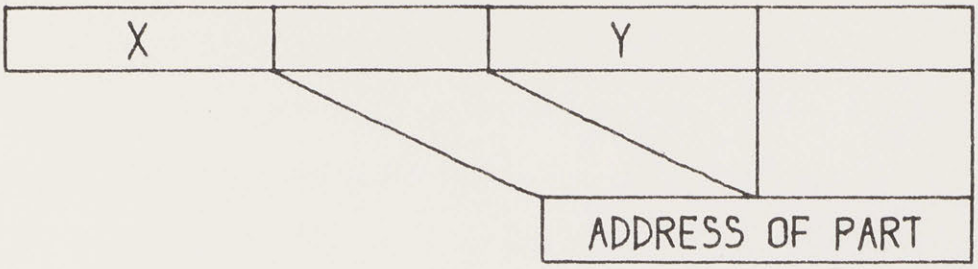


FIGURE 4.5.
ADDRESS IN DISPLAY REGISTER

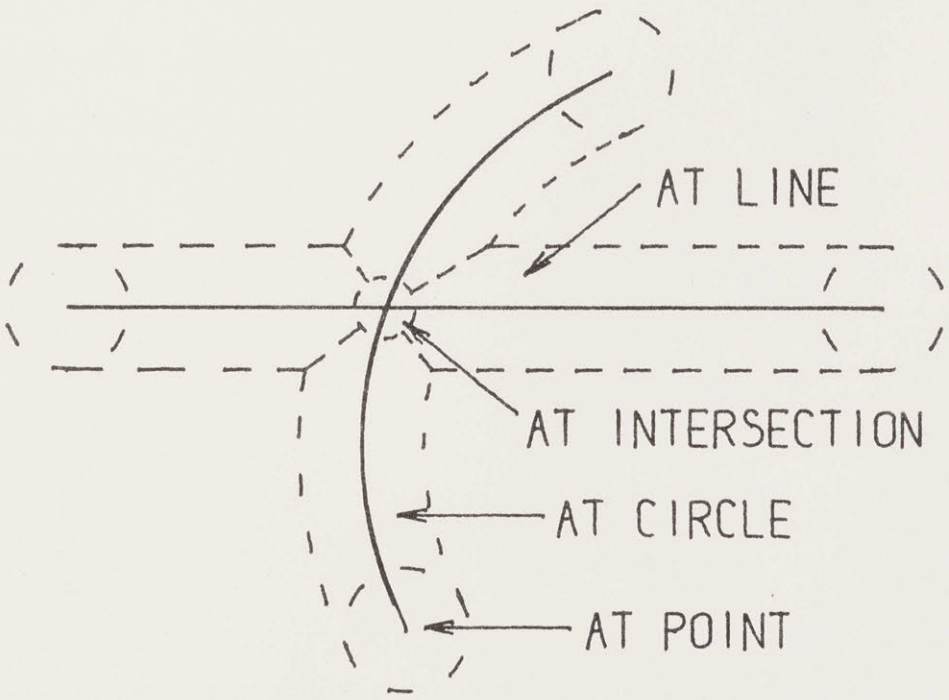


FIGURE 4.6.
OPERATION OF PSEUDO PEN LOCATION

Meta Platforms, Inc.
Exhibit 1140
Page 061

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

Meta Platforms, Inc.
Exhibit 1140
Page 062

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

Meta Platforms, Inc.
Exhibit 1140
Page 063

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

Meta Platforms, Inc.
Exhibit 1140
Page 064

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

Meta Platforms, Inc.
Exhibit 1140
Page 065

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.

Meta Platforms, Inc.
Exhibit 1140
Page 066

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

Meta Platforms, Inc.
Exhibit 1140
Page 067

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.

Meta Platforms, Inc.
Exhibit 1140
Page 068

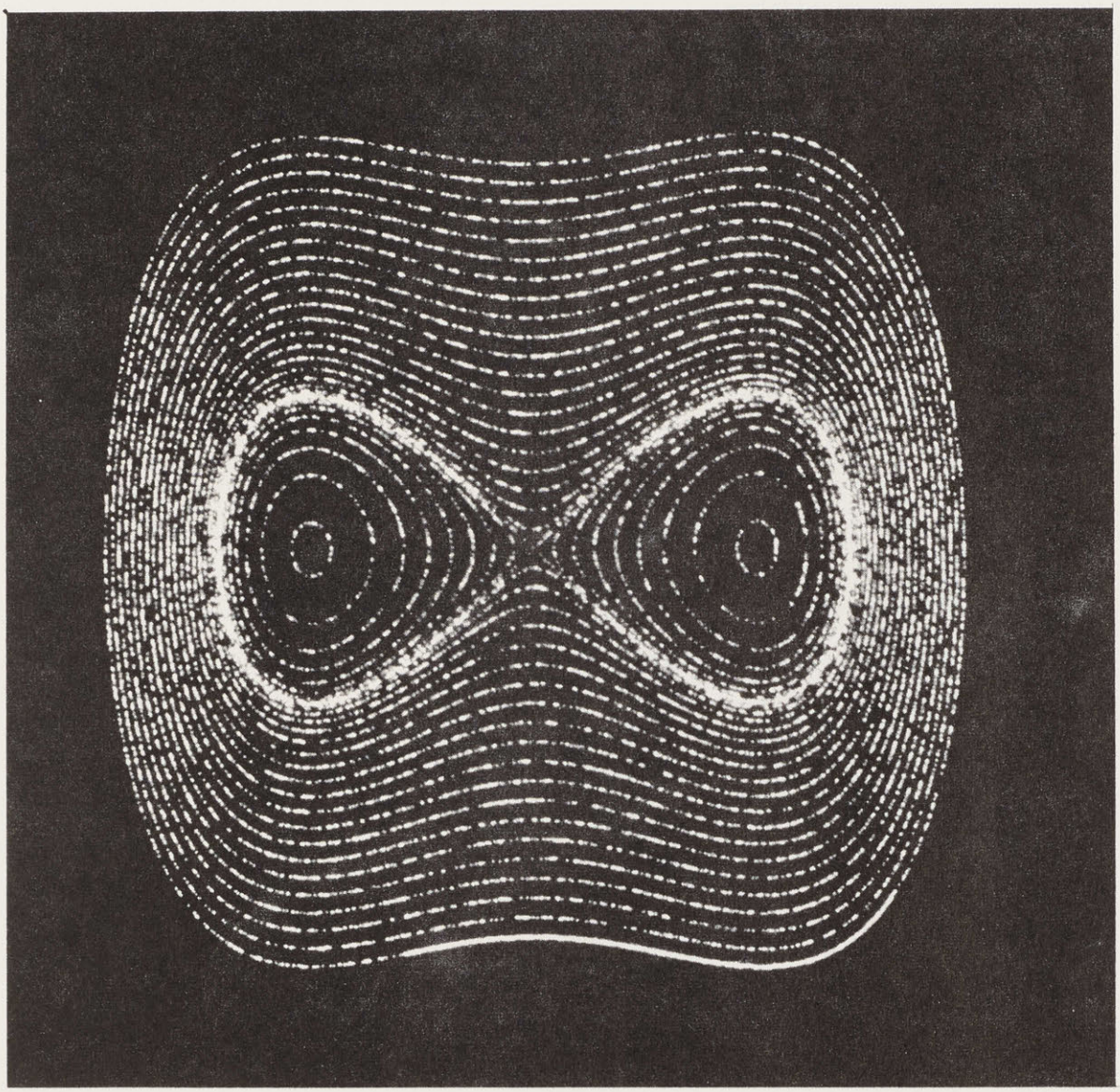


FIGURE 5.1. TWINKLING DISPLAY.

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.

Meta Platforms, Inc.
Exhibit 1140
Page 069

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

Meta Platforms, Inc.
Exhibit 1140
Page 070

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

Meta Platforms, Inc.
Exhibit 1140
Page 071

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.

Meta Platforms, Inc.
Exhibit 1140
Page 072

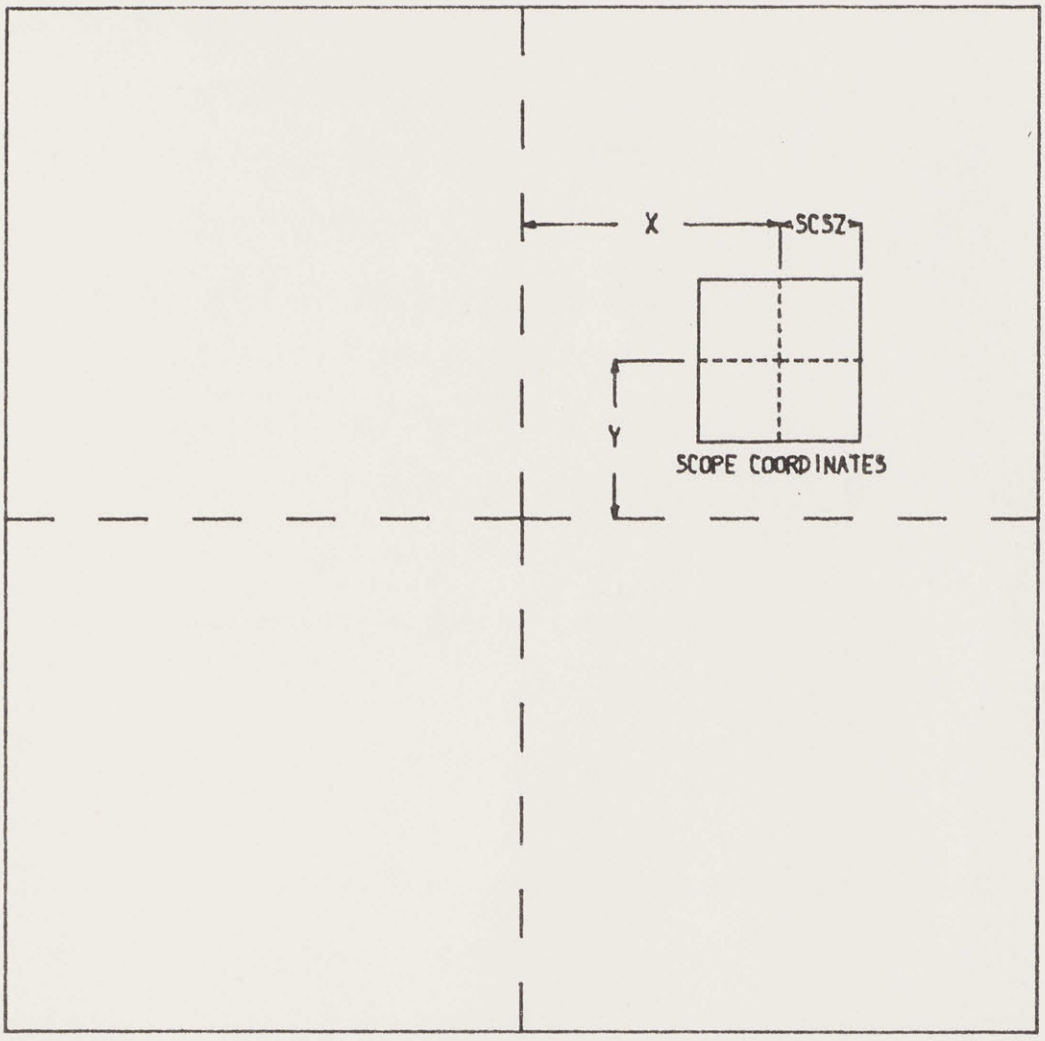


FIGURE 5.2. COORDINATE SYSTEMS

Meta Platforms, Inc.
Exhibit 1140
Page 073

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.

Meta Platforms, Inc.
Exhibit 1140
Page 074

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?

Meta Platforms, Inc.
Exhibit 1140
Page 075

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:

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

Meta Platforms, Inc.
Exhibit 1140
Page 076

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

Meta Platforms, Inc.
Exhibit 1140
Page 077

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:

$$x_i = x_{i-1} + \frac{1}{R}(y_{i-1} - y_c)$$
$$y_i = y_{i-1} - \frac{1}{R}(x_{i-1} - x_c) \qquad (5\text{-}3)$$

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:

$$x_i = \frac{R}{\sqrt{1 + R^2}}\left\{x_{i-1} + \frac{1}{R}(y_{i-1} - y_c)\right\}$$
$$y_i = \frac{R}{\sqrt{1 + R^2}}\left\{y_{i-1} - \frac{1}{R}(x_{i-1} - x_c)\right\} \qquad (5\text{-}4)$$

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:

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

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

Meta Platforms, Inc.
Exhibit 1140
Page 078

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.

Meta Platforms, Inc.
Exhibit 1140
Page 079

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.

Meta Platforms, Inc.
Exhibit 1140
Page 080

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.

Meta Platforms, Inc.
Exhibit 1140
Page 081

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

Meta Platforms, Inc.
Exhibit 1140
Page 082

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

Meta Platforms, Inc.
Exhibit 1140
Page 083

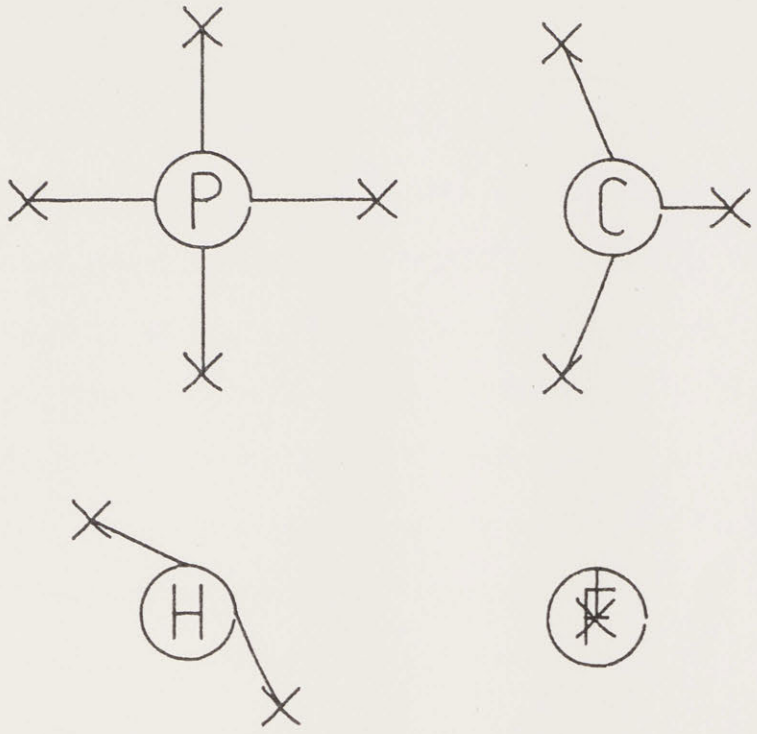


FIGURE 5.3.
DISPLAY OF CONSTRAINTS

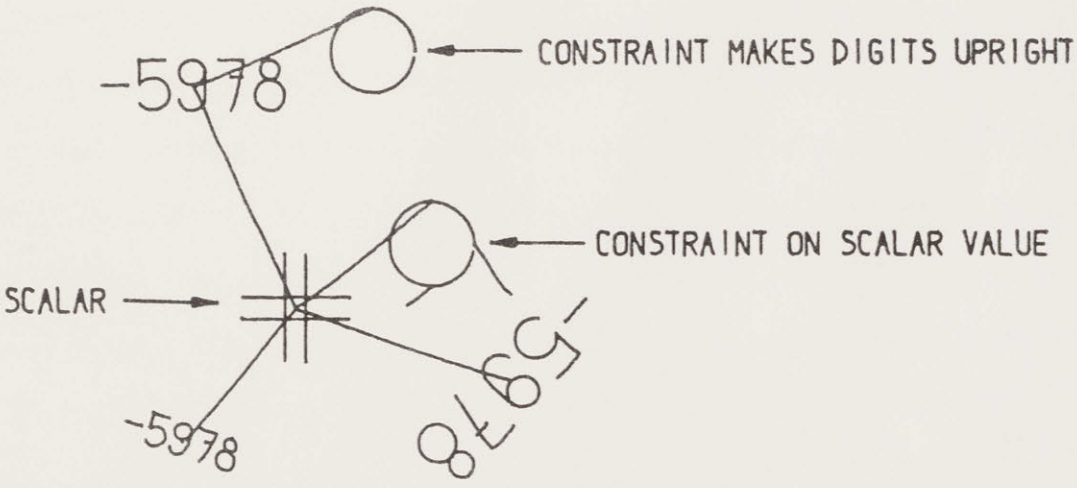


FIGURE 5.4.
DISPLAY OF SCALAR AND DIGITS

Meta Platforms, Inc.
Exhibit 1140
Page 084

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

Meta Platforms, Inc.
Exhibit 1140
Page 085

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.

Meta Platforms, Inc.
Exhibit 1140
Page 086

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.

Meta Platforms, Inc.
Exhibit 1140
Page 087

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

Meta Platforms, Inc.
Exhibit 1140
Page 088

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.

Meta Platforms, Inc.
Exhibit 1140
Page 089

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

Meta Platforms, Inc.
Exhibit 1140
Page 090