ESD-TR-72-126 AD 742252

NETWORKING AND GRAPHICS RESEARCH: FINAL REPORT

December 1971

DEPUTY FOR COMMAND AND MANAGEMENT SYSTEMS
HQ ELECTRONIC SYSTEMS DIVISION (AFSC)
L. G. Hanscom Field, Bedford, Massachusetts 01730

Sponsored by: Advanced Research Projects Agency
1400 Wilson Boulevard
Arlington, Virginia 22209

ARPA Order No. 952

(Prepared under Contract No. F19628-71-C-0174 by Harvard University, Cambridge, Massachusetts 02138.)

Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY *(Corporate author)* Harvard University Cambridge, Massachusetts 02138	2a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED 2b. GROUP N/A

3. REPORT TITLE

NETWORKING AND GRAPHICS RESEARCH: FINAL REPORT

4. DESCRIPTIVE NOTES *(Type of report and inclusive dates)*

None

5. AUTHOR(S) *(First name, middle initial, last name)*

None

6. REPORT DATE December 1971	7a. TOTAL NO. OF PAGES 72	7b. NO. OF REFS 11

8a. CONTRACT OR GRANT NO. F19628-71-C-0174 b. PROJECT NO. ARPA Order No. 952 c. Program Code No. 61101D d.	9a. ORIGINATOR'S REPORT NUMBER(S) ESD-TR-72-126 9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*

10. DISTRIBUTION STATEMENT

Approved for public release; distribution unlimited.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY Deputy for Command and Management Systems Hq Electronic Systems Division (AFSC) L G Hanscom Field, Bedford, Mass. 01730

13. ABSTRACT

This report describes the results of the program of research carried out at Harvard University utilizing the PDP-1-based graphics facility, the PDP-10, and the ARPA Network. Projects undertaken range from development of techniques for graphic input and display and development of graphics applications, to development of a programming system intended as a tool for attacking "difficult" programming projects and suitable for use concurrently at several nodes of the ARPA Network on a common computational task, and to development of programs specific to use of the ARPA Network and file transfer techniques. This report references various published papers and technical reports which contain more detailed accounts of the work undertaken.

Security Classification

14 KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Computer Graphics						
Computer Networking						
Computers						
Extensible Languages						
Flight Simulation						
Graphics Applications						
Logic Design						
Organic Syntheses						
Networking						
Urban Traffic Simulation						
Visual Perception						

UNCLASSIFIED
Security Classification

FOREWORD

This report presents the results of research conducted by Harvard University, Cambridge, Massachusetts, during the period 1 July 1968 through 31 August 1971 in support of ARPA Order 952 under Contract F19628-68-C-0379. Dr. J. B. Goodenough (ESD/MCIT) was the ESD Project Monitor.

This technical report has been reviewed and is approved.

Robert B Doane

G. FERNANDEZ, Colonel, USAF
for Deputy for Command & Management Systems

ABSTRACT

This report describes the results of the program of research carried out at Harvard University utilizing the PDP-1-based graphics facility, the PDP-10, and the ARPA Network. Projects undertaken range from development of techniques for graphic input and display and development of graphics applications, to development of a programming system intended as a tool for attacking "difficult" programming projects and suitable for use concurrently at several nodes of the ARPA Network on a common computational task, and to development of programs specific to use of the ARPA Network and file transfer techniques. This report references various published papers and technical reports which contain more detailed accounts of the work undertaken.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

SECTION I

TECHNICAL SUMMARY

This report describes the results of the program of research carried out at Harvard University utilizing the PDP-1-based graphics facility, the PDP-10, and the ARPA Network during the contract period.

Work in the graphics area has included (1) development of mathematical and hardware techniques for graphical input and display, (2) development of an appropriate man-machine interface which provides the user with a sophisticated input/output facility appropriately tailored to his particular problem area, (3) development of and experimentation with specific application programs to permit evaluation of the man-machine interface, and (4) experimental use of the ARPA Network.

The ECL system contains as its language component an extensible programming language with facilities for new data type and operator definitions. The system has been designed as a tool for tackling "difficult" programming projects -- that is, projects on which existing languages can be used only with considerable waste in machine or programmer time. The system is intended to be usable for common computational tasks carried out concurrently on several nodes of the ARPA Network.

Work more specifically related to the ARPA Network has included the construction of Network Control Programs on both computers, and work directed toward facilitating the transfer of files over the Network.

SECTION II

INTRODUCTION

This report describes the results of the program of research carried out at Harvard University utilizing the PDP-1-based graphics facility, the PDP-10, and the ARPA Network during the contract period.

The PDP-1-based graphics facility includes input via tablets and a 3-D input device designed and constructed at Harvard and output via four relatively high speed point-plotting cathode ray tubes. The latter have been supplemented by a device permitting color and stereoscopic display. The PDP-1 has also been interfaced to the Interface Message Processor and to the PDP-10 via a direct (half-duplex) high speed link. The PDP-10 has also been interfaced to the ARPA Network. In addition, an ARDS terminal including two storage tubes, a keyboard, and a tablet is attached to the PDP-10.

Work in the area of computer graphics has been performed during the entire contract period. Section III of this report provides an overview of the entire area as well as more detailed discussions of a number of the individual projects undertaken. We wish to draw particular attention to the work described in Section 3.7 in the area of computer assisted design of complex organic syntheses. This work has been guided by Professor E. J. Corey of the Harvard Chemistry Department and represents, we believe, an outstanding example of the wedding of computer science to an applications area.

Section IV contains an overview of the ECL programming system, whose development has been partially supported by the subject contract. Implementation of ECL was begun in June, 1970 and the initial version became available for local use in September, 1971. There is one particular aspect of ECL which, we believe, is unique. That is, the system is intended to be used for common computational tasks carried out concurrently on several nodes of the ARPA

Network and, hence, the system would be particularly appropriate for such tasks as management of large scale distributed data bases, interactive graphics, and so on. Active work on this aspect of ECL awaits availability of ECL at another node in the network, which we expect to accomplish in the near future.

Work on computer networks started in the first half of 1970. This has included participation in the Network Working Group development of the HOST-HOST and file transfer protocols. Network Control Programs have been designed and implemented on both the PDP-1 and PDP-10 systems; the PDP-1 design included development of a time-sharing monitor to allow multiple user access to the system. Prior to completion of the NCP's, a number of programs had been written to allow interim use of the Network in a "local" mode -- that is, making the PDP-1 facilities available to the PDP-10. In addition, we have prepared for performing experimental work in the area of file transfer *via* the network. In order to transfer files between machines which use different representations for data items (e.g., 36 bit integers represented as sign and magnitude vs. 32 bit two's complement integers), data type conversion may be necessary. The work on ECL has lead to sharpening of the intuitive notion of data type; in the final subsection of this report, we describe what we believe to be implications of this work for file transfer and storage.

SECTION III

COMPUTER GRAPHICS

As has been known and observed for many years, the bottleneck of interactive systems is the man/machine interface. This interface is traditionally a sequence of characters, into or from the machine. Punched cards, paper tapes, magnetic tapes, teletypes, line printer and more -- all are sequential devices for characters. This kind of communication seems to follow the most common form of human communication, natural language. Sequential language is sufficient for most applications -- most but not all. There are many areas in which words alone are not enough, because they cannot convey some kinds of messages. It is very difficult, or impossible, to describe certain shapes using words only. Architecture without blueprints, electrical engineering without drawings, geodesics without maps, or general design without sketches are not acceptable. When engineers discuss problems they need drawings and blackboards. Nothing can describe a bridge as well as its drawing. Nothing can describe a house as well as its floor plan.

The process of reading words differs from the process of hearing words only in a small factor of speed. These human input channels are slow and cumbersome compared with visual perception. If something can be described by images rather than by words, for example, by graphs instead of tables of numbers, or by drawing instead of verbal description, then it can be received by a human input channel which is hundreds of times faster and better equipped for the task. This difference between reception of sequential characters, and perception of images is the main justification of computer graphics. Computer graphics is a way of using computers, and communicating to and from them, by using shapes in addition to words. These shapes can be static or dynamic, on the CRT or on paper, real world images or symbols. What is common to all is that they convey information that otherwise would require an endless amount of time and paper to be presented.

Computer graphics is also used for its fast capabilities for mass output. Using computer graphics technology, CRTs and the like, enables mass amounts of alpha-numeric information to be displayed in nearly no time.

The most interesting graphics applications are those which rely not only on the speed of alpha-numeric display but on the capability of displaying arbitrary shapes dynamically.

The applications of computer graphics can be divided into several classes. One class deals mainly with symbolic displays. These applications use a display to show the structure and function of elements without showing their real shape. For example, complex data structures can be displayed by means of networks, trees, and the like. Communication networks can be represented by graphs without any relation to the physical location of each node. Electronic design is represented by a collection of symbols which have nothing to do with the real shapes of the modules which are represented by them.

Another class of applications deals with real-life objects. Flight simulation, for example, requires the real shape of the world to be displayed. Mechanical and civil engineering, or architecture, require handling of real-world shapes, which cannot be represented by symbolic displays. Certain applications require only static display. Others require dynamic updating of the display. Dynamic displays are needed to animate motion and evolution. One can display the changes in data structure, like growth of data trees, by dynamic presentation of the data structure at any stage. One can animate growth of a city or growth of economy by dynamically altering the display. The time scale in which this animation occurs depends only on the convenience of the viewer. Another class requires motion in real time. Simulating moon landing or space craft docking requires dynamics in a real-time. The restriction of real-time display imposes a new dimension of complexity on these applications.

Computer graphics is uniquely suited to many applications. In some areas, such as engineering planning and design, it is natural to provide a man with a graphic display as a tool to aid him in the design process. Often this application suggests a further use of the computer -- to interpret the graphical display, to "understand" what has been drawn. Then the computer can investigate the properties of the design extensively and return its findings graphically. They might take the form of simple measurements, or, in the case of complex display, the computer can simulate the system represented. The process of graphical specification of complex systems and graphical simulation of their behavior is a tremendous aid to understanding and problem solving in many areas.

In the following we describe seven classes of computer graphics applications:

1) Simulation of systems
2) Dynamic display of real world shapes in real-time
3) Animation
4) Manipulation of symbols
5) Computer aided design
6) Experiments on visual perception
7) Design of complex organic syntheses

3.1 SIMULATION OF SYSTEMS

It is often the case that practical problems deal with system behavior, rather than behavior of a single particle or a single element. Describing and dealing with systems is many-fold more complex than working with a single element. Often one can describe very precisely the exact mathematics which governs the behavior of a single element. However, it is very seldom that one can find equations which describe a system completely, and still be consistent with the behavior of each particle of it. Consider, for example, fluid mechanics. One can describe the differential equations which govern the behavior, the speed, and the position of each fluid particle. One can even integrate these equations and describe the behavior of the entire svstem, the entire fluid, but only for some trivial cases. Once the system is not trivial (because of realistic boundary conditions, for example) one cannot integrate the particles' behavior into a description of the system. One can describe the growth of a single cell in an organ; however, integration of the individual behavior into organ growth equations is most difficult and often impossible. One can describe behavior of a single person in a crowd. However, integration of this description into crowd-psychology is not simple and may be misleading. The inherent problem in the integration is the interaction between single particles.

Simulation of urban traffic, or air traffic, are other examples of the same difficulty. It is the case that one can describe very precisely the motion of a single car or of a single airplane. If the car is alone, and its motion is unrestricted, then its behavior is simple to explain. When more than one element is introduced into the system, the interaction between them adds a new dimension to the problem. The complexity of the interactions might grow as the square of the number of objects in the interaction.

In general, one can solve situations where few vehicles are involved. However, any practical problem involves too many objects for a human being to solve without a computer.

These system-problems lend themselves very well to computer use. In order to solve these systems on a computer, one can use simulation techniques, rather than integrating equations into system-behavior. A computer can perform the tedious job of simulation particle by particle and make local judgments and local decisions according to the environment as seen by each of the particles. This process is applicable to some problems by describing each particle individually at a given time (t) and updating all the particles going into the next step in time ($t + \Delta t$). This solution might depend very highly on the order in which the particles are simulated, but introducing random sequence for updating the particles might overcome this problem.

It is often the case, unfortunately, that introducing any realistic boundary condition implies mathematical impossibilities. For example, many theoretical models of the Atlantic Ocean, proposed to explain its currents, assume either a rectangular ocean or a triangular one. There is no hope to solve mathematically the dynamics of the ocean with its real shape. Fortunately, for computers, introducing more restrictive boundary conditions may help the problem. The more restrictions exist in the system, the less checking and computation the computer has to perform. The computer can, in many cases, handle very easily any kind of non-unif.rm boundary condition just as if the boundary was uniform. The difficulty introduced by the non-uniformity is, in cases, negligible.

If the behavior of each individual particle is non-deterministic and some distribution and probabilities are involved in the description of each particle, then the behavior of the entire system is non-deterministic; in order to simulate it properly one has to simulate the distributions. These non-deterministic simulations have to be repeated many times in order to average the behavior and the distribution, to get meaningful results. Clearly, it is appropriate to use a computer for such simulations.

Computer graphics lends itself very well to this kind of simulation. After every updating cycle, one can display the state of the system. For example, if one simulates the growth of a tree based on some known growing rules, one would like to know through which steps the tree goes. The visual display of this information, at a rate meaningful for the viewer, might introduce new understanding of the behavior of the system. In the case of traffic simulation, whether it is urban traffic or air traffic, one can learn a great deal by viewing the intermediate steps through which the system is going.

For example, one might observe that due to some latency in traffic lights, some cars happen to jam an intersection, which in turn might cause a total breakdown of the traffic flow. If the conditions of cause and effect are not known in advance, global measures are not enough to explain this kind of behavior. The only way to understand the system behavior is by viewing it. By watching the behavior of the system, one might observe a pattern which he never expected to find. These behavior patterns, which are not known before they are observed, rely very highly on the intelligence of the human and his ability to recognize patterns he never looked for.

If the behavior patterns are known in advance, one might assign the computer to look for them, measure them, and report them. However, in many cases the internal patterns are not known and one has no idea what to look for. The visual graphics simulation allows one to recognize behavior patterns which he never expected to find and watch them develop. This recognition leads to an understanding of the system.

It is often the case that systems are "tuned" by readjusting some parameter which defines and controls each element. One has a very rough idea of what happens to the system as a result of this tuning. A good way to understand the effect of small individual changes on the system is by visually examining the behavior patterns and noticing the differences due to the effect of the tuning.

3.1.1 INTERACTIVE URBAN TRAFFIC SIMULATION

Urban traffic is an example of a complex system for which simulation techniques are appropriate. There are so many individual vehicles and so many variables determining their behavior that a precise analytic description of any system is impossible. It also seems that the simulation should be an interactive one because many traffic problems can be formulated intuitively, and a man's daily experience and perceptions should be utilized in the problem-solving process. This interaction is best served by a graphical display of the traffic flowing through the area under consideration. Traffic patterns are easily recognized when displayed, and factors of cause and effect are evident. A man can easily apply his perceptions and knowledge of the problem once it is presented to him in such a natural way.

We have constructed such a graphical simulation of urban traffic. One begins by specifying the street map to be considered. This is accomplished graphically, by means of a tablet. The user need only specify where he wants the streets to be and in what direction, and the program draws in the streets and intersections for him. In addition, it will draw center medians and traffic signals automatically. This process is interactive; the map may be edited at any time. After the street map is drawn, the program enters the simulation phase (fig. 1). In this stage, one scope shows a street map with cars traveling through it, and another shows a control panel with bar graphs of various descriptive parameters (fig. 2). As the simulation proceeds, the user may change the traffic density in different directions, or the speed, or other characteristics. This he also does with the stylus, merely pointing to the end point of the bar graph and moving it as he wishes. He may also turn the traffic signals red or green by pointing to them with the stylus.

Finally, it is possible for him to specify automatic settings for the traffic signals. He does this by drawing a bar graph of the times during a fixed-length cycle when the light is to be green and when it is to be red. He also has the facility of assigning the same setting to other signals, or the same setting with a fixed delay time. He may also specify that certain signals are to be given the same settings and then perform the above operations

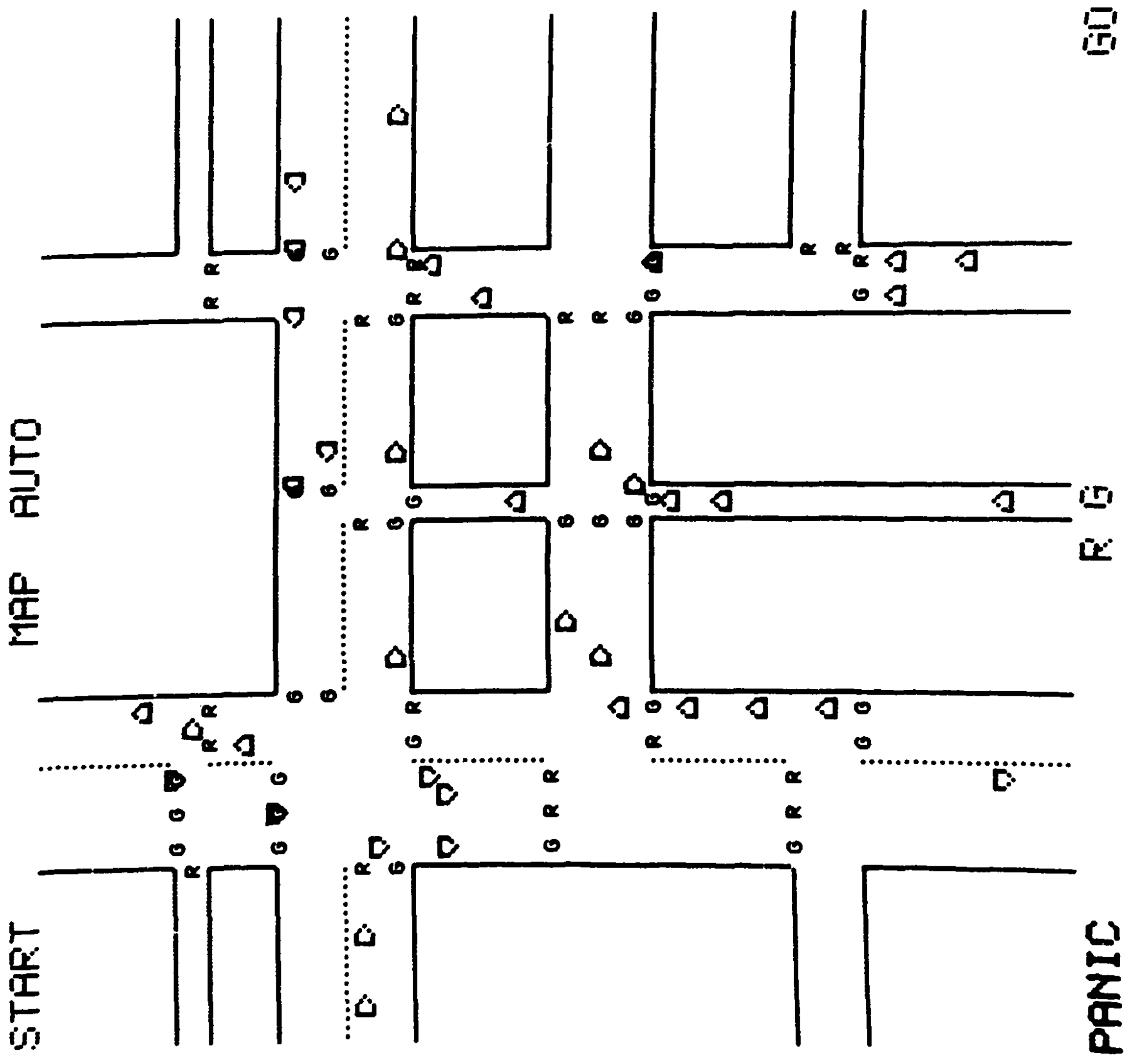

Figure 1. Urban Traffic Simulation

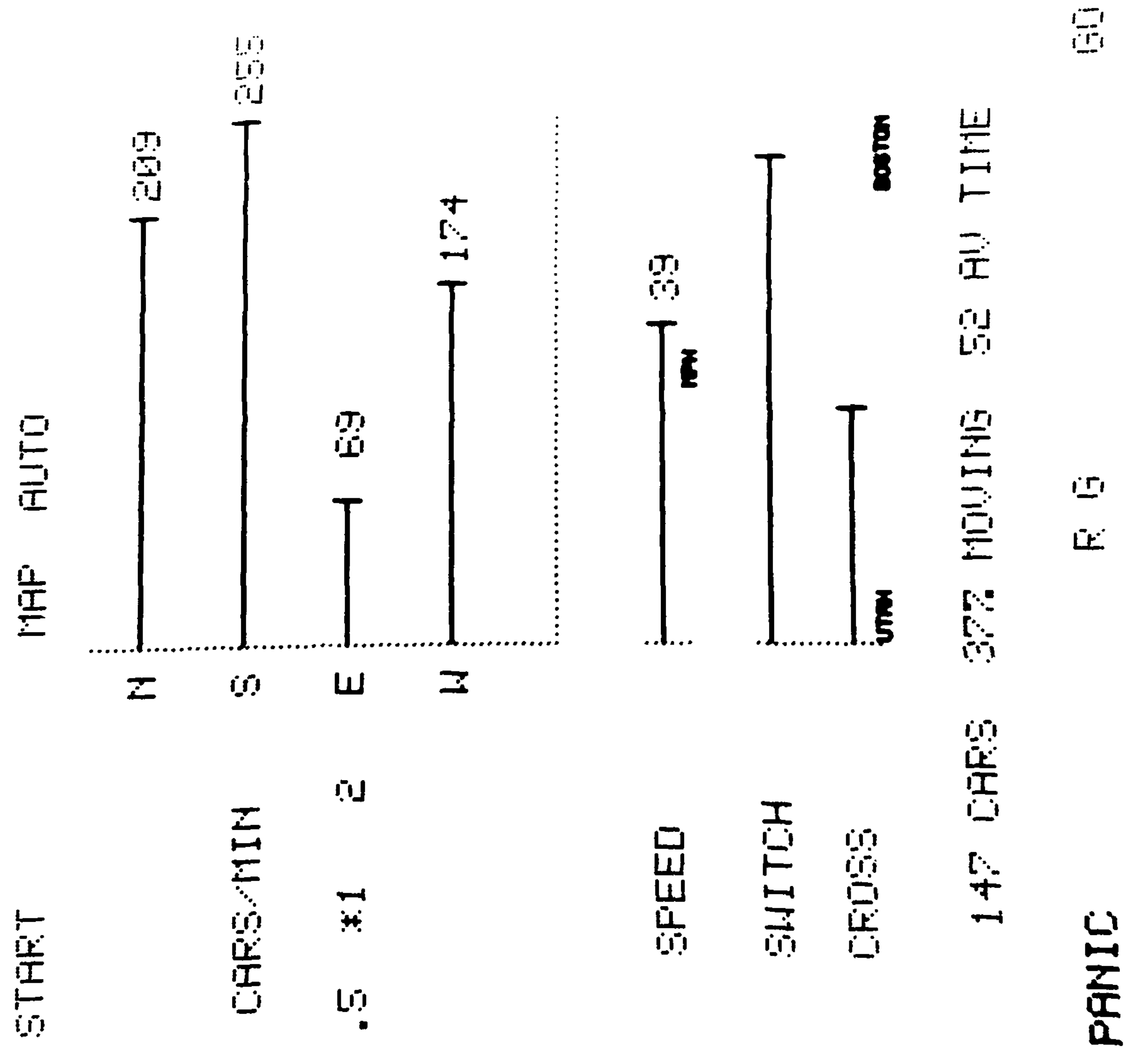

Figure 2. Control Panel for Traffic Simulation

on groups of signals rather than single ones. In this way, it is relatively easy to construct a strategy of traffic signal settings for a complex network of intersections. (fig. 3)

The program is designed so that it is natural to use interactively. After specifying one map, the user can try different signal settings under different traffic conditions to find appropriate means of control. He sees the effects of these changes in real time, as traffic flows through the network. He may return to draw in a new map and alter his strategies further, all in an interactive manner. We have found this approach to be a very valuable one in formulating and solving problems in urban traffic.

3.1.2 AIR-TRAFFIC CONTROL SIMULATION

The air-traffic control problem is a unique problem in the sense that it involves a very complex system of many airplanes sharing the same air space concurrently. In order to describe the system of the air traffic, one needs a dynamic tool which enables him to describe in real-time current positions of many airplanes which move in different directions at the same time. There is no way but graphically to describe the state of the system at any time. In real life the way the air traffic system is described is by graphical means, the radar which is used by the controllers. The control information which is issued is in the form of instructions to the airplanes telling them positioning and timing information, issuing "vectors", instructions for turning, and so on. Because of the nature of the problem, it is desirable to have facilities that enable one to communicate with the system graphically for input control information and to receive the state of the system at any point. For

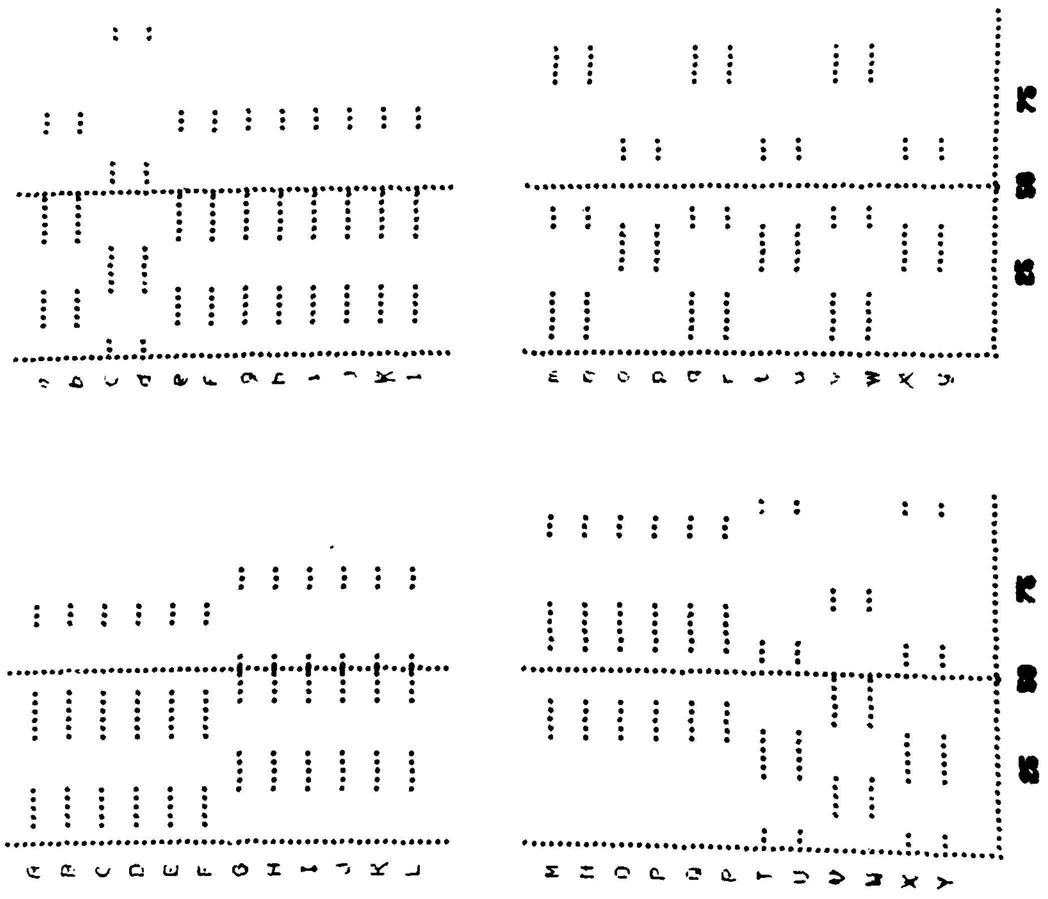

Figure 3. Signal Settings for Traffic Simulation

example, a controller should be able to define a route for an airplane merely by drawing the route on the face of a scope rather than verbally describing it. Collision hazards should be represented to the controller by showing him two airplanes whose routes tend to merge, and perhaps flashing some warning lights to attract his attention to this fact. The interaction between the controllers and the real airplanes should benefit from the use of the graphics as well. The controller should be able to point to an airplane rather than calling it verbally. This assumes, of course, that the system behind the graphics is aware of which airplane is where, and can automatically issue some communication to this airplane upon graphical request of the operator. In order to demonstrate these ideas and to provide a training environment, a computer program was developed in our laboratory.

In the first phase of the program, one can define the map of the area in which he wants to operate, and assign it any arbitrary shape. One can define the shape and the position of the airports to be included in this area. One can define the "Victor" and the "Jet" airways intersecting this area, and can define standard holding patterns. Navigation aids can be introduced into the map in the shape of triangles and squares. This definition stage is, of course, interactive. One can change his mind during the definition stage or later by editing the map, changing it, deleting obsolete objects, and adding new features to it.

After the definition phase, the operation phase begins. This operation phase requires two people to operate it. One is simulating the air-traffic controller and the other one simulates concurrently all the pilots of all the airplanes in the area. The "pilot" can issue routing instructions to each airplane in graphical form. The routing instruction may have the form of "climb and maintain flight level 200", and "follow victor 20, turn to victor 16 at station x", etc. The "controller" can see on his scope the position of each airplane and can interrogate these airplanes graphically, requiring information about altitude, speed, identification and so on. Ideally, the controller should be able to express instructions to the airplanes graphically. However, in order to simulate closely today's systems, the program does not automatically carry the graphical instructions of the "controller" to the airplanes, but the "controller" has to issue them verbally, as if he were talking on the radio to the pilot. The "pilot" then can apply these instructions to the airplanes, exactly according to the "controller's" instructions or he may deviate from them. This way the "pilot" can simulate misunderstandings between the air traffic control and the pilot in the air. The only way that the "controller" can find out about these misunderstandings is by noticing, on his "radar", that some airplanes do not follow the instructions that he had issued before. All communication with the airplanes either by the "pilot" or the "controller" is very natural. In order to specify an airplane all they have to do is to "touch" the airplane on the scope with a stylus. All control information is requested graphically, and the flight paths of the aircraft on the radar screen provide the necessary feedback.

3.1.3 LOGIC DESIGN

Computer graphics lends itself perfectly to use in logic design. Computer hardware is usually designed by using two dimensional sketches of combinations of logic elements. Computer graphics can be used in order to help draw the logic elements and draw the connections between them. In order to use the system for design, one needs the capability of defining different logical elements, defining various interconnections and groupings and defining levels and waveforms to be applied as input. The interactive design requires a constant feedback from the system. Upon completion of drawing some subcircuit, one may wish to see the operation of that circuit by applying some inputs and observing the outputs. The same system can be used not only as a design aid, but also as a teaching aid. Introducing students to logic design by an interactive system like this which visually displays each state of the system is most helpful.

The interactive system for logic design, which was written recently in our laboratory, enables one to define these logic elements as combinatorial logic units, i.e., units whose outputs are boolean functions of the current inputs. One can define any boolean function as a "gate", assign any arbitrary shape to represent this "gate", and then use this shape wherever desired. One can watch the values of any output at any time. It is possible to group together several elementary gates into a subsystem and watch the operation of this subsystem as a single unit. (fig. 4)

One observes the values of any output by "attaching" a scope symbol to any desired point. The scope will show either zero or one according to the current logic level of this point. The power of the system is due mainly to the simulation program behind the graphics. This capability of interactive simulation makes the program a valuable aid to hardware design, in addition to its use as a drafting tool.

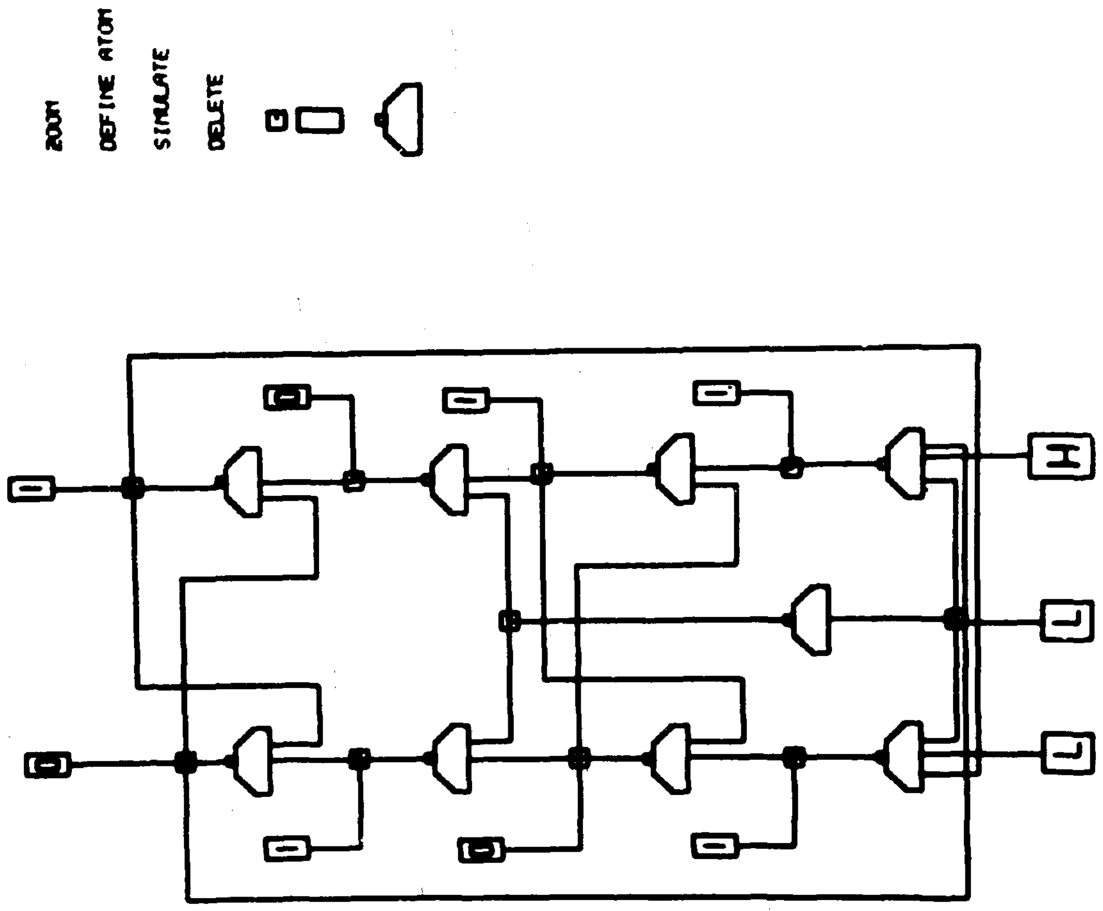

Figure 4. Logic Design

The program allows the user to think that he has a very large drawing area. If the drawing is larger than that which can conveniently fit on the scope, he may slide his "window" up or down, right or left, or in or out; i.e., he may "back off", so that he sees a larger area, but the picture is reduced in size. He may also examine sections as closely as he wishes.

3.2 REAL SHAPES IN REAL TIME

Many applications require graphic representation of real-world shapes in real time. The most outstanding application is visual flight simulation. The purpose of this application is generating the visual scenario, as should be seen by the pilot "flying" a simulator, in space according to some flight dynamics of the vehicle which is simulated. The traditional flight simulation techniques either neglect the visual altogether and simulate only the instruments, or use old-fashioned physical models in order to produce the visual scenario. The new digital computer graphics approach to this problem is better in many aspects than the old approach. First, the digital approach requires no physical models and no "moving parts" in the system. Second, most important, the digital computer introduces flexibility which is limited only by the capacity of the computer. There is no inherent complicated mathematical difficulty in producing the visual scenario. However, performing the three dimensional operations required for as much data as needed in order to get a realistic picture, might be too much load for a computer. The difficulty is not the complexity of the computation, but the frequency with which this computation has to be repeated. Due to recent innovations in technology and in the understanding of the problem, this task is now possible.

Another visual problem for flight simulation is synthetic generation of radar images in addition to, or instead of, the out-of-the-window view. The problem of generating radar images is similar to the problem of generating the out-of-the-window images. Both problems consider simple perspective transformation which has to be repeated many times. The number of these repetitions depends on the number of data units in the system. In both cases in order to get realistic images, a large number of data points is necessary. In addition to simulating the radar system, we also simulated a technique to convert radar images (PPI) into perspective images. The transformation of PPI images into perspective images is very important and very helpful for pilots on final approaches to airports. It was recently announced that such a system will be used in future airplanes.

We have two projects on this subject, one in flight simulation which is described in 3.2.1 and the other in radar simulation which is described in 3.2.2.

3.2.1 VISUAL FLIGHT SIMULATION

We have several PDP-1 programs for visual flight simulation. All of them allow the operator to "fly" an airplane and watch the flight instruments and the world through his window. The programs are divided into two main sections. The first is the dynamics simulation of the flight; the second is the graphics for the visual displays. The dynamics simulation scans the controls as operated by the "pilot" and according to their positions computes the powersetting of the airplane, its position, its speed, and its altitude at any instance. The model used by these programs assumes very simple dynamics somewhere between a "three-dimensional car" and a small airplane. The time scale of the simulation is modified for the convenience of the pilot.

The program allows the pilot to take off from an airport, to fly around it, and to land on any of the runways in any direction he wants. The program does not allow the pilot to be on the ground anywhere but on the runways. It also checksthe impact of the airplane with the ground upon landing and complains if it exceeds some maximum value. The program also does not like "Chinese landings" (one-wing-low) and does not like exceeding the maximum allowed speed.

The part of the program which simulates the dynamics is a module which can easily be replaced in order to simulate another kind of airplane. By varying this portion of the code, one can simulate anything from helicopters to supersonic jets. The second part of the program, the graphics display part, is the more interesting one. This part of the program is responsible for displaying both the flight instruments and the out-of-the-window display. The flight indicators are supposed to resemble the true indicators which are used in airplanes, like artificial-horizon, powersetting, air speed, a radio direction finder, a compass, and so on.

In the age of heads-up displays, this program is a natural environment for testing ideas for new types of indicators. Since all the data is gathered (or generated) by the computer, one can display some new kind of indicators instead of, or in addition to, the old indicators. The new indicators can show some new combinations of data which used to be displayed on the old indicators, giving the pilot direct information about the important flight parameters. Consider, for example, the air speed. Since the system might know the indicated air speed, the attitude, outside temperature, and the altitude, the system would compute the true air speed and display it to the pilot in addition to the "indicated" air speed. Together with the true air speed, the system could compute the stall speed for any attitude and display it next to the true air speed or even display the ratio between these two. We are aware that such indicators do not exist in today's airplane; however, in the future airplanes, where all the information is centralized by some digital system, there is no reason to display only the information which is directly measured, as it used to be in the Wright Brothers' era. Future airplanes will have on their heads-up display new indicators which do not exist today and give more information to the pilot, freeing him from computing this information manually from the old displays. Our program is a natural environment to experiment with this kind of indicator before trying them on real airplanes.

The second part of the display program generates the out-of-the-window view. The out-of-the-window view can be generated for any width of viewing field, up to 180 degrees. It is important to supply to the pilot not only the front view, but the side views in addition. In the case of fighters, it is eminently important to supply a top view, and in the case of VSTOL airplanes it is important to generate also the bottom view. It turns out that generating displays requires a computation which is less than linear in the number of windows displayed.

In addition to displaying the indicators and the out-of-the-window view, it is possible also to generate some other interesting displays. For example, one can display the gravity acting on the airplane, the drag as acting on each wing, the lift as generated by each wing, the act of the power, the centrifugal force, and all theother forces acting on the airplane. This display might serve very well as an instruction aid for student pilots.

Another possible display is an extrapolation-forward, trying to predict the path of the airplane in the next few minutes based only on information which might be available to a ground-based radar system. One can simulate systems like that which might be used by controllers on the ground to predict the position and the speed of the airplanes in an effort to recognize and eliminate collision hazards.

3.2.2 RADAR SIMULATION

Computer graphics is the most natural environment for radar simulation. It is possible to generate on the face of the CRT the real world views. One can try to be as close to it as possible but it is never just the same as viewing the real world. Viewing the CRT for radar operators is exactly the same whether the CRT is driven by a radar system or by a computer. It is possible to use computers to simulate radar systems, and use the output of the computer on exactly the same medium as used by radar systems, namely the CRT. In the case of a perfect radar simulation system, it should be impossible to tell whether the images are produced by real radar systems from the real world or by computer systems from data stored in digital memory.

Generating radar images is very similar to generating halftone displays with hidden line elimination. It is not surprising to find out that special purpose hardware can be built for radar simulation which resembles very much the special purpose hardware which can be built for hidden line elimination. The hidden line elimination in the case of radar display is much simpler than the general hidden line problem, because of the nature of the display objects, namely ground terrains.

We have several programs devoted to that project. One of them simulates the system, allowing two operators to "operate" two boats in an ocean around several islands. Each of them is equipped with its own radar system on which it can view the islands and the other boats, if visible. The third display shows the relative positioning of the boats in respect to the islands and the ocean. This system, if properly modified (which can very easily be done), can serve for the Coast Guard, for example, in training for docking and maneuvering in harbors or narrow bays and narrow waterways in night and fog conditions, using only the radars.

The other radar program allows the user to define data on the ground, select his position in space, x, y and altitude, and watch on the several possible displays of this system. All these displays are based on information as gathered by conventional radar systems; however, this information which is input from the radar antenna canbe displayed in several techniques. It can be displayed as a PPI, or as an A scope, or a B scope in presentations. In addition it can also be presented in a perspective way, which we call the C scope. The advantage of the C scope is that it presents the pilot a view that he could see with his own eyes, if it was not night or fog conditions.

This system can be used both for flight training, especially navigator training, and for briefing of pilots before going to new airports.

3.3 ANIMATION

Visual animation is a unique tool for education. One can use computer animation either to educate other people, or to educate himself to better understand a process. The impact of animation on education in certain fields can be compared to the impact of the blackboard. Just as words alone are not enough and not adequate to explain two-dimensional structures, so is the blackboard not adequate to explain any evolution process or anv dynamic process which changes with time. Computer graphics is so powerful not only because of its ability to display arbitrary shapes, but mainly for its ability to display changes dynamically.

It is not just a mere coincidence that in many languages the word "see" and the word "understand" are synonyms. In many cases to see is to understand. For these cases computer animation might be most beneficial.

We have several animation programs; one of them is a program described earlier for logic design. This program, in addition to being a tremendous aid for computer-aided logic design, can serve quite efficiently as an animation tool for teaching logic design, or for understanding it. A further step in the same direction is an animation of a micro program computer. This program is described in 3.3.1. Another program animates the operation of an assembler. It shows visually how an assembler works step by step. This program is described in 3.3.2. A further step toward compilers is a visual animation of a parser, discussed in 3.3.3.

3.3.1 SIMULATION OF MICRO-PROGRAMMED COMPUTERS

A program was written to simulate visually the operation and the design concept of a micro-programmed computer. The computer chosen to be simulated is a hypothetical one which has a flexible structure which can be easily modified. The graphics is used to display the internal registers of the computer and to show the data traffic between them. Each machine instruction of this computer is composed of several inter-register elementary operations. The elementary operations are symbolized by arrows between the sources and the destinations of the data. The student using the system can watch how each machine instruction, a macro-instruction, is decomposed into the elementary operations which compose it, and how the control executes them one by one. The purpose of this program is to explain visually the operation of a micro-programmed machine such that the student does not need, in his understanding, to apply to "black-boxes" which perform magic things like instruction decoding, operand fetching, indirection, etc.

The student sitting in front of the system can use the graphics not only for watching the machine operation but also for peeking into any area of memory and changing it if so desired.

After working with the system for a while, a student can gain a very good understanding of machine structure, the different machine registers, different addressing schemes, and trade-off between several different instruction repertoires.

3.3.2 ANIMATION OF AN ASSEMBLER

The next step after teaching a student how computer hardware works and how the instructions are executed is to teach him how the symbolic instructions that he writes are translated into binary machine instructions in memory. In order to accomplish this task, a graphical animation of an assembler allows users to write an assembly code which appears instanteously on the scope in front of him. The assembly language used allows several instructions and several pseudo-instructions like reservation of blocks and literal definitions. When the input phase is completed, the assembler starts each operation displaying each step on the scope. During the first phase the symbol-table is built. The computer scans the instructions one bv one and assigns locations to them according to their nature. A student can watch on another scope the symbol-table being built, including all symbols and literals. In the next phase the computer assembles the symbolic code into computer binary code. Each instruction, when encountered in the assembly code, is searched for in the operations-table until its binary equivalent is found. If it is not found, an error message is compiled and displayed. If the operation is found in the instructions table, it is copied into the right place in core, and then a similar search is executed to find the address part of the instruction. The student can watch this search done. If this search is successful, then the binary address is put together with op-code to make one machine instruction. If this search fails, another error message is compiled and displayed.

3.3.3 ANIMATION OF A PARSER

The next step after teaching and visually demonstrating to students the operation of an assembler is teaching the operation of a compiler. The most interesting part of the compiler is the parser used. In the program written here, only the parser was animated. This parser uses a bottom-up procedure. The operator can define the grammar of the language, which appears instantaneously on the scope when defined, and then can supplv a target string. The parser parses the string according to the grammar. After each parsing step, the partial trees are displayed, illustrating the operation of the parser. The parsing procedure is successful if, and only if, all the sub-trees which merge from the target string are combined into one tree whose root is the root symbol. (fig. 5)

By watching the trees being built and being backed up, the student gets a clear understanding of the advantages, and mainly the disadvantages, of this scheme. Upon changing the tactics from bottom-up to top-down, for example, and watching the results, the student is able to compare the two techniques. Watching the difficulties through which the compiler has to go can indicate to the student the disadvantages and the weak points of the given grammar.

It is possible to evaluate the grammar by measuring how manv steps it takes to parse a given string. By doing that without animating and displaying each step, the user does not get an understanding of what is going wrong in the scheme.

3.4 MANIPULATION OF SYMBOLS

A very important class of computer graphics applications are those concerning graphical manipulation of svmbolic shapes. These symbolic shapes most often have some spatial relation between them and are well suited to computer graphics. Examples of this would be electronic design, decision trees, musical notes, notation of organic chemistry, and so on, where symbols represent modules that have nothing to do with their real shapes. All these applications have in common the need to define symbols of arbitrary shapes, position them arbitrarily in some space, and define relations between them.

It is very important to have systems which allow the user to input his ideas and his instructions to the system by using the same svmbols he uses for communications with other people or even for writing notes to himself.

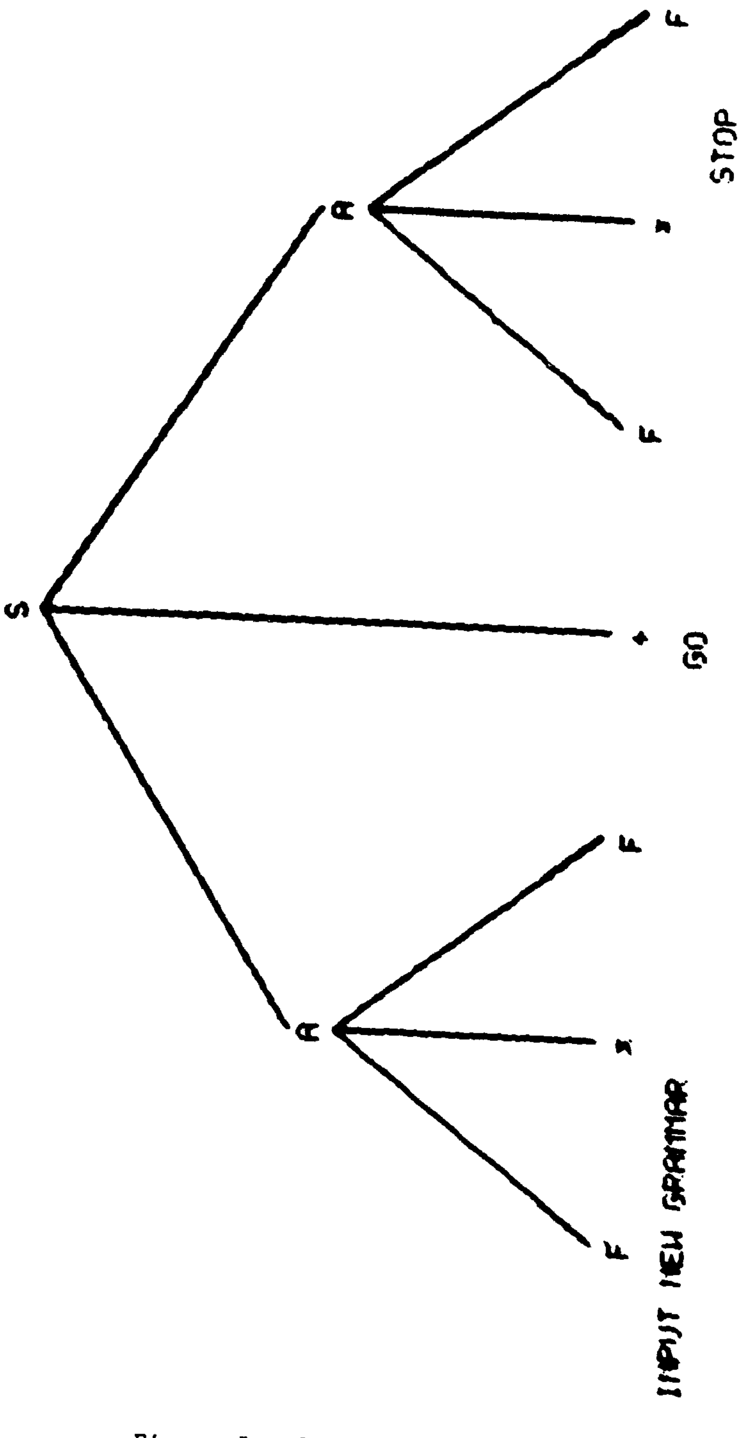

Figure 5. Graphical Parser

It is also most important to have the system output its findings by using the same notation. If the system has these two properties, then people who are not familiar with the peculiarities of computer languages but are very familiar with applications can easily use the computer without bothering about learning new symbolics and new ways to express themselves.

A tremendous help along these lines is a character recognizer. This recognizer is used for free-hand character input to the system. Free-hand input is especially important if the size, the orientation, and the position of the characters has some meaning. Mathematical formulations and some programming techniques require input of characters together with shapes, size, orientation, and positioning. For these applications, a character recognizer is most useful. Mathematical notations like integral signs, sigma signs, continual fractions, exponents, subscripts, and superscripts are two dimensional forms of writing formulas. It is possible to linearize all these two-dimensional notations into one sequential string; however, this is very unnatural and inconvenient for most people. Free-hand input of characters can be also most useful for writing programs by writing some program statements into boxes of a block diagram. However, it is doubtful if free-hand writing is the most efficient way of writing general code, instead of using teletypes or typewriters. We have several programs which demonstrate the ease and convenience of using free-hand input characters used for mathematical notations, and for writing programs in a two-dimensional programming language. In the following sections we will discuss some other uses of symbols manipulation, such as decision trees (sec. 3.4.4), musical notes (sec. 3.4.2), organic chemistry (sec. 4.3.7), or mathematical data smoothing (sec.3.4.1).

3.4.1 DATA SMOOTHING AND CURVE FITTING

A man, when hearing a crunching sound in the driveway upon the return home of his wife, can tell if the car's fenders are "smooth" or not by simply looking at them. Similarly, a scientist determines whether an experimentally obtained set of discrete data points is "smooth" by looking at them on a graph. In both instances, the verdict "not-smooth" results in the use of tools to provide corrective action to improve the situation, the tools being those of the mechanic in the former case and those of the mathematician in the latter.

What is common to these two events is the recognition that the property of smoothness is essentially a geometric concept which is often best determined on a visual basis. Mathematical aids are useful in this determination, but presently data smoothing is more of an art than a science. One generally has a notion in advance of what "smoothness" means in a given situation (e.g., a car fender ought not to look like a refugee from a home for old accordians) and then compares this notion with what he actually sees.

The use of computer graphics allows an individual to "see" his data points on a graph and determine if they are "smooth" or not. He can fit interpolating or approximating curves to the points and then visually compare such curves on a CRT to help him analyze the data. Furthermore, he can study different mathematical measures of smoothness, such as the sum of the squares of the third divided differences, by displaying and comparing various measures, as well as by obtaining reference values for curves which are known to be smooth.

The use of a graphics system allows an individual to try many different smoothing methods and compare the results in a relatively short period of time.

An interactive system lets him experiment with several different approaches and choose the one which is most promising for more detailed analysis.

A. Priver has developed an interactive computer graphics system utilizing the PDP-1 and the four DEC340 scopes to assist humans in efficiently "smoothing" and "fitting" curves to data points involving statistical or measurement errors. The user is embedded in an "application-domain" and is thus not concerned with the actual nature of the computer and the accompanying program. The user is able to input a discrete set of data points, change the values of these points (by editing), apply mathematical operations defining smoothing methods to alter (smooth) these points, and evaluate measures of smoothness and closeness of fit to compare different methods.

The system uses several buffer storage areas so that different results can be easily saved for comparison purposes. The values in these areas can be displayed in numerical tables and/or can be displayed graphically. The displays for each area are individually controlled so that any desired comparisons may be made. Additionally, a record of the sequence of operations used in obtaining the results shown is given on one of the CRT's. Provision is also made for storing (and retrieving) sets of points for future use by the system. (fig. 6)

Rapid response is achieved through the use of sophisticated data structures (for the graphic displays) and numerical techniques (such as the Fast Fourier Transform, for rapid computation of convolutions). Among the curves which can be fitted by the system are cubic splines, least-squares polynomials, and trigonomietric series.

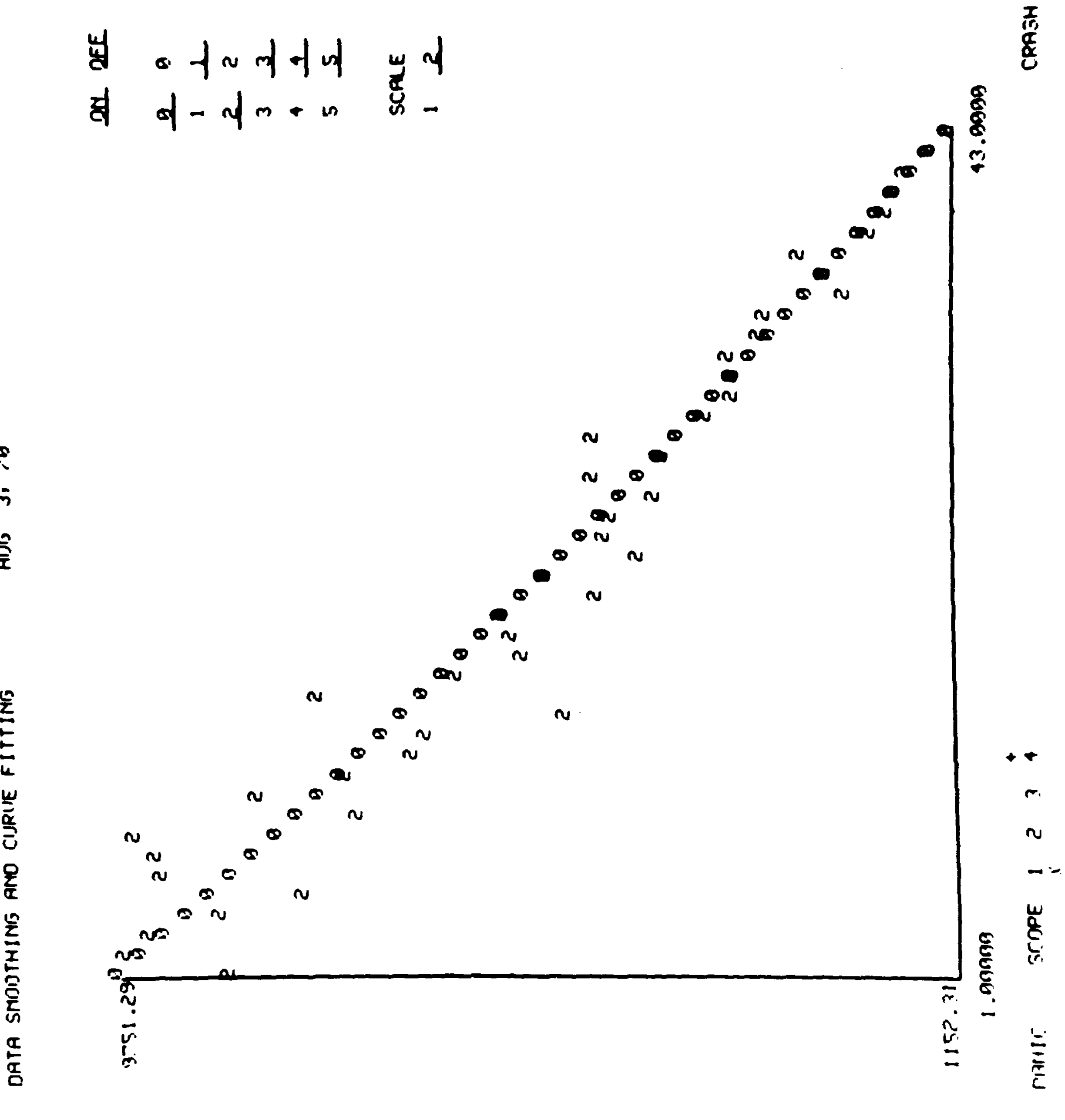

Figure 6. Graphical Data Smoothing

3.4.2 MANIPULATION OF MUSICAL NOTES

The program was written in order to enable musicians to input music by using the same notation with which they are familiar. By using the display system a musician can input a piece of music by writing notes with a stylus in just the same way he would write them on paper. These notes are stored in the computer such that they can be performed by the computer or, if so desired, be edited and re-performed. (fig. 7)

It is not clear that a system like this can improve the process of writing music by composers. However, it does demonstrate an important concept. This graphics system embodies a new kind of notation such that users can communicate in the notations without ever referring to any idiosyncracy of computer language. A musician with no knowledge of any computer system can learn to use the system within a few minutes.

3.4.4 DECISION ANALYSIS

The DECision and ANalysis program was a preliminary effort at the design of an interactive graphic system for aiding in decision analysis. DECAN is based on the use of a tree structure as the model of a decision problem. This form of analysis has been growing in popularity and is used in several courses at Harvard and M.I.T.

Unfortunately, as decision problems grow larger and more complex, the need for computational assistance from a computer quickly becomes evident. Ordinary batch-oriented and time-sharing computer facilities are not adequate. The decision analyst must be able to experiment freely with changes in the underlying structure of his model as well as the changes in individual parameters.

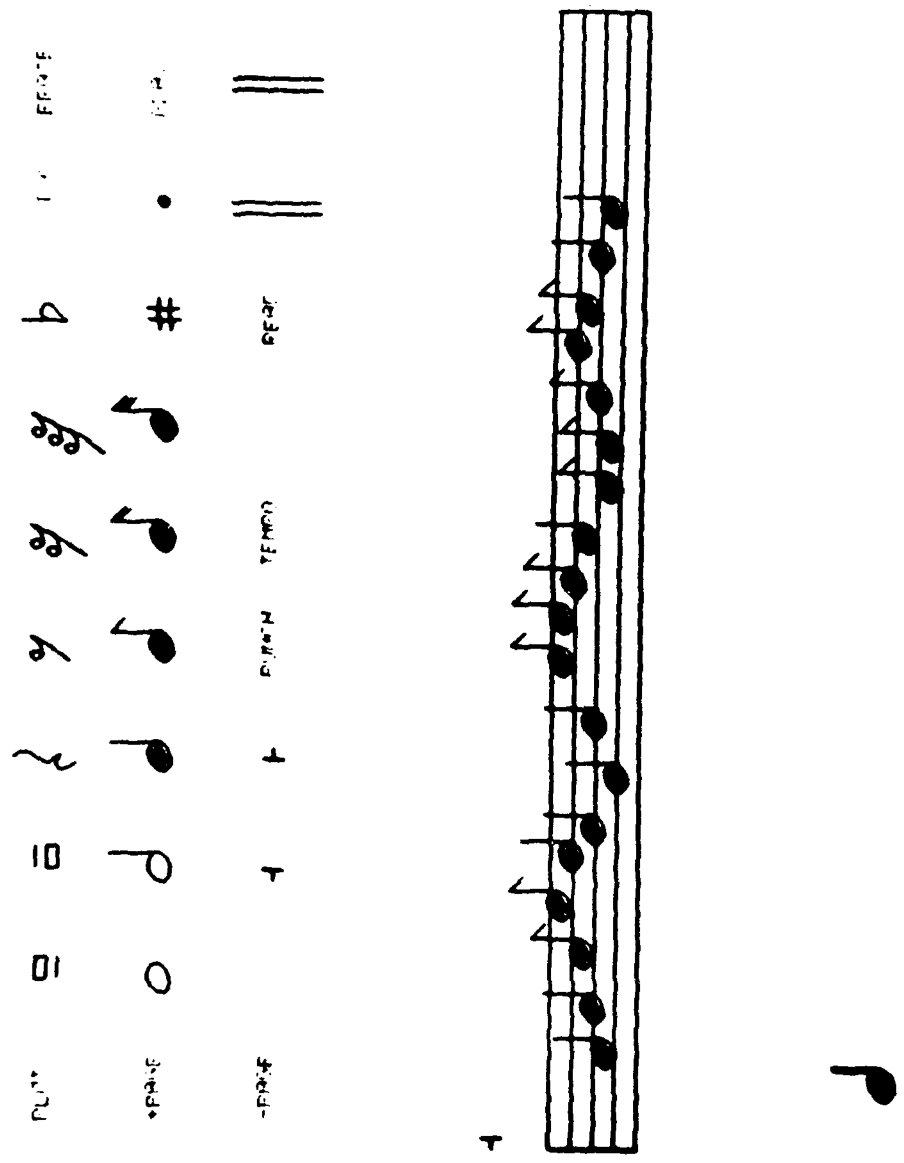

Figure 7. Computer 'lusic

He should be able to do this without the loss of the conceptual clarity provided by a two-dimensional representation of the tree structure Only an interactive graphics system can provide the analyst with this type of support.

A user of the PDP-1 DECAN program constructs his decision tree, labels its branches, and inserts parameters by means of light pen connected to the cathode ray tube display unit. If his tree grows too large to fit entirely within the display window, he can use the joystick to indicate the portion of the tree he wants to observe. After the model has been constructed, the program will compute the optimal decision and display it as a path through the tree. DECAN is very easy to use and can be learned in about two minutes by anyone familiar with the analytic techniques upon which it is based.

The current DECAN system is useful for only a highly restricted subset of decision problems. But it does indicate the potential for a more sophisticated system.

3.5 DESIGN APPLICATIONS

Computer-aided design is an area in which graphics is a natural form of communication. It can serve as the interface between man and machine, providing the designer with a familiar means of dealing with his work. Behind the graphical equipment stands a program specific to the application that understands the design and can interpret and measure it.

Computer graphics is useful to the design process both as a natural means of input and as a quick and flexible means of output. A typical system for graphical computer-aided design allows the user to enter his design by drawing it schematically. This representation might satisfy certain conditions according to the application (for example, only horizontal and vertical connections might be allowed in an electrical drawing). The system can help the designer in planning the physical form of his design and in producing complex diagrams quickly from simple components. It can relieve the designer of many repetitive tasks, and tedious calculations.

After the design is entered graphically, the computer program can evaluate it on the basis of certain rules and measurements. It can perform routine checks and calculations and supply the user with the results, perhaps in graphic form. In the case of civil engineering, this stage might involve stress analysis or weight distribution analysis. Any evaluations that the computer displays to the user should use the same symbolic form in which the design was represented by the user. In this way, the feedback is made very natural and easy to understand.

We made mention of two design programs above in the section on simulation of systems. These are the urban traffic simulation and the program for logic design. The traffic program allows one to specify the street map to be considered, the placement of traffic signals, and their time settings. Feedback from the program comes in the form of graphic simulation of the traffic moving through the streets. One is then free to modify the map or the signal timings and to see the resulting effects on traffic flow. The process of designing the map is made quite simple and efficient, as is the process of specifying time patterns for the traffic lights. Thus, the design of a network of intersections is made interactive, and the designer can see how good his plan is immediately, in graphical form.

The logic design program is similar in approach. One has at his command flexible but powerful tools for drawing a logical network. Individual units are specified by the designer, both in representation and in formation. The program is a considerable aid in drawing complex designs because one can draw repetitive structures simply, and make all connections neatly and systematically. Also, one can consider as large or as small an area of the "drawing board" as he wishes, and can move around it at will. Finally, one can apply any arbitrary signals to the circuit and observe the results graphically. Both of these programs constitute a tremendous savings in time and money over the more usual method of building the design and experimenting with its use. (fig. 8 and 9)

3.6 EXPERIMENTS ON VISUAL PERCEPTION

The space pen interfaced with the PDP-1 was first operational in September, 1970. It consists of three orthogonal plane microphones and a spark pen. The pen launches a shock wave from the hole in its tip each time a spark expands the gas in the pen. At the same time, three counters are started; each is stopped by a signal from the corresponding preamplifier when the shock wave first arrives at the microphone. Tests indicate that the point specification accuracy of the spen system is one part in 512.

As reported earlier (ESD-TR-70-202, p. 13), the display system uses the field sequential filter technique. It consists of using a hand held motor which rotates a filter disk in front of the eyes. As the eyes are looking through red filters on the disk, the computer display presents image elements to be seen as red, or having a red component of their color, and so on. Typically this is done 20 times a second, a speed easily fused in the visual

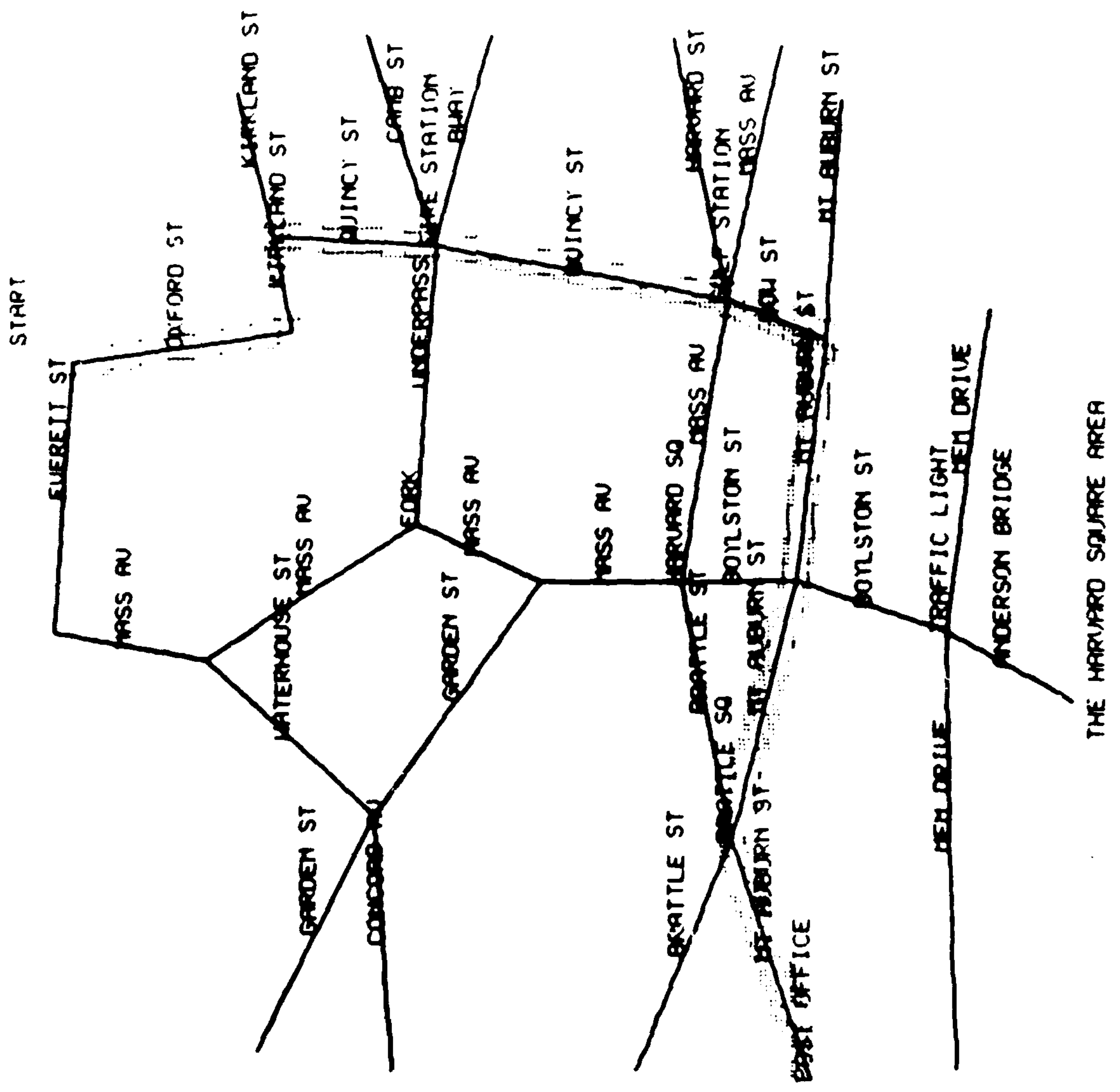

Figure 8. Street Map with Selected Route

```
      DIRECTIONS
START
TAKE OXFORD ST
BANG A LEFT ONTO KIRKLAND ST
TURN RIGHT ONTO QUINCY ST TOWARDS FIRE STATION
GO STRAIGHT AT FIRE STATION CONTINUING ON QUINCY ST TOWARDS GULF STATION
GO STRAIGHT AT GULF STATION ONTO BOW ST
TURN RIGHT ONTO MT AUBURN ST
GO STRAIGHT CONTINUING ON MT AUBURN ST TOWARDS BRATTLE SQ
GO STRAIGHT AT BRATTLE SQ CONTINUING ON MT AUBURN ST TOWARDS POST OFFICE
AND THERES POST OFFICE
YOU CANT MISS IT
```

Figure 9. Computerized Directions

system for simple image tasks, the observer seeing the additive mixture of colors in the image. The system also offers the possibility of stereo viewing, since areas of the filter wheel may be opaque, thus selecting the eye which will see the display, in addition to the color. Using a filter wheel which is half-opaque, one can do black and white stereo and reproduce all of the binocular rivalry experiments done during the past one hundred years. It has been interesting to observe that since the real world does not present us with rivalrous information except under extraordinary circumstances, persons viewing the binocular rivalry experiments often have differing responses, thus indicating that no inate interpretation system exists for this viewing task.

A moving dot experiment demonstrates two features of time-dependent viewing: We apparently integrate backwards from a "significant" perceptual event to process visual information residual in the neurophysiological system, and the cognitive field for recognizing simple forms is restricted to the small central visual field when slow writing rates are used in displays.

During the course of talks presenting much of the experience in using the display system for perceptual experiments, two viewing events have been acknowledged as unique. Using the fast phosphor display (60 microsecond decay, P-4 phosphor), one moves a dot in a line at speeds selected by some input device. The visual experience for velocities of about 10 cm/sec is to see a short line, from which the dot appears to then move along the rest of the path. This results from the delay time necessary to establish the involuntary smooth pursuit of the visual system. Moving the dot in a square provides four points, the corners as starting locations for lines resulting from this effect, and these are observed. Also related to the servo system, which smooth pursuit requires, is the observation of overshoot at the corners when vision is not fixated at the corner, but follows the moving dot. The surprise is that lines are observed coming into the corners; their length corresponds to times of the order of 50 ms. It seems that the corner event causes the mind to reconsider the information about the dot seen previously and to integrate it into the more informative feature, a line. This effect has been independently observed by a group working with John Ross in Western Australia.

A further remarkable feature of the dot moving in the square at speeds such that the square just seems to close (the lines from the corners seem to meet on all sides) is that one cannot recognize the square trace except when *fixating close to or within the square.* If one fixes his attention at a point some 10 degrees above the figure, the square merely appears as a collection of corners, the array differing from one subject to another. It would appear that the primitive "square detector" only operates for centrally fixated images within the time domain represented in this experiment.

A wide variety of related perceptual experiments have been performed, with results familiar from psychological studies done with exceedingly more complicated equipment. The moving picture effect, where a fast sequence of still pictures with subtle differences indicates motion, is available with the interesting flexibility provided by continuous adjustment of image displacement and framing rate. Adjustment of display timing can also produce "metacontrast" or backward masking events -- e.g., displaying a letter, then some 100 ms later a box surrounding it, makes the letter <u>un</u>observable to the human visual processing system. One may also simulate the display of multiple slides and observe the decrease in change rate as the number of slides increases. The very subtle and variable "subjective colors" are also available by producing flickering reference fields within which a flickering stimulus with increased extinction time brings a response, indicating that color is being observed from the time-dependent color vision system of the eye.

3.7 DESIGN OF COMPLEX ORGANIC SYNTHESES

LHASA (Logic and Heuristics Applied to Synthetic Analysis) is the name of an interactive Organic Chemistry program which has been gradually evolving over the past four or five years under the guidance of Professor E. J. Corey of the Harvard Chemistry Department. We give a brief introduction to the program environment, and then deal with some of the more recent developments in greater detail.

When a synthetic organic chemist analyses a synthetic problem he is basically trying to derive a sequence of chemical reactions which starts with structurally simple molecules, and gradually builds them up to a fairly complicated structure, called the "target" molecule. There may be any number of reasons why the chemist would want to synthesize a particular target compound, but that is of no importance here. What is of interest is determining one or more chemically valid reaction sequences ("pathways") to the target molecule. The approach chosen is to analyze the target compound, and generate a collection of molecules, each of which is one known chemical reaction away from the target. Then any of these is in turn analyzed as the target, generating another collection of precursors. The analysis continues in this fashion in the "retrosynthetic" direction until one or more simple, readily available compounds result. So during the analysis, a tree of organic structures has been grown. The branches of the tree correspond to, hopefully, valid synthetic pathways from the simple starting materials up to the complex target structure.

The chemist enters the target structure to the computer (Harvard's PDP-1) via a Rand tablet and stylus, and draws the molecule exactly the way he would draw it on paper. There are various stylus-activated "buttons" on the scopes to allow manipulation of the structure. The perception module of LHASA is brought into play at this point. It looks up the raw input data in the atom and bond table, and gleans from it synthetically significant structural information, such as functional groups, aromaticity, ring systems, etc. One of the chemistry packages is then swapped in, and it matches various features of the molecule with the internal reaction tables. When it finds a reaction (or transform, in the retrosynthetic direction) which it deems contextually reasonable, it applies it to the target structure and displays the new structure to the chemist. Typically it will generate about ten new structures from the analysis of the target. The "tree" is displayed on a second scope, with the target at the top, and the immediate precursors fanned out below it on one level. When LHASA can do no more chemistry on the target, it returns control to the chemist. He can then select any of the first level precursors and process it as above, causing the tree to grow down another level, and so on.

The use of computer graphics in this form represents the ideal of communication for this chemical application; it is for the chemist by far the most natural and efficient, and also the easiest means of structural communication.

The second phase of this project, now in progress, continues the use of graphical communication, but involves the use of a far greater chemical data base, a sophisticated program for automatic selection of chemical transformations and strategies, and very advanced software for internal evaluation during the problem-solving process. Use is still made of the venerable PDP-1 facility, but simultaneously the program is being rewritten and refined for Harvard's PDP-10 facility which is linked to the PDP-1 and associated graphics equipment.

The theory of chemical synthesis is being developed in an abstract and general form for the first time, and it is being put to the test of application in the developing computer program. A new and uniquely organized form of the chemical data base is also being elaborated. The data base now contains about 300 chemical reactions. At present the program (which is in highly modular form) runs to about 50,000 lines of code (in DECAL, the PDP-1 assembly language) and uses all the available memory of the PDP-1, including a 2.25×10^6 bit magnetic drum used with swapping as a virtual memory. The modular form of the program allows the continued use of the PDP-1 to develop new sections of the program, however.

The new version of the program being prepared for the PDP-10 is almost all in FORTRAN IV with a minimum in assembly language to reduce machine dependence and make the program relatively easy to use by others. It will probably not be complete, however, until the remaining major modules of the chemistry program are developed. The most difficult of these are the strategy and stereochemistry modules, which represent major undertakings.

In order to codify the huge collection of known synthetic reactions it was necessary to classify them according to the type of molecular substructure resulting from the reactions. The following five major classes have been established: (1) two-group transforms, (2) single-group transforms, (3) functional group interchanges, (4) functional group addition transforms, and (5) ring-oriented transformations. The first four classes are operative now in LHASA, and work is progressing on class (5). Transforms in classes (1) and (2) have the effect of simplifying the structures on which they are allowed to operate, and so, are said to be goal-directed. However, those in classes (3) and (4) simply modify certain portions of the molecule, and so either do not simplify it, or increase the complexity of the molecule. Indiscriminate application of transforms in those classes would be counterproductive to a "rational" synthesis. Therefore those transforms are relegated to subgoal status, that is, they are only allowed to go if they somehow set up the molecule for subsequent application of a class (1) or (2) transform which was previously blocked.

The method chosen to represent the transforms in the computer is quite interesting, and is the result of a great amount of theoretical work in the LHASA research group. Since there are thousands of known organic reactions, and probably billions of theoretically stable organic compounds, it would be clearly impossible to have a direct lookup scheme for each organic molecule. Instead only the substructures involved in the known reactions have been included in the tables. This obviously simplifies the storage problem; however, it is not enough. All organic reactions can be affected by electronic conditions in various parts of the molecule, and a chemist must take the rest of the molecule into account in deciding whether or not a reaction will take place at the desired location in the molecule. To handle these contextual considerations a table-oriented interpretive language has been developed which is quite readable to chemists with little or no programming experience. Once the program finds that the molecule has a certain substructure for which it has a matching transform, it then processes a set of questions and actions associated with the transform. These actions, written in the interpretive language, effect a modification of a basic transform rating. For example, if a transform is adversely affected by the presence of a certain functional group in the molecule, that question would be written in a table, and the transform rating would be decreased if the interpretor program found a group of that type in the molecule. Those transforms which fall below a cutoff rating are eliminated from further consideration. More recently, a new interpretive language has been designed to be used with the transforms of class (5). It is even closer to Chemical English than the first language, and is slightly Algol-oriented in its sentence and logic structure.

In the last year, much work has been done in teaching LHASA about stereochemistry. In certain molecules, such as those which have asymmetric centers, stereochemical considerations outweigh all others in designing an effective synthetic strategy. The first problem was to design a simple and familiar method of inputting stereochemical information via the pen and tablet. This can be done now, using buttons which display bonds as being dotted (behind the plane) or wedged (sticking out of the plane). This is quite similar to the way a chemist would indicate stereochemistry on paper. From this basic information, a stereo perception module perceives cis and trans ring appendage relationships, and cisness and transness at double bonds. It will also perceive absolute configurations at asymmetric carbon atoms.

Development of stereochemical perception and manipulation techniques was a necessary precursor to the present implementation of ring-oriented chemical strategies. One of the ring transforms, the Diels-Alder reaction, has been the subject of recent work in the LHASA group. Because of its synthetic importance, it is being handled differently from transforms in other classes. Instead of requiring an exact match in the molecule, the program will go to unusual lengths to set up an exact match, if a part of the molecule even vaguely resembles a matching situation. Using the interpretive language just developed, the program calls subgoals to modify certain offending regions of the molecule, until it is finally clear to perform the Diels-Alder disconnection. Each of the modifications of course must correspond to reasonable chemical reactions. The effect of this sequence would be seen on the tree as a branch growing down several steps, instead of growing across on one level. Eventually this technique will be extended to other major transforms.

Much of the work in the past has centered around adding new reaction classes to the program. What is being implemented now, as indicated above, is a collection of synthetic strategies. These strategy modules will examine the structure and decide which collections of transforms would be best to apply, and how hard to work to clear the way for the application of certain powerful transforms. Look-ahead techniques will lead the analysis through seemingly counterproductive intermediate steps, if the end result is a drastic simplification of the structure. At all times of course, the chemist can retain full control of the direction of the analysis. The philosophy so far has been toward the implementation of new strategies which will guide the search automatically, but still leave the chemist the option of manually selecting certain chemistry packages, as he has in the past.

With regard to the "intelligence" of the analyses produced by LHASA, a number of compounds that have appeared in the Synthetic Organic literature have been processed and have often come quite close to the published syntheses. In some cases LHASA has come up with more elegant solutions. While there are still many glaring holes in LHASA's chemical knowledge, its performance within its sphere of expertise has been very reassuring.

In the longer run it is envisaged that sometime during the period 1980-85 computer-assisted synthetic analysis will be a standard technique used by scientists everywhere. The technique will make possible tremendous savings in time and also permit the chemist to develop more sophisticated and more efficient chemical syntheses at lower cost.

As the objectives of the program are attained, it is likely that other related projects will be initiated. Most interesting of these at present is research on the use of computer graphics in computerized instruction in chemistry and in chemical information retrieval. Also of substantial potential importance is the addition of a learning capability to the problem-solving program.

The original publication on the LHASA program (then called OCSS) appeared in SCIENCE in 1969[1]. Three more major papers have been accepted by J. Amer.

[1]E. J. Corey and W. T. Wipke, SCIENCE 166, 178, (1969).

Chem. Soc. for publication in early 1972. The first of the three[2] deals with the graphics section of LHASA and discusses its interactiveness with the chemist. The second paper[3] discusses the various perception modules and perception algorithms. It also describes the data structures chosen for the different parts of the program. The last and largest of the three[4] details the chemistry packages, and discusses the table-oriented chemical language that was developed. A fourth paper[5] will also appear, describing a new ring perception algorithm of general applicability. Currently work is being done on a second set of papers dealing with stereochemistry, the theoretical work on the Diels-Alder reaction, strategic bond perception and application, and new algorithms developed for the FORTRAN version of LHASA being written on a PDP-10 computer. In addition, smaller articles have appeared in Computer Decisions (1970) and Scientific American (1970). Various aspects of the LHASA project have recently been presented by Professor Corey at:

(1) American Chemical Society symposium in Rochester, October, 1970.

(2) Hoffmann-Laroche in Basel, Switzerland, October, 1970.

(3) A seminar at Northwestern University, September, 1970.

(4) American Chemical Society symposium, Los Angeles, March, 1971.

[2] E. J. Corey, W. T. Wipke, R. D. Cramer, and W. J. Howe, Journal of American Chemical Society, 94, (1972).

[3] E. J. Corey, W. T. Wipke, R. D. Cramer, and W. J. Howe, Journal of American Chemical Society, 94, (1972).

[4] E. J. Corey, R. D. Cramer, And W. J. Howe, Journal of American Chemical Society, 94, (1972).

[5] E. J. Corey, G. A. Petersson, Journal of American Chemical Society, 94, (1972).

(5) The Centenary Lecture at the Chemical Society meeting in London, April, 1971.

(6) A seminar at the University of Michigan, May 1971.

(7) The 23rd International Congress of Pure and Applied Chemistry in Boston, July, 1971.

SECTION IV

THE ECL PROGRAMMING SYSTEM

The ECL programming system has been designed as a tool for tackling "difficult" programming projects, that is, projects on which existing languages could be used only with considerable waste in machine or programmer time.

Such projects include much of the frontier of computer technology; whenever several application areas are conjoined and whenever solution of a problem requires linguistic development -- in algorithmic notation or information structures. Examples range from the management of large scale distributed data bases to applied artificial intelligence.

Specifically, projects of this nature are systems characterized by two requirements:

(1) Considerable experimentation is required to develop the system; that is, the design and development of the system must go hand in hand. Typically, this occurs when problems are so complex that significant computer assistance and experimentation are needed in system design.

(2) When a complete system is ultimately designed and programmed, it must be possible to take the working programs and produce a highly efficient product - both in machine time and space - with no change to the basic algorithms or their representation.

The ECL system has been designed as a vehicle for such undertakings. At the present time an experimental version of the system is operational -- a version which only partially meets the above requirements. Additional system development is underway and will continue for some time. This discussion outlines the goals of this work and the ECL system as planned.

The ECL programming system consists of a programming language, called EL1, and a system built around that language to provide a complete environment for the human-oriented use of the language. The system allows on-line conversational construction, testing, and running of programs. It includes an interpreter, a fully compatible compiler, and an editor -- all callable at run-time, on programs constructible at run-time either by the programmer or as the result of computation.

EL1 is an extensible language. Thus, it provides a number of facilities for defining extensions so that the programmer can readily shape the language to the problem at hand, and progressively reshape the language as his understanding of the problem and its solution improves. Like the familiar notions of subroutine and macro definition, these extension facilities allow one to abstract significant aspects of a complex algorithm. Such functional abstraction serves both as a representational aid and as a handle or the production of an efficient product.

Specifically, the language provides facilities for extension of four axes: syntax, data types, operations, and control.

(1) Syntactic extension allows the specification of new linguistic forms and their meaning in terms of existing forms.

(2) Data type extension allows the programmer to define new data types and new information structures whenever needed to model the problem at hand. El1 is significant in this regard in that considerable attention has been given to the efficient representation of programmer-defined data types. There is a special compiler for data-type definitions which computes space-efficient packing of structures into machine words, and generates machine code for rapid handling of objects and their components.

(3) Operator extension allows the programmer to define new operations on new data types and to extend existing operations to cover new types. There are two key points here:

   (a) Operators and procedures are not restricted to act on built-in types in the language but can, and in general will, take arguments whose mode is programmer defined.

   (b) Declarations can be made to allow the compiler to perform type-checking and type-conversion code generation. Hence, it is possible to write programs operating on extended data types whose execution at run-time is comparable to that for using built-in modes.

(4) Control extension allows the creation, deletion, and coordination of independent asynchronous processes, called paths in ECL. The extension mechanisms are sufficiently flexible that co-routines, the P and V operations of Dijkstra, multiple parallel returns, and path scheduling are all definable in the system as proper extensions. Hence, it is straightforward to program almost all known control structures as well as an unknown variety of others.

In addition to these definition mechanisms provided by the language, the ECL programming system provides a number of other handles which the programmer can use to extend and tailor the environment in which he operates. Many of the system's facilities are written in the language and hence open to argument, modification, and replacement by the programmer. These include the compiler, the editor, and most of the input/output and file system.

Extensibility alone is, however, not sufficient; its counterpart -- contractibility -- is also required. That is, having produced running prototype programs, it must be possible to subject these programs to a sequence of contractions -- commitments with respect to subsequent nonvariation so as to obtain a final system optimal for the project requirements.

The ECL programming system and the EL1 programming language have been designed to allow this. Programs can be run either by an interpreter or a fully compatible compiler. Compiled and interpreted functions can call each other in either direction. Compilation can itself be progressively refined. The programmer is free to supply as much declarative information as he wishes (or knows) at a given time and the compiler will do the best it can with the information given. Successive recompilation with additional information will produce progressively better code. Such contractions include: evaluation of arbitrary expressions at compile-time, fixing the values of procedures and operators, fixing the lengths of arrays, and fixing the data types of both local and free variables. For example, one can specify that the data type of a variable fall into any of the following categories: 1) it is completely dynamic at run-time, 2) it is restricted to a specific set of fixed modes, 3) it is a fixed mode but its length is not fixed, 4) its mode and size are both fixed. The programmer can choose, for each variable, in which category it is to fall, and he can choose again -- with minimal programming effort -- as the project progresses.

At the beginning of this section, we described ECL as a tool for tackling "difficult" programming projects. As evidence that this goal has been accomplished, we can cite an extension of ECL which was designed and constructed in two weeks by one of us; the extension provides the equivalent of the SNOBOL language designed and implemented on the IBM 7094 by Farber, Griswold, and Polonski at a cost of, we would estimate, at least two orders of magnitude more effort than was required using ECL.

In summary, the intended application of the ECL programming system is the programming project which would otherwise be prohibitively uneconomical. To this end, the ECL system has been designed to allow flexible programmer-oriented program construction and testing coupled with facilities for subsequent optimizing contractions which produce an efficient final product.

SECTION V

COMPUTER NETWORKING

5.1 NETWORK DEVELOPMENT

During the spring of 1970, the problem taken up by our seminar in operating system design was the interconnection of the PDP-1 and PDP-10, the objective being to give the graphics user access to the greater computational power of the larger machine and, conversely, to make the PDP-1 peripheral equipment (notably, the line printer, tablets, and displays) available to PDP-10 users. An additional motivation was that the LHASA project (see section 3.7, above) would soon expand beyond the resources of the PDP-1. Although a high-speed, half-duplex link between the two machines was under development, analysis showed that attachment of both computers to the ARPA network would yield significantly larger benefits. Accordingly, our main effort was directed toward implementing that method of interconnection. During June, 1970, four of our students participated in the (primarily) UCLA and University of Utah final development of the HOST-HOST protocol.

Design and implementation of the IMP service routines and NCP for the PDP-10 was accomplished by R. L. Sundberg, starting in July, 1970. At this time of writing, this work is finished, save for repairing any additional bugs encountered during usage. Design and implementation of a PDP-1 time-sharing monitor (the monitor previously in use was a single-user monitor) and NCP was accomplished by W. R. Conrad, starting in September, 1970. This work is also finished. User manuals for both systems will be available shortly.

Since completion of the hardware interfaces to the IMP at the beginning of 1971, a number of programs have been written using the graphics facilities of the PDP-1 and the computational power of other processors. The first of these was an effort to run an aircraft carrier landing simulation program using the combined resources of the Harvard machines and the Project MAC PDP-10 and LDS-1. The program was edited, assembled, and loaded on the Harvard PDP-10. A core image was then sent through the network to MAC, where it was executed. The LDS-1 was activated and caused to store a computed image. This was then sent to the Harvard PDP-1 for display. The system was made sufficiently operational to demonstrate the bandwidth restrictions imposed by the 50 kilobaud network transmission rate; update rate approximated one frame per second. Bugs in the LDS-1 and the MAC network interface prevented completion of the experiment.

Use of the PDP-1 line printer from the PDP-10 has been a daily occurrence; using the PDP-1 as a high-speed character display terminal is also common. These programs suffer a bandwidth restriction from the gyrations necessary using the (interim) pseudo-teletype mode of communicating with the PDP-10. The bandwidth constraint is around 10 kilobaud; this constraint will be relieved when the programs are rewritten to use the NCP's.

A package has been written to make PDP-1 display file generating subroutines directly callable from FORTRAN on the PDP-1. This has been used to write a number of small applications graphics programs. We hope soon to have (using the new monitor) a multi-user graphics facility, using the displays, tablets, and teletypes on the PDP-1 as terminals to any machine on the ARPA network.

5.2 FILE TRANSFER AND CONVERSION

The IMP-IMP and HOST-HOST protocols for the ARPA Network provide a standardized means, independent of the types of hardware and/or operating systems comprising the HOST systems, for routing and transferring data between HOSTs This means that use of a local network control program to get a connection established and to send data to a distant HOST is done in the same way, independent of the characteristics of the distant HOST. Contrariwise, the HOSTs are oblivious of the logical structure of the data transferred -- data is treated as a stream of bits (or bytes of a specified size). Thus, responsibility for resolving problems arising from differences of data organization and representation at the two HOSTs devolves on the data management and/or user programs running on the two HOSTs.* Current Network Working Group efforts are in part directed toward defining file and data transfer protocols which will further reduce the sensitivity of user programs to differences between HOST systems.

We note at this point that the overall problem is virtually identical to that faced by most computer installations in the mid-sixties during conversion to new systems; the total cost amounted to hundreds of millions of dollars. The net effect of this experience was to lock a sizable percentage of all installations into continued use of essentially their current type of hardware, languages, and mass storage systems -- to mount another large scale conversion effort is, for a great number of installations, literally unthinkable. By the same token, the manufacturer is locked into perpetuation of his current instruction set, except possibly for extensions.

The implications of the above are clear; to the extent we can find ways of transfer and conversion of data which are automatic, rather than requiring changes in user programs to match the distant HOST, we will have contributed

*For the purposes of this discussion, program transfer and storage is considered to be a special case of data transfer and storage.

toward avoidance of the system conversion trauma and expense -- the potential payoff is almost incalculable. This is to say that our stakes are far larger than might be inferred from consideration of our desire to promote use of the ARPA Network by automating the file transfer and conversion process to the extent practicable.

While we cannot pretend to have a "solution" to this set of problems in hand, we have developed a set of insights, to a great extent embodied in the ECL system, which seem to us to be essential ingredients of any partial or complete solutions. These are descirbed in the remainder of this subsection.

5.2.1 PHENOMENA, DATA, AND DATA REPRESENTATIONS

A collection of data is, we contend, a theory concerning some set of phenomena. That is, one posits some set of entities or objects (people, inventory items, or the like) and makes some set of assertions about them ("the name of that person is John Doe, his age is 43, his sex is male, his wife's name is Jane Doe, his license number is 031165810", and so on). Thus, the data can be viewed as a conjunction of predicates of the form "the value of attribute <u>zilch</u> of entity <u>zot</u> is <u>zatch</u>." This theory corresponds, in the case of a punched card representation of the data, to a set of cards for entity <u>zot</u>, the field named <u>zilch</u> having the value <u>zatch</u> punched into it. If the cards are transcribed to tape or are printed, we have another representation of the same data. Thus, we may say that a data representation is any way of recording the data (the theory) for purposes of communication, storage, or processing.

5.2.2 DATA TYPES

Just as we have a dual notion of data (that is, the theoretical object as opposed to a representation of that object), our intuitive notion of data type exhibits a similar duality. At the theoretical level, the data type (for instance, integer, real, list, and so on) corresponds to the behavior of data of that type -- that is, to the allowable processing operations. For example, we can concatenate two lists, but we cannot divide one by another; division is an undefined, or unallowable, operation for data of type list.

On the other hand, a data object (that is, a representation of a data item or collection) also has a data type. This data type has properties in addition to those of the generic data type (the type of the data being represented) which relate to the specific manner of representation -- e.g., representation of an integer as a 32 bit twos complement binary number -- and to the storage organization of the data object, which specifies how it is mapped into storage. In order to maintain this distinction, we shall use the term mode to denote the data type of the data object, and the term data type to denote the generic data type. Evidently, a theoretical data time or collection has one data type, but may have a different mode in each representation as a data object.

5.2.3 DATA DESCRIPTION IN PROGRAMMING LANGUAGES

The distinction between data type and mode is confounded in most programming languages, including FORTRAN, COBOL, ALGOL 60 and PL/I. It is clearly the intent of most language designers to emphasize the notion of data type, rather than that of mode; the object of standardizing a language is, to a large extent, to permit transferability -- that is, to so arrange things that a program may work correctly in an evironment different from that in which it was first written and checked out, using different representations for the data. Ideally, an ALGOL 60 program using the data types int, real, and bool should work correctly in any environment.

On the other hand, we find the notion of mode cropping up in some language designs. This is due to a desire for efficiency of program execution and/or data storage in hardware environments which permit, for instance, several hardware representations of integers (for example, on the IBM System/360). Thus, both PL/I and COBOL allow the programmer to select a representation (e.g., COMPUTATIONAL-2 in COBOL or INTEGER BINARY LENGTH(32) on PL/I).

Finally, we note that, even using languages which allow no choice of mode for a given data type, it is not uncommon to find features of programs which depend on knowledge of the representation which will be used by the programming system (for instance, precision). We can be confident that such programs will not operate correctly in all environments, due to differences in representation of the data types.

What, then, is needed to permit "automatic" conversion of files? To say that we need a complete description of each file is hardly illuminating, for there are cases of what seems to be complete descriptions that are incomplete from our present point of view. An example would be a COBOL description -- it is not enough to know that a data item is COMPUTATIONAL-2, for we need to know the representation used, not merely the name for it used in the Data Division. In addition we know that padding is frequently used in COBOL implementations to ensure that data items fall on appropriate storage boundaries -- complete description must communicate this fact.

Indeed, we cannot specify precisely what we mean by a "complete description" at present. This question is currently under investigation in the context of ECL. In the remainder of this subsection, we sketch what appears to be needed for an ECL input/output facility which uses internal representations in order to illustrate problems additional to those touched upon above.

5.2.4 AN ECL INPUT/OUTPUT FACILITY

ECL has a number of primitive (that is, defined by the system implementation) modes, among which is the mode DDB (Data Definition Block) containing all necessary defining information for data objects of that mode. Among the primitive modes are the basic modes -- defining such modes as BOOL, INT, REAL, CHAR, ATOM, and REF.(ATOM is the mode of an entry in the symbol table, while REF is the mode of a pointer to a data object of any mode). Other modes are definable by the user in terms of the basic modes and three mode constructor functons ROW, STRUCT, and PTR. ROW constructs a data object which is an ordered sequence of data objects of the same mode -- a one diminsional array. STRUCT constructs a data object which is an ordered sequence of data objects of inhomogeneous mode, each of which may be named (as in COBOL or PL/I). PTR constructs a data object which can point to data objects of any one of a number of specified modes (this allows a more efficient representation and processing than use of the mode REF).

Let us consider output of a data object, accomplished by calling a function WRITE(A,P) where A is the name of the data object and P the name of an output port. WRITE has both the location and mode of A available to it. In order to ensure that A is fully described in the output file, we must do something equivalent to writing out the DDB for the mode of A. Each DDB contains a field which contains the external form of a character string which defines the mode (that is, from which a DDB can be constructed by the system), for instance

"STRUCT(A:PTR(INT,REAL),B:ROW(2,INT))"

This defines data objects of class STRUCT; their first component is a pointer to an INT or to a REAL and is named A, and their second component is a ROW of 2 INT's, named B. One could either output the above canonical name for the mode in front of the data object or output each new canonical name as it is encountered and reference it by its serial number in front of each data object of that mode. Of course, any component modes whose definitions had not occurred in the output streams would have to be written out first.

The question of what to do about imbedded pointers is more involved (as in the above example). The question is, how do we tell whether the data object pointed to should be output as well as that containing the pointer? This can be done by using a syntactic device in the canonical name -- for instance, writing PTR("INT","REAL") in the case that the object pointed to should not be output. In the former case, we still have the problem of how to represent the pointers, but this is taken care of by observing that in such cases a sequence of data objects will be output, and a pointer to any one of them may be represented by the serial number in that sequence of the data object pointed to. An inverse conversion must be performed on input, of course.

5.2.5 GENERALIZATION TO DIFFERING ENVIRONMENTS

The above sketch indicates a feasible approach toward input/output, provided that we are dealing only with a single implementation of ECL, such as the current PDP-10 implementation. We now examine the issues raised by a change of environment -- for instance, a file written by a PDP-10 implementation of

ECL and read by a System/360 implementation of ECL. In this case, we first note that use of the mode INT in the canonical names in the file will not work because it is not specific as to representation of integers. However, if all usages were to have been replaced by a more specific mode name on output -- for instance PDP10-INT -- the conversion apparatus in ECL would perform the proper conversion on input, given primitive conversion functions.* Similarly for other basic modes.

This is not enough, however. The ECL system reading the file must also have an indication of the environment in which the file was written at the head of the file. This is necessary, for instance, because compound objects may be preceded by a dope vector whose form is not indicated by a DDB but is implementation-specific.

If we generalize further to files read or written by systems other than ECL, issues escalate again. However, these are questions of standardization rather than the type of issue raised above. We believe that our conceptual arsenal is now adequate for the problem. What remains are the experimental implementations required to sharpen our technique and uncover any remaining conceptual problems.

*See "The Treatment of Data Types in ECL."

BIBLIOGRAPHY

Items in this Bibliography are followed by the number of the section in this report to which they are related.

1. Cohen, Dan, and John M. McQuillan, "Computer Graphics for System Applications," 1971 Hawaii International Conference on System Sciences, (III).

2. Cohen, Dan, and John M. McQuillan, "Computer Graphics for Transportation Problems," 1971 Spring Joint Computer Conference, (III).

3. Corey, E. J., and W. T. Wipke, Science 166, 178, (1969), (III).

4. Corey, E. J., W. T. Wipke, R. D. Cramer, and W. J. Howe, Journal of American Chemical Society 94, (1972), (III).

5. Corey, E. J., R. D. Cramer, and W. J. Howe, Journal of American Chemical Society 94, (1972), (III).

6. Corey, E. J., and G. A. Petersson, Journal of American Chemical Society 94, (1972), (III).

7. Land, R. I., "Computer Art: Color-Stereo Displays," Leonardo, Vol. 2, (1969) pp. 335-344, (III).

8. Land, R. I., and I. E. Sutherland, "Real Time, Color, Stereo, Computer Displays," Computer Applied Optics, Vol. 8, No. 3 (March, 1969) pp. 721 ff.(III)

9. Wegbreit, B., "The ECL Programming System," Proceedings FJCC, Vol. 39 (1971) pp. 253-262, (IV)

10. Wegvreit, B., "A Generalized Compactifying Garbage Collector," to appear in The Computer Journal, August, 1972, (IV).

11. Wegbreit, B., "The Treatment of Data Types in ECL," Technical Report, Center for Research in Computer Technology, Harvard University, August, 1971, (IV).