

09. [Introduction]

Sketchpad

A Man-Machine Graphical Communication System

The Sketchpad system—developed by Ivan Sutherland on a computer built for military radar—is the graphical ancestor of today’s human-computer interaction and computer graphics, and much of new media in general. In the early 1960s this system not only let people draw on a computer display, it also embodied a crucial conceptual step: considering the screen as more than a poor replacement for a piece of paper.

Sketchpad’s most familiar features to us today are the basic vocabulary of computer drawing—tools such as rubber-band lines (that start with one point, and change the size and direction of the line segment until the next point is defined) and circles that are drawn based on their own constraints, so that the path of drawing need not be anything like a circle. These features made Sketchpad the first direct-manipulation interface (◊33), the first interface to use constraints, and (as Susan Brennan argues) the first *conversational* interface.

But something we rarely consider about computer drawings, something which Sketchpad first demonstrated, is that these images are *objects*, and, as such, can be manipulated, constrained, instantiated, represented iconically, copied, and recursively operated upon, even recursively merged. Because Sketchpad implemented all of these things, it laid the foundation for modern object-oriented programming and graphic user interfaces.

Sutherland describes how the building up of objects and constraints within Sketchpad can allow work on cantilever and arch bridge designs, with appropriate force calculations; how linkages can be used to produce simple computer animations; and how electrical circuit diagrams can be drawn and reused. In this way, through their work process, users can assemble complex objects out of previous constructions, from their personal library of objects. When changes are made to improve or otherwise alter earlier work, these changes are propagated automatically. While Vannevar Bush had suggested a similar strategy with scholarly information (◊02), and basic code-reuse was certainly not new to programmers, Sutherland implemented this in the area of *dynamic media*. Alan Kay (◊26) followed up on these ideas, creating the object-oriented programming language Smalltalk and user-empowering media environment of the Dynabook—but in our current new media landscape these aspects of Sutherland’s creation are seldom as fully realized.

—NWF

Sketchpad also moved beyond paper by allowing the user to work at any of 2000 levels of magnification—enabling the creation of projects that, in physical media, would either be unwieldly large or require detail work at an impractically small size.

Further Reading

Baecker Ronald M., Jonathan Grudin, William A. S. Buxton, and Saul Greenberg. “A Historical and Intellectual Perspective.” In *Readings in Human-Computer Interaction: Toward the Year 2000*. 2d ed. San Mateo, Calif.: Morgan Kaufman, 1995.

Brennan, Susan E. “Conversation as Direct Manipulation: An Iconoclastic View.” In *The Art of Human-Computer Interface Design* 393–404, edited by Brenda Laurel. Reading, Mass.: Addison Wesley, 1990.

Nelson, Theodor H. “Sutherland’s Sketchpad.” In *Computer Lib / Dream Machines*, page DM23. Self-published, 1974.

In “Conversation as Direct Manipulation” Susan Brennan writes: “Appropriately enough, the first great conversational human/computer interface was also the first great direct manipulation one. Ivan Sutherland’s Sketchpad enabled a user and a computer ‘to converse rapidly through the medium of line drawings.’ This style of interaction was primarily graphical, yet it exhibited some of the important features of human conversation. A user conversed with Sketchpad by pointing. The system responded by updating the drawing immediately, so that the relationship between the user’s action and the graphical display was clear. In fact, because the feedback was so timely and relevant, it could be considered analogous to *backchannels*, or secondary speech in human/human communication” (394–395). Of course, while Brennan identifies salient aspects of conversation in Sketchpad, she is writing in the technological era when “conversation” was necessarily a metaphor for interaction. Conversational agents today implement conversation in a more literal way by allowing multi-modal conversations based on speech and similar to those found in human-human discourse.

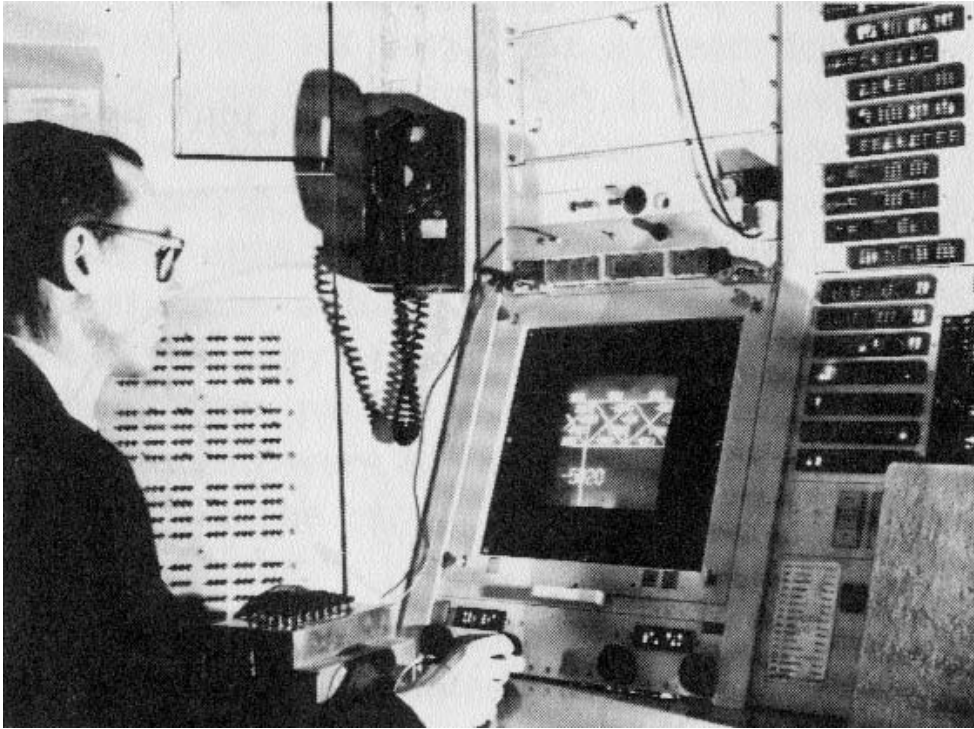


Figure 9.1. TX-2 operating area—Sketchpad in use. On the display can be seen part of a bridge similar to those of Figure 9.15. The Author is holding the light pen. The push buttons “draw,” “move,” etc., are on the box in front of the Author. Part of the bank of toggle switches can be seen behind the Author. The size and position of the part of the total picture seen on the display are controlled by the four black knobs just above the tables.

Original Publication

American Federation of Information Processing Societies (AFIPS) Conference Proceedings 23: 329–246. Spring Joint Computer Conference, 1963.

Sketchpad

A Man-Machine Graphical Communication System

Ivan E. Sutherland

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 man 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.

An Introductory Example

To understand what is possible with the system at present let us consider using it to draw the hexagonal pattern in Figure 9.4. 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 computer¹ is shown in Figure 9.1. When our drawing is complete it may be inked on paper, as were all the drawings in this paper, by a PACE plotter.¹⁵

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 9.2. Additional presses of the button will produce

additional lines, leaving the closed irregular hexagon shown in Figure 9.3A.

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 angular length only is controlled by light pen position as shown in Figure 9.2.

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 light pen, stretching two rubber band line segments behind it. By pointing to the circle and terminating, 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 as shown in Figure 9.3D.

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

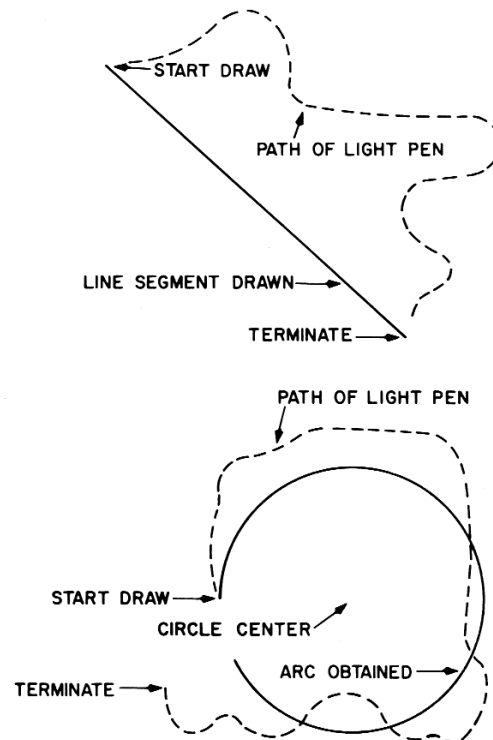


Figure 9.2. Steps for drawing straight lines and circle arcs.

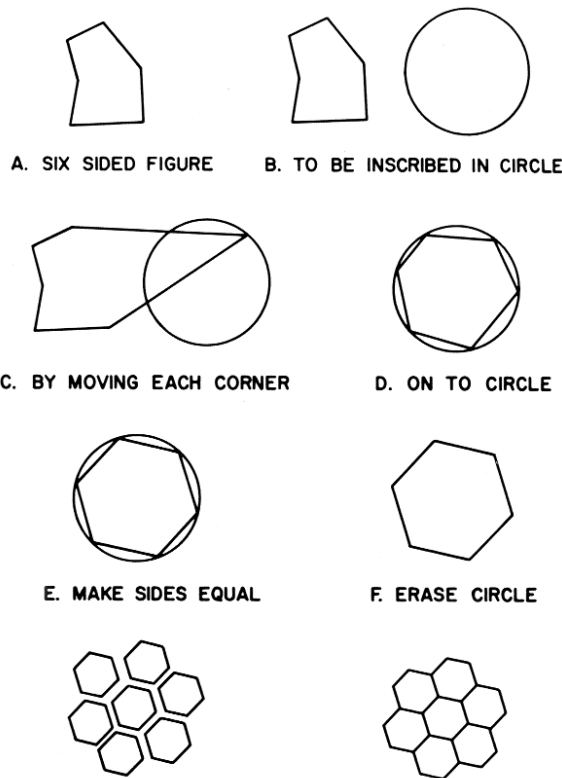


Figure 9.3. Illustrative example, see text.

the sides of the hexagon be of equal length, a regular hexagon will be constructed.

With Sketchpad we can say, in effect, make *this* line equal in length to *that* line, pointing to the lines with the light pen. 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 9.3F.

To make the hexagonal pattern in Figure 9.4 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 an “instance” or subpicture, six hexagons around

a central seventh in approximate position as shown in Figure 9.3G. A subpicture may be positioned with the light pen, rotated or scaled by turning the knobs, or fixed in position by a termination signal, but its 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 9.3H. An entire group of hexagons, once assembled, can be treated as a symbol. An “instance” of 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.

Interpretation of Introductory Example

In the introductory example above we used the light pen both to position parts of the drawing and to point to existing parts. 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, or an airplane wing. Any number of different symbols may be drawn, in terms of other simpler symbols if desired, and any symbol may be used as often as desired.

Constraint:

When we asked that the vertices of the hexagon lie on the circle we were making use of a basic relationship between picture parts that is built into the system. Basic relationships (atomic constraints) to make lines vertical, horizontal, parallel, or perpendicular; to make points lie on lines or circles; to make symbols appear upright, vertically above one another or be of equal size; and to relate symbols to other drawing parts such as points and lines have been included in the system. Specialized constraint types may be added as needed.

Definition Copying:

We made the sides of the hexagon be equal in length by pressing a button while pointing to the side in question. 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.

Implications of Introductory Example

As we have seen, a Sketchpad drawing is entirely different from the trail of carbon left on a piece of paper. Information about how the drawing is tied together is stored in the computer as well as the information which gives the drawing its particular appearance. Since the drawing is tied together, it will keep a useful appearance even when parts of it are moved. For example, when we moved the corners of the hexagon onto the circle, the lines next to each corner were automatically moved so that the closed topology of the hexagon was preserved. Again, since we indicated that the corners of the hexagon were to lie on the circle, they remained on the circle throughout our further manipulations.

As well as storing how the various parts of the drawing are related, Sketchpad stores the structure of the subpictures used. For example, the storage for the hexagonal pattern of Figure 9.4 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 but not the structure of the hexagonal pattern will be changed. For example, if we change the basic hexagon into a semicircle, the fish scale pattern shown in Figure 9.4 instantly results.

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 geometric constraints applied to the points and lines of the drawing model the design constraints which limit the values of design variables. The ability of Sketchpad to satisfy the geometric constraints applied to the parts of a drawing models the ability of a good designer to satisfy all the design conditions imposed by the limitations of his materials, cost, etc. In fact, since designers in many fields produce nothing themselves but a drawing of a part, design conditions may well be thought of as applying to the drawing of a part rather than to the part itself. When such design conditions are added to Sketchpad's vocabulary of constraints, the computer will be able to assist a user not only in arriving at a nice looking drawing, but also in arriving at a sound design.

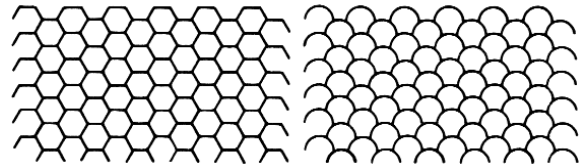


Figure 9.4. Hexagonal lattice with half hexagon and semicircle as basic elements.

Present Usefulness

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 Storing and Updating 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. A library of drawings will thus 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.

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

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. For instance, Sketchpad makes it easy to study mechanical linkages, observing the path of some parts when others are moved.

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

Since the storage structure 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.

II Ring Structure

The basic n -component element structure described by Ross¹⁰ 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, not only may the end points of a line segment be found by following pointers in the line block (n -component element), but also 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, hence the name "ring." The ring points both ways to

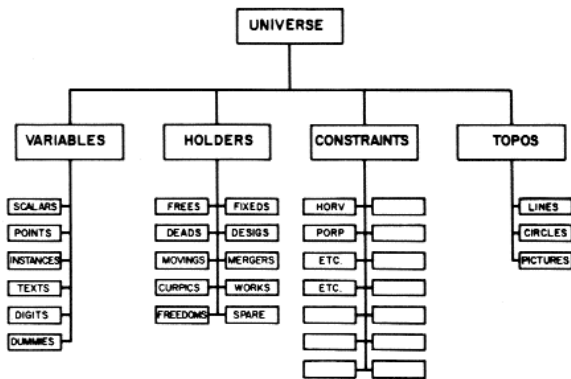


Figure 9.5. Generic structure. The n -component elements for each point or line, etc., are collected under the generic blocks "lines," "points," etc., shown.

make it easy to find both the next and the previous member of the ring in case, as when deleting, some change must be made to them.

Basic Operations

The basic ring structure operations are:

1. Inserting a new member into a ring at some specified location on it, usually first or last.
2. Removing a member from a ring.
3. Putting all the members 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, the basic programs operate without making special cases.

Subroutines are used for setting up new n -component elements in free spaces in the storage structure. As parts of the drawing are deleted, the registers which were used to represent them become free. 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.

Generic Structure, Hierarchies

The main part of Sketchpad can perform basic operations on any drawing part, calling for help from routines specific to particular types of parts when that is necessary. For example, the main program can show any part on the display system by calling the appropriate display subroutine. 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 without any need to change the general parts.

In the data storage structure the separation of general and specific is accomplished by collecting all things of one type together in a ring under a generic heading. The generic heading 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. The generic blocks are further gathered together under super-generic or generic-generic blocks, as shown in Figure 9.5.

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 the new type. 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. Without a generic structure it would be almost impossible to add the instructions required to handle a new type of element.

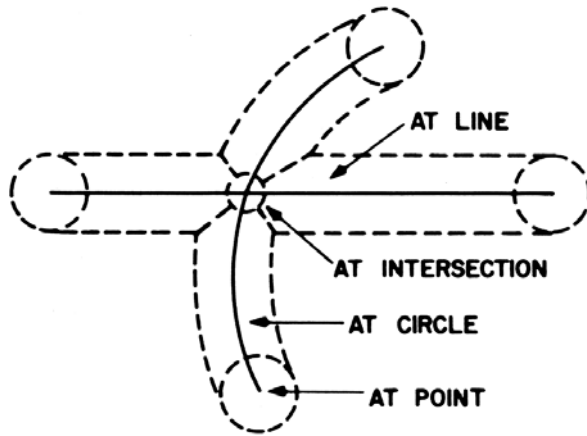


Figure 9.6. Areas in which pen must lie to “aim at” existing drawing parts (solid lines).

III 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

*The reader unacquainted with light pens should refer to the paper on Man-Machine Console Facilities by Stotz.¹²

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.

Pen Tracking

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. Sketchpad uses loss of tracking as a “termination signal” to stop drawing. The user signals that he is finished drawing by flicking the pen too fast for the tracking program to follow.

Demonstrative Use of Pen

During the 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. Thus, a table of the picture parts seen by the light pen is assembled during each 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.”

The one-half inch diameter field of view of the light pen, although well suited to tracking, is relatively large for pointing. Therefore, the Sketchpad system will reject any seen part which is further from the center of the light pen than some small minimum distance; about 1/8 inch was found to be suitable. 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.

After eliminating all parts seen by the pen 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 (subpictures) are especially important. One can thus aim at the end point of a line even though only the line is displayed. Figure 9.6 outlines the various regions within which the pen must lie to be considered aimed at a line segment, a circle arc, their end points, or their intersection.

Pseudo Pen Location

When the light pen is aimed at a picture part, the exact location of the light pen is ignored in favor of a “pseudo pen location” exactly on the part aimed at. If no object is aimed at, the pseudo pen location is taken to be the actual pen location. The pseudo pen location is displayed as a bright dot which is used as the “point of the pencil” in all drawing operations. As the light pen is moved into the areas outlined in Figure 9.6 the dot will lock onto the existing parts of the drawing, and any moving picture parts will jump to their new locations as the pseudo pen location moves to lie on the appropriate picture part.

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.

IV 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. The coordinates of the spots which are to be seen on the display are stored in a large table so that computation and display may proceed independently. If, instead of displaying each spot successively, the display program displays them in a random order or with interlace, the flicker of the display is reduced greatly.

Marking of Display File

Of the 36 bits available to store each display spot in the display file, 20 give the coordinates of that spot for the display system, and the remaining 16 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 (subpicture) 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.

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 nonmoving parts need not be disturbed. Moving parts are made invisible to the light pen.

Magnification of Pictures

The shaft position encoder knobs below the scope (see Figure 9.1) are used to tell the program to change the display scale factor or the portion of the page displayed. 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.

For a magnified picture, Sketchpad computes which portion(s) of a curve will appear on the display and generates display spots for those portions only. The "edge detection" problem is the problem of finding suitable end points for the portion of a curve which appears on the display.

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 logical complexity the edge detection problem is a difficult one. 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?

Line and Circle Generation

All of Sketchpad's displays are generated from straight line segments, circle arcs, and single points. The generation of the lines and circles is accomplished by means of the difference equations:

$$x_i = x_{i-1} + \Delta x \quad (1)$$

$$y_i = y_{i-1} + \Delta y$$

for lines, and

$$x_i = x_{i-2} + \frac{2}{R}(y_{i-1} - y_c) \quad (2)$$

$$y_i = y_{i-2} - \frac{2}{R}(x_{i-1} - x_c)$$

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. Even so, about 3/4 of the total Sketchpad computation time is spent in line and circle generation. A vector and circle generating display would materially reduce the computational load of Sketchpad.

For computers which do only one addition at a time, the difference equations:

$$x_i = x_{i-1} + \frac{1}{R}(y_{i-1} - y_c) \quad (3)$$

$$y_i = y_{i-1} - \frac{1}{R}(x_i - x_c)$$

should be used to generate circles. Equations (3) approximate a circle well enough and are known to close exactly both in theory and when implemented, because the x and y equations are dissimilar.

Digits and Text

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 as a single line of not more than 36 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

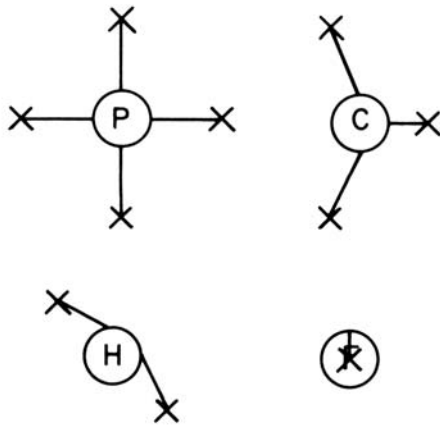


Figure 9.7. Display of constraints.

sign; binary to decimal conversion is provided, and leading zeros are suppressed.

Subpictures, whose use was seen in the introductory example above, are each represented in storage as a single n -component element. A subpicture is said to be an “instance” of its “master picture.” To display an instance, all of the lines, text, etc. of its master picture must be shown in miniature on the display. The instance display program makes use of the line, circle, number, and text display programs and *itself* to expand the internal structure of the instance.

Display of Abstractions

The usual picture for human consumption displays only lines, circles, text, digits, and instances. However, certain very useful abstractions which give the drawing the properties desired by the user are represented in the ring structure storage. 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 makes it possible for him to aim at them with the light pen and, for example, erase them. 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 9.7. The central circle and code letter 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 9.7 the constraints are shown applied to “dummy variables,” each of which shows as an X.

Another abstraction that can be displayed if desired is the value of a set of digits. For example, in Figure 9.8 are shown three sets of digits all displaying the same scalar value, -5978 . The digits themselves may be moved, rotated, or changed in size, without changing the value displayed. If we wish to change the value, we point at its abstract display, the # seen in Figure 9.8. The three sets of digits in Figure 9.8 all display the same value, as indicated by the lines connecting them to the #; changing this value would make all three sets of digits change. Constraints may be applied independently to either the position of the digits or their value as indicated by the two constraints in the figure.

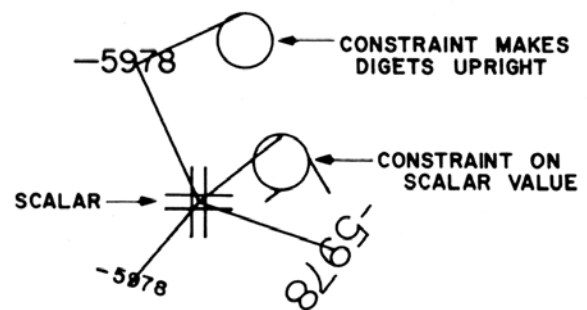


Figure 9.8. Three sets of digits displaying the same scalar value.

V 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 power obtained from the small set of generalized functions in Sketchpad is one of the most important results of the research.

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

1. Expansion of instances, making it possible to have subpictures within subpictures to as many levels as desired.
2. Recursive deletion, whereby removal of certain picture parts will remove other picture parts in order to maintain consistency in the ring structure.
3. Recursive merging, whereby combination of two similar picture parts forces combination of similarly related other picture parts, making possible application of complex definitions to an object picture.

Recursive Deleting

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, since the n-component elements for lines contain no positional information, 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.

Recursive Merging

If two things of the same type which are independent are merged, a single thing of that type results, and all things which depended on either of the merged things depend on the result of the merger.* For example, if two points are merged, all lines which previously terminated on either point now terminate on the single resulting point. In Sketchpad, if a thing is being moved with the light pen and the termination flick of the pen is

given while aiming at another thing of the same type, the two things will merge. Thus, if one moves a point to another point and terminates, the points will merge, connecting all lines which formerly terminated on either. This makes it possible to draw closed polygons.

If two things of the same type which do depend on other things are merged, the things depended on by one will be forced to merge, respectively, with the things depended on by the other. The result of merging two dependent things depends, respectively, on the results* of the mergers it forces.* For example, if two lines are merged, the resultant line must refer to only two end points, the results of merging the pairs of end points of the original lines. All lines which terminated on any of the four original end points now terminate on the appropriate one of the remaining pair. More important and useful, all constraints which applied to any of the four original end points now apply to the appropriate one of the remaining pair. This makes it possible to speak of line segments as being parallel even though (because line segments contain no numerical information to be constrained)

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

the parallelism constraint must apply to their end points and not to the line segments themselves. If we wish to make two lines both parallel and equal in length, the steps outlined in Figure 9.9 make it possible. More obscure relationships between dependent things may be easily defined and applied. For example, constraint complexes can be defined to make line segments be collinear, to make a line be tangent to a circle, or to make the values represented by two sets of digits be equal.

Recursive Display of Instances

The block of registers which represents an instance is remarkably small considering that it may generate a display of any complexity. For the purposes of display, the instance block makes reference to its master picture. The instance will appear on the display as a figure geometrically similar to its master picture at a location, size, and rotation indicated by the four numbers which constitute the "value" of the instance. The value of an instance is considered numerically as a four dimensional vector. The components of this vector are the coordinates of the center of the instance and its actual size as it appears on the drawing times the sine and cosine of the rotation angle involved.

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

Attachers and Instances

Many symbols must be integrated into the rest of the drawing by attaching lines to the symbols at appropriate

points, or by attaching the symbols directly to each other. For example, circuit symbols must be wired up, geometric patterns made by fitting shapes together, or mechanisms composed of links tied together appropriately. An instance may have any number of attachment points, and a point may serve as attacher for any number of instances. The light pen has the same affinity for the attachers of an instance that it has for the end point of a line.

An "instance-point" constraint, shown with code T in Figure 9.10C, is used to relate an instance to each of its attachment points. An instance-point constraint is satisfied only when the point bears the same relationship to the instance that a master point in the master picture for that instance bears to the master picture coordinate system.

Any point may be an attacher of an instance, but the point must be designated as an attacher in the master drawing of the instance. For example, when one first draws a resistor, the ends of the resistor must be designated as attachers if wiring is to be attached to instances of it. At each level of building complex pictures, the attachers must be designated anew. Thus of the three attachers of a transistor it is possible to select one or two to be the attachers of a flip-flop.

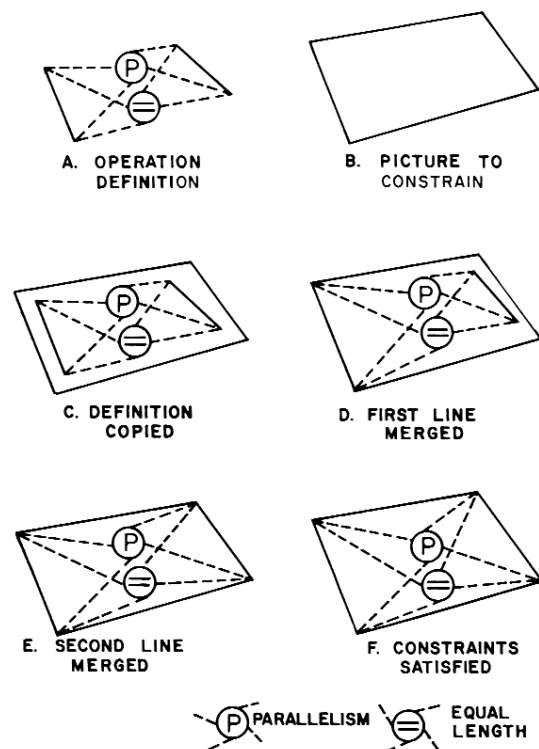


Figure 9.9. Applying a two-constraint definition to turn a quadrilateral into a parallelogram.

VI Building a Drawing, the Copy Function

At the start of the Sketchpad effort certain ad hoc drawing functions were programmed as the atomic operations of the system. Each such operation, controlled by a push button, creates in the ring structure a specific set of new drawing parts. For example, the "draw" button creates a line segment and two new end points (unless the light pen happens to be aimed at a point in which case only one new point need be created). Similarly, there are atomic operations for drawing circles, applying a horizontal or vertical constraint to the end points of a line aimed at, and for adding a "point-on-line" constraint whenever a point is moved onto a line and left there.

The atomic operations described above make it possible to create in the ring structure new picture components and relate them topologically. The atomic operations are, of course, limited to creating points, lines, circles, and two or three types of constraints. Since implementation of the copy function it has become possible to create in the ring structure any predefined combination of picture parts and constraints at the press of a button. The recursive merging function makes it possible to relate the copied set of picture parts to

any existing parts. For example, if a line segment and its two end points are copied into the object picture, the action of the “draw” button may be exactly duplicated in every respect. Along with the copied line, however, one might copy as well a constraint, Code H, to make the line horizontal as shown in Figure 9.10A, or two constraints to make the line both horizontal and three inches long, or any other variation one cares to put into the ring structure to be copied.

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

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

Normally only two designated attachers are used because it is hard to keep track of additional ones.

If the definition picture consists of two line segments, their four end points, and a constraint, Code M, on the points which makes the lines equal in length, with the two lines designated as attachers as shown in Figure 9.10B, copying enables the user to make any two lines equal in length. If the pen is aimed at a line when “copy” is pushed, the first of the two copied lines merges with it (taking its position and never actually being seen). The other copied line is left moving with the light pen and will merge with whatever other line the pen is aimed at when termination occurs. Since merging is recursive, the copied equal-length constraint, Code M, will apply to the end points of the desired pair of object picture lines.

Copying Instances

As we have seen above, the internal structure of an instance is entirely fixed. The internal structure of a copy, however, is entirely variable. An instance always retains its identity as a

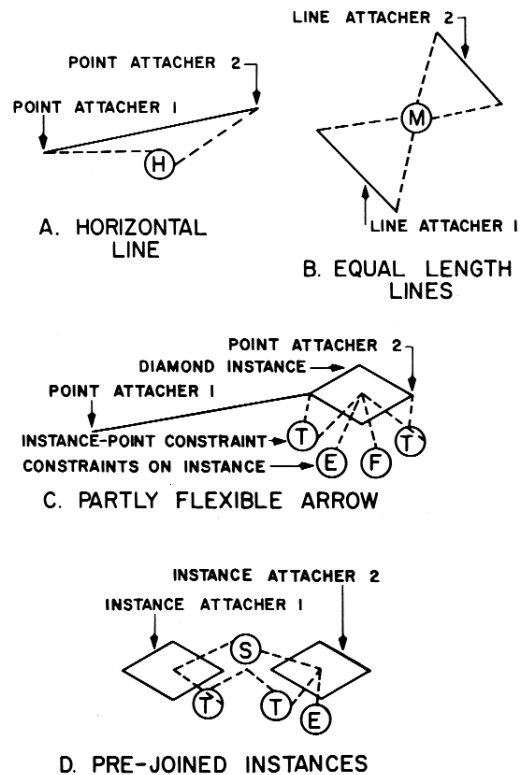


Figure 9.10. Definition pictures to be copied, see text.

single part of the drawing; one can only delete an entire instance. Once a definition picture is copied, however, the copy loses all identity as a unit; individual parts of it may be deleted at will.

One might expect that there was intermediate ground between the fixed-internal-structure instance and the loose-internal-structure copy. One might wish to produce a collection of picture parts, some of which were fixed internally and some of which were not. *The entire range of variation between the instance and the copy can be constructed by copying instances.*

For example, the arrow shown in Figure 9.10C can be copied into an object picture to result in a fixed-internal-structure diamond arrowhead with a flexible tail. As the definition in Figure 9.10C is set up, drawing diamond-arrowheaded lines is just like drawing ordinary lines. One aims the light pen where the tail is to end, presses “copy,” and moves off with an arrowhead following the pen. The

diamond arrowhead in this case will not rotate (constraint Code E), and will not change size (constraint Code F).

Copying pre-joined instances can produce vast numbers of joined instances very easily. For example, the definition in Figure 9.10D, when repetitively copied, will result in a row of joined, equal size (constraint Code S) diamonds. In this case the instances themselves are attachers. Although each press of the “copy” button copies two new instances into the object picture, one of these is merged with the last instance in the growing row. In the final row, therefore, each instance carries all constraints which are applied to either of the instances in the definition. This is why only one of the instances in Figure 9.10D carries the erect constraint, Code E.

VII Constraint Satisfaction

The major feature which distinguishes a Sketchpad drawing from a paper and pencil drawing is the user’s ability to specify to Sketchpad mathematical conditions on already drawn parts of his drawing which will be automatically satisfied by the computer to make the drawing take the exact shape desired. The process of fixing up a drawing to meet new conditions applied to it after it is already partially complete is very much like the process a designer goes through in turning a basic idea into a finished design. As new requirements on the various parts of the design are thought of, small changes are made to the size or other properties of parts to meet the new conditions. By making Sketchpad able to find new values for variables which satisfy the conditions imposed, it is hoped that designers can be relieved of the need of much mathematical detail. The effort expended in making the definition of constraint types as general as possible was aimed at making design constraints as well as geometric constraints equally easy to add to the system.

Definition of a Constraint Type

Each constraint type is entered into the system as a generic block indicating the various properties of that particular constraint type. The generic block tells how many variables are constrained, which of these variables may be changed in order to satisfy the constraint, how many degrees of freedom are removed from the constrained variables, and a code letter for human reference to this constraint type.

The definition of what a constraint type does is a subroutine which will compute, for the existing values of the variables of a particular constraint of that type, the error

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

When the one pass method of satisfying constraints to be described later on fails, the Sketchpad system falls back on the reliable but slow method of relaxation¹¹ to reduce the errors indicated by various computation subroutines to smaller and smaller values. For simple constructions such as the hexagon illustrated in Figure 9.3, the relaxation procedure is sufficiently fast to be useful. However, for complex systems of variables, especially directly connected instances, relaxation is unacceptably slow. Fortunately it is for just such directly connected instances that the one pass method shows the most striking success.

One Pass Method

Sketchpad can often find an order in which the variables of a drawing may be re-evaluated to completely satisfy all the conditions on them in just one pass. For the cases in which the one pass method works, it is far better than relaxation: it gives correct answers at once; relaxation may not give a correct solution in any finite time. Sketchpad can find an order in which to re-evaluate the variables of a drawing for most of the common geometric constructions. Ordering is also found easily for the mechanical linkages shown in Figures 9.13 and 9.14. Ordering cannot be found for the bridge truss problem in Figure 9.15.

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

Suppose that some variable can be found which has so few constraints applying to it that it can be re-evaluated to completely satisfy all of them. Such a variable we shall call a "free" variable. As soon as a variable is recognized as free, the constraints which apply to it are removed from further consideration, because the free variable can be used to satisfy them. Removing these constraints, however, may make adjacent variables free. Recognition of these new variables as free removes further constraints from consideration and may make other adjacent variables free, and so on throughout the maze of constraints. The manner in which freedom spreads is much like the method used in Moore's algorithm⁸ to find the shortest path through a maze. Having found that a collection of variables is free, Sketchpad will re-evaluate them in reverse order, saving the first-found free variable until last. In re-evaluating any particular variable, Sketchpad uses only those constraints which were present when that variable was found to be free.

VIII Examples and Conclusions

The examples in this section were all taken from the library tape and thus serve to illustrate not only how the Sketchpad system can be used, but also how it actually has been used so

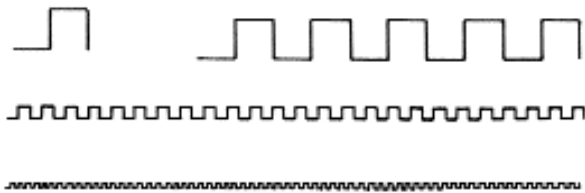


Figure 9.11. Zig-Zag for delay line.

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

Patterns

The instance facility enables one to draw any symbol and duplicate its appearance anywhere on an object drawing at the push of a button. This facility made the hexagonal

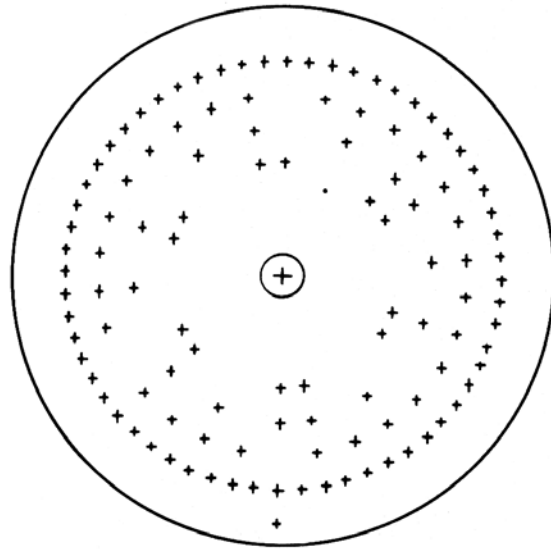


Figure 9.12. Binary coded decimal encoder for clock. Encoder was plotted exactly 12 inches in diameter for direct use as a layout.

pattern we saw in Figure 9.4 easy to draw. It took about one half hour to generate 900 hexagons, including the time taken to figure out how to do it. Plotting them takes about 25 minutes. The drafting department estimated it would take two days to produce a similar pattern.

The instance facility also made it easy to produce long lengths of the zig-zag pattern shown in Figure 9.11. As the figure shows, a single "zig" was duplicated in multiples of five and three, etc. Five hundred zigs were generated in a single

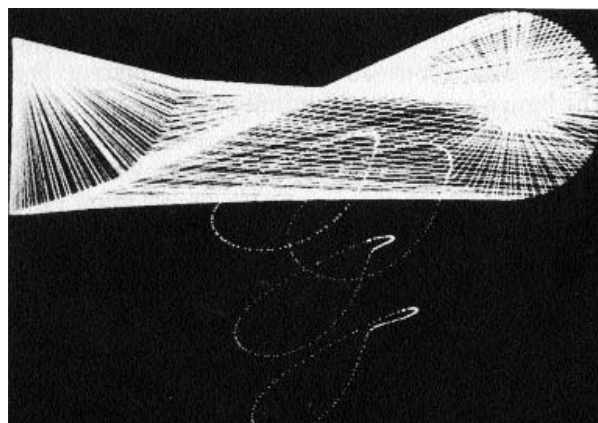


Figure 9.13. Three bar linkage. The paths of four points on the central link are traced. This is a 15 second time exposure of a moving Sketchpad drawing.

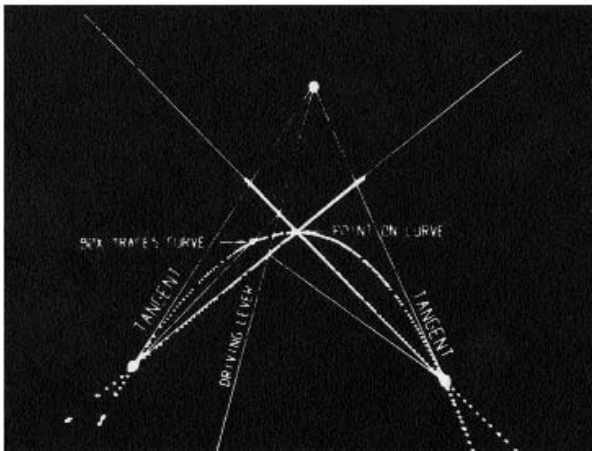
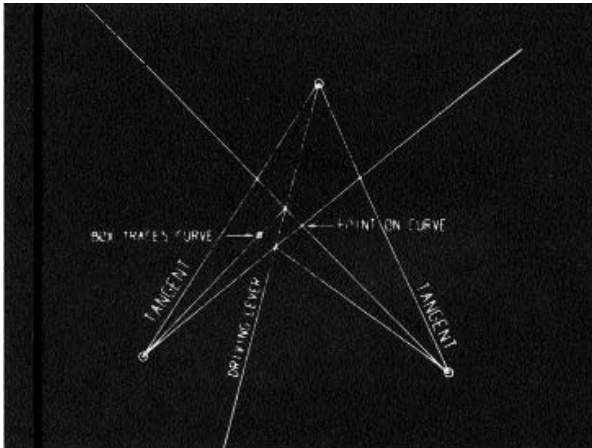


Figure 9.14. Conic drawing linkage. As the “driving lever” is moved, the point shown with a box around it (in A) traces a conic section. This conic can be seen in the time exposure (B).

row. Four such rows were plotted one-half inch apart to be used for producing a printed circuit delay line. Total time taken was about 45 minutes for constructing the figure and about 15 minutes to plot it.

A somewhat less repetitive pattern to be used for encoding the time in a digital clock is shown in Figure 9.12. Each cross in the figure marks the position of a hole. The holes are placed so that a binary coded decimal (BCD) number will indicate the time. Total time for placing crosses was 20 minutes, most of which was spent trying to interpret a pencil sketch of their positions.

Linkages

By far the most interesting application of Sketchpad so far has been drawing and moving linkages. The ability to draw and then move linkages opens up a new field of graphical manipulation that has never before been available. It is remarkable how even a simple linkage can generate complex motions. For example, the linkage of Figure 9.13 has only three moving parts. In this linkage a central \perp link is suspended between two links of different lengths. As the shorter link rotates, the longer one oscillates as can be seen in the multiple exposure. The \perp link is not shown in Figure 9.13 so that the motion of four points on the upright part of the \perp may be seen. These are the four curves at the top of the figure.

To make the three bar linkage, an instance shaped like the \perp was drawn and given 6 attachers, two at its joints with the other links and four at the places whose paths were to be observed. Connecting the \perp shaped subpicture onto a linkage composed of three lines with fixed length created the picture shown. The driving link was rotated by turning a knob below the scope. Total time to construct the linkage was less than 5 minutes, but over an hour was spent playing with it.

A linkage that would be difficult to build physically is shown in Figure 9.14A. This linkage is based on the complete quadrilateral. The three circled points and the two lines which extend out of the top of the picture to the right and left are fixed. Two moving lines are drawn from the lower circled points to the intersections of the long fixed lines with the driving lever. The intersection of these two moving lines (one must be extended) has a box around it. It can be shown theoretically that this linkage produces a conic section which passes through the place labeled “point on curve” and is tangent to the two lines marked “tangent.” Figure 9.14B shows a time exposure of the moving point in many positions. At first, this linkage was drawn and working in 15 minutes. Since then we have rebuilt it time and again until now we can produce it from scratch in about 3 minutes.

Dimension Lines

To make it possible to have an absolute scale in drawings, a constraint is provided which forces the value displayed by a set of digits to indicate the distance between two points on the drawing. This distance-indicating constraint is used to make the number in a dimension line correspond to its length. Putting in a dimension line is as easy as drawing any

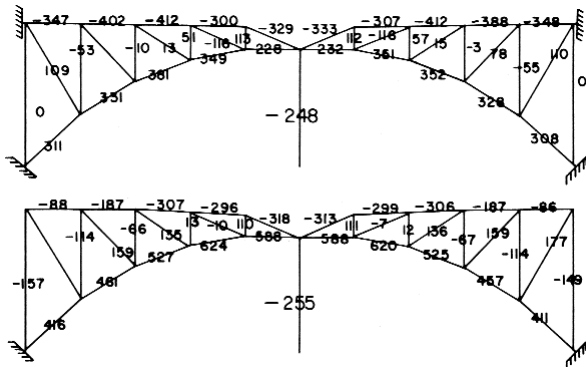


Figure 9.15. Cantilever and arch bridges. The numbers indicate the forces in the various members as computed by Sketchpad. Central load is not exactly vertical.

other line. One points to where one end is to be left, copies the definition of the dimension line by pressing the “copy” button, and then moves the light pen to where the other end of the dimension line is to be. The first dimension line took about 15 minutes to construct, but that need never be repeated since it is a part of the library.

Bridges

One of the largest untapped fields for application of Sketchpad is as an input program for other computation programs. The ability to place lines and circles graphically, when coupled with the ability to get accurately computed results pictorially displayed, should bring about a revolution in computer application. By using Sketchpad’s relaxation procedure we were to demonstrate analysis of the force distribution in the members of a pin connected truss.

A bridge is first drawn with enough constraints to make it geometrically accurate. These constraints are then deleted and each member is made to behave like a bridge beam. A bridge beam is constrained to maintain constant length, but any change in length is indicated by an associated number. Under the assumption that each bridge beam has a cross-sectional area proportional to its length, the numbers represent the forces in the beams. The basic bridge beam definition (consisting of two constraints and a number) may be copied and applied to any desired line in a bridge picture by pointing to the line and pressing the “copy” button.

Having drawn a basic bridge shape, one can experiment with various loading conditions and supports to see what the effect of making minor modifications is. For example, an arch bridge is shown in Figure 9.15 supported both as a three-hinged arch (two supports) and as a cantilever (four

supports). For nearly identical loading conditions the distribution of forces is markedly different in these two cases.

Artistic Drawings

Sketchpad need not be applied exclusively to engineering drawings. For example, the girl “Nefertite” shown in Figure 9.16 can be made to wink by changing which of the three types of eyes is placed in position on her otherwise eyeless face. In the same way that linkages can be made to move, a stick figure could be made to pedal a bicycle or Nefertite’s hair could be made to swing. The ability to make moving drawings suggests that Sketchpad might be used for making animated cartoons.

Electrical Circuit Diagrams

Unfortunately, electrical circuits require a great many symbols which have not yet been drawn properly with Sketchpad and therefore are not in the library. After some time is spent working on the basic electrical symbols it may be easier to draw circuits. So far, however, circuit drawing has proven difficult.

The circuits of Figure 9.17 are parts of an analog switching scheme. You can see in the figure that the more complicated circuits are made up of simpler symbols and circuits. It is

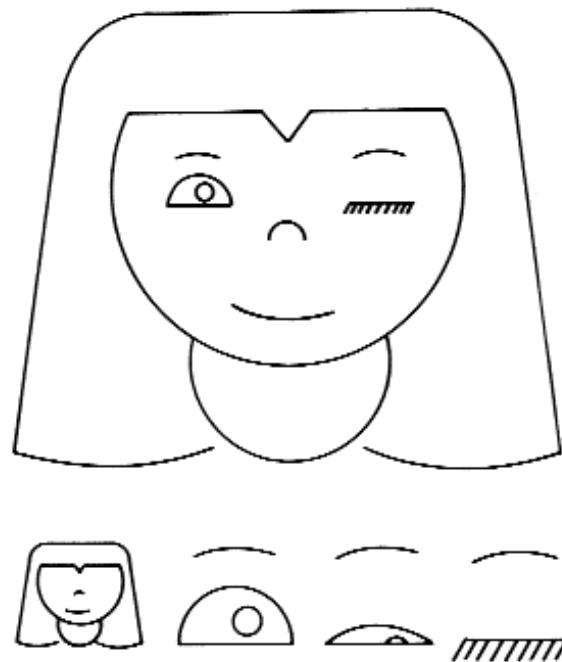


Figure 9.16. Winking girl, “Nefertite,” and her component parts.

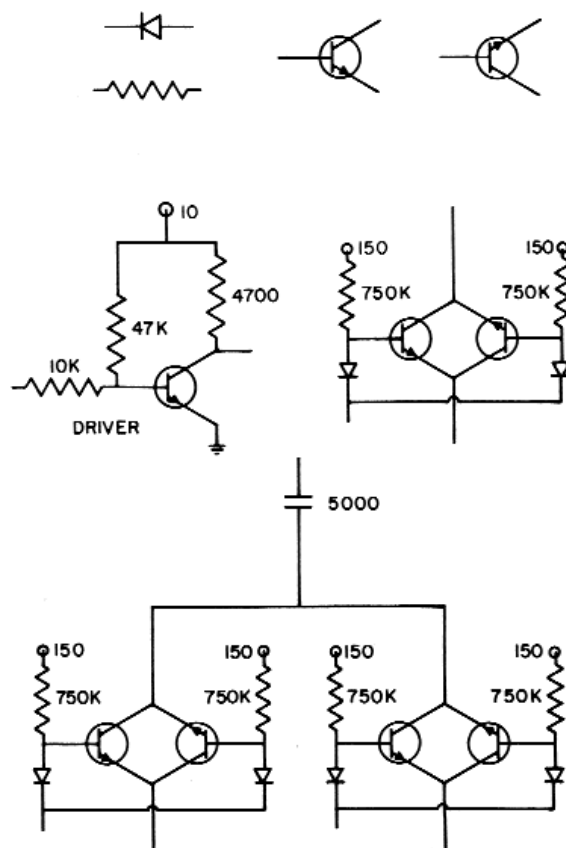


Figure 9.17. Circuit diagrams. These are parts of the large circuit mentioned in the text.

very difficult, however, to plan far enough ahead to know what composites of circuit symbols will be useful as subpictures of the final circuit. The simple circuits shown in Figure 9.17 were compounded into a big circuit involving about 40 transistors. Including much trial and error, the time taken by a new user (for the big circuit not shown) was ten hours. At the end of that time the circuit was still not complete in every detail and he decided it would be better to draw it by hand after all.

Conclusions

The circuit experience points out the most important fact about Sketchpad drawings. It is only worthwhile to make drawings on the computer if you get something more out of the drawing than just a drawing. In the repetitive patterns we saw in the first examples, precision and ease of constructing great numbers of parts were valuable. In the

linkage examples, we were able to gain an understanding of the behavior of a linkage as well as its appearance. In the bridge examples we got design answers which were worth far more than the computer time put into them. If we had had a circuit simulation program connected to Sketchpad so that we would have known whether the circuit we drew worked, it would have been worth our while to use the computer to draw it. We are as yet a long way from being able to produce routine drawings economically with the computer.

Future Work

The methods outlined in this paper generalize nicely to three dimensional drawing. In fact, the work reported in "Sketchpad III" by Timothy Johnson³ will let the user communicate solid objects to the computer. Johnson is completely bypassing the problem of converting several two dimensional drawings into a three dimensional shape. Drawing will be directly in three dimensions from the start. No two dimensional representation will ever be stored.

Work is also proceeding in direct conversion of photographs into line drawings. Roberts reports a computer program⁹ able to recognize simple objects in photographs well enough to produce three dimensional line drawings for them. Roberts is storing his drawings in the ring structure described here so that his results will be compatible with the three dimensional version of Sketchpad.

Major improvements to Sketchpad of the same order and power as the existing definition copying capability can be foreseen. At present Sketchpad is able to add defined relationships to an existing object drawing. A method should be devised for defining and applying changes which involve removing some parts of the object drawing as well as adding new ones. Such a capability would permit one to define, for example, what rounding off a corner means. Then, one could round off any corner by pointing to it and applying the definition.

Notes and Acknowledgements

This paper is based in part on a thesis submitted to the Department of Electrical Engineering, M.I.T., in partial fulfillment of the requirements for the Degree of Doctor of Philosophy.

Lincoln Laboratory is operated with the support of the U.S. Army, Navy, and Air Force.

The author is indebted to Professors Claude E. Shannon, Marvin Minsky and Steven A. Coons of the Massachusetts Institute of Technology for their help and advice throughout the course of this research.

The author also wishes to thank Douglas T. Ross and Lawrence G. Roberts for their help and answers to his many questions.

Bibliography

1. Clark, W. A., Frankovich, J. M., Peterson, H. P., Forgie, J. W., Best, R. L., Olsen, K. H., "The Lincoln TX-2 Computer," Technical Report 6M-4968, Massachusetts Institute of Technology, Lincoln Laboratory, Lexington, Mass., April 1, 1957, *Proceedings of the Western Joint Computer Conference*, Los Angeles, California, February 1957.
2. Coons, S. A., *Notes on Graphical Input Methods*, Memorandum 8436-M-17, Dynamic Analysis and Control Laboratory, Massachusetts Institute of Technology, Department of Mechanical Engineering, Cambridge, Mass., May 4, 1960.
3. Johnson, T. E., "Sketchpad III, Three Dimensional Graphical Communication with a Digital Computer," *Proceedings of the Spring Joint Computer Conference*, Detroit, Michigan, May 21-23, 1963.
4. Johnston, L. E., *A Graphical Input Device and Shape Description Interpretation Routines*, Memorandum to Prof. Mann, Massachusetts Institute of Technology, Department of Mechanical Engineering, Cambridge, Mass., May 4, 1960.
5. Licklider, J. C. R., "Man-Computer Symbiosis," *I.R.E. Trans. on Human Factors in Electronics*, vol. HFE, pp. 4-10, March 1960.
6. Licklider, J. C. R., and Clark, W., "On-line Man-Computer Communication," *Proceedings of the Spring Joint Computer Conference*, San Francisco, California, May 1-3, 1962, vol. 21, pp. 113-128.
7. Loomis, H. H. Jr., *Graphical Manipulation Techniques Using the Lincoln TX-2 Computer*, Group Report 51G-0017, Massachusetts Institute of Technology, Lincoln Laboratory, Lexington, Mass., November 10, 1960.
8. Moore, E. F., "On the Shortest Path Through a Maze," *Proceedings of the International Symposium on the Theory of Switching*, Harvard University, Harvard Annals, vol. 3, pp. 285-292, 1959.
9. Roberts, L. G., *Machine Perception of Three Dimensional Solids*, Ph.D. Thesis, Massachusetts Institute of Technology, Electrical Engineering Department, Cambridge, Mass., February 1963.
10. Ross, D. T., Rodriguez, J. E., "Theoretical Foundations for the Computer-Aided Design System," *Proceedings of the Spring Joint Computer Conference*, Detroit, Michigan, May 21-23, 1963.
11. Southwell, R. V., *Relaxation Methods in Engineering Science*, Oxford University Press, 1940.
12. Stotz, R., "Man-Machine Console Facilities for Computer-Aided Design," *Proceedings of the Spring Joint Computer Conference*, Detroit, Michigan, May 21-23, 1963, (this issue).
13. Vanderburgh, A. Jr., *TX-2 Users Handbook*, Lincoln Manual No. 45, Massachusetts Institute of Technology, Lincoln Laboratory, Lexington, Mass., July 1961.
14. Walsh, J. F., and Smith, A. F., "Computer Utilization," *Interim Engineering Report 6873-IR-10 and 11*, Electronic Systems Laboratory, Massachusetts Institute of Technology, Cambridge, Mass., pp. 57-70, November 30, 1959.
15. Handbook for Variplotter Models 205S and 205T, PACE, Electronic Associates Incorporated. Long Branch, New Jersey, June 15, 1959.