A SMALL SCALE REAL TIME COMPUTER GRAPHICS SYSTEM

Thesis for the Degree of M. S.
MICHIGAN STATE UNIVERSITY
CHRISTOPHER SCUSSEL
1976

ABSTRACT

A SMALL SCALE REAL TIME COMPUTER GRAPHICS SYSTEM

by

Christopher Scussel

With the increasing popularity of graphic displays as a medium for computer interaction, and with the advent of the relatively inexpensive small scale computer, a need for an inexpensive add-on real time graphics system has developed. A review of current display technology indicates a lack of availability of such a system.

This paper describes a system consisting of an oscilloscope, an analog vector generator, IBM 1800 computer, and a set of FORTRAN compatible user level subroutines. The system is capable of displaying up to 200 vectors, representing several three dimensional wire frame objects, and can translate, rotate, deform, and project them in perspective in real time. The system is compatible with most modern minicomputers, and if an oscilloscope is already available, the total additional cost of hardware for the system is about thirty dollars.

A SMALL SCALE REAL TIME

COMPUTER GRAPHICS SYSTEM

by

Christopher Scussel

A THESIS

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

MASTER OF SCIENCE

Department of Computer Science

1976

To My Wife and Parents

ACKNOWLEDGMENTS

I would like to thank my advisor, Professor J. Forsyth, for his time, effort, and very helpful organizational suggestions during the preparation of the several drafts of this manuscript. Also, I would like to thank the other members of my examination committee, Professors R. Reid, J. Burnett, and F. LeCureux, for their time, consideration, and suggestions. And finally, I would like to thank Gary Bergeron, senior in electrical engineering, for constructing the prototype vector generator.

TABLE OF CONTENTS

Page

LIST OF FIGURES

INTRODUCTION

With the increasing popularity of graphic displays as a medium for computer interaction, and with the advent of the relatively inexpensive small scale computer, a need for an inexpensive add-on real time graphics system has developed. A review of current display technology indicates a lack of availability of such systems. The purpose of this paper is to describe an inexpensive stand-alone graphics system, designed around a typical minicomputer. The system consists of a display processor, display software, a hardware vector generator, and a display device. To help attain the cost objective, the display processor is actually the minicomputer, which shares its time between display processing, display refreshing, and normal (non-display) processing. The display software allows apparently continuous transformations on up to two hundred vectors (typically) in three space, including rotation and perspective, without requiring floating point hardware, thus attaining the real time objective. The vector generator is a small device consisting of operational amplifiers and solid state switches, and the display device is an unmodified laboratory oscilloscope. Since oscilloscopes are generally available at minicomputer installations, the additional hardware cost of the system is quite low.

The system is designed to provide the minicomputer user with small scale real time interactive graphical capability, through the use of a set of FORTRAN-compatible subroutines for generating and manipulating wire-frame representations of objects. A prototype of this system has been assembled, using an IBM 1800 processor-controller as the minicomputer, a low performance vector generator (total component cost about thirty dollars), an object oriented software system, and a laboratory oscilloscope associated with the IBM 1800 installation. This system, hereafter referred to as "the prototype system," is capable of displaying multiple wire-frame objects undergoing distinct rotational, translational, deformational, and perspective transformations, in real time.

This paper is divided into three sections: a review of current real time display technology, a description of the prototype system, and an explanation of the prototype system software.

1. Current Real Time Display Technology

Even though real time graphics systems range from rudimentary point drawing systems to extremely expensive systems capable of generating images of shaded polyhedral objects, all graphics systems can be broken down into three basic parts: the display device, the display generator, and the display controller. A small scale system designed for real time operation cannot currently hope to achieve a shaded image, so the best that can be hoped for is line drawing capability. Thus, the display generator is actually a line, or vector, generator.

These three divisions are discussed below, and the best candidates for a small scale real time graphics system are selected.

1.1. Display Devices

The cathode ray tube has been the traditional choice for electrovisual systems for many years, and there are a number of ways in which it can be used in a graphics system. Also, new forms of display devices are beginning to emerge from the laboratory, such as plasma panels and liquid crystal displays. Since the choice of display device is crucial to the design of the rest of the system, it is logical to begin with an examination of the devices available.

1.1.1. Systems Utilizing Cathode Ray Tube Displays

1.1.1.1. Video Based Systems

Video devices utilize a raster scanned cathode ray tube, similar to a television monitor. Some systems are designed to take advantage of this similarity by using standard television format video signals to drive the display. This enables such systems to use a television monitor as a display device, making large screen and multiple displays readily available, as well as inexpensive.

The raster scan technique continually retraces the screen in order to maintain the image, and so the data comprising the image must be constantly resupplied to the display. This may demand excessive data rates from the computer. As an example, consider a display with about 500 scan lines and as many points per line, where each point may be intensified or unintensified, and suppose thirty frames per second are displayed. This is similar to an ordinary television monitor, except that such a monitor has many more than just two intensity levels. The specified resolution generates about 250,000 points for which intensity information must be sent to the display thirty times per second. Even though this information amounts to only one bit per point, the required data rate is in excess of 7,000,000 bits per second, which is too fast for the computer to provide directly unless it is doing little more than refreshing the display.

Because of the excessive demand upon the computer if it is driving the display directly, a buffer memory is generally inserted between the two. This shifts the high data rate to being between the display and the buffer, and allows the computer to spend most of its time performing display computations instead of refreshing the display. Also, the computer need transmit to the buffer only those points which have changed state (dark to light or vice versa) since the last frame. This results in a considerable decrease in the amount of data transmitted, especially in the case of line drawing or wire frame displays.

The buffer memory is comprised of integrated circuits, either random access memories or shift registers. While shift registers are inexpensive and require very simple control circuitry, the average time for access to a given point is one half of the time required to refresh the display. In the example given above, this is 0.0167 seconds, which is far too great to be tolerated. Thus, shift register buffer memory is generally limited to applications in which the display changes relatively infrequently, and so the computer can afford to access all the points of the screen sequentially in one frame.

Random access memories completely circumvent this problem, although with increased cost and controller complexity (in particular, the computer can be allowed memory access only between accesses by the display). Random access buffer memories are utilized in several display systems designed specifically for use with minicomputers having a sixteen bit word size and video monitors.[R1, R2] These devices provide 256 by 256 resolution, yielding a one to one correspondence between addressable display points and all possible sixteen bit words. Such a system is not suitable for real time applications, as the computer must generate lines by accessing each point of the line, and so each line may require on the order of a millisecond to be formed.

1.1.1.2. Oscilloscope Based Systems

Recently a number of display interfaces have been marketed for use with a laboratory oscilloscope as the actual display device. The simplest of these interfaces is a point plotter, which is supplied with points, in the form of digital coordinate pairs, by the computer.[R3] After entering a coordinate pair into its single point buffer, the coordinates are converted into voltages by a pair of digital to analog converters and the beam of the oscilloscope is deflected to the desired point, and that point is illuminated. The computer must draw lines as a series of points, and must periodically redraw the image if it is to remain on the screen.

More sophisticated devices combine point plotting capability with a refresh buffer and a vector generator, so that the oscilloscope display may be maintained without the attention of the computer.[R4] The vector generator vastly lowers the required computer-to-display data rate and the refresh memory size, if the images to be produced consist mainly of line segments. Generally, the refresh memory cannot be accessed by the computer, and thus the refresh memory must be completely cleared and reloaded in order to make the slightest change in the display (short of the addition of new line segments). This is a property common to most display devices which maintain refresh memory external to the computer serviced by the display. The concept of "shared memory" for the computer-display interface has evolved to avoid this difficulty (see section 2.3).

1.1.1.3. High Performance Hardware Systems

There are a number of commercially available graphics systems which rely on fast special-purpose hardware to maintain and transform the display on a cathode ray tube.[R5, R6] Such hardware usually includes circuitry for rapidly performing matrix multiplication, in order to facilitate translation, rotation, and perspective transformations. At least one system contains not only this but also hardware for hidden surface elimination and smooth shading of polyhedral objects.[R8] This system is fast enough that objects consisting of several hundred triangular faces may be displayed with shading and transformed in real time. Unfortunately, this capability is prohibitively expensive to all but the largest computer graphics laboratories.

1.1.2. Systems Not Utilizing Cathode Ray Tube Displays

The cathode ray tube is far from ideal as an electrovisual interface. It is bulky, requires unwieldy voltages to accelerate and deflect its electron beam, can smear moving images through phosphor persistence, may be damaged by overly bright images, and is potentially dangerous because of the risk of implosion. Several alternative devices have been proposed which avoid some of these problems: the plasma panel, light emitting diode matrices, and liquid crystal displays. Of these devices only the plasma panel is currently commercially available, while the other two are still undergoing development.

1.1.2.1. Plasma Panel Displays

The plasma panel is a flat, rectangular enclosure containing low pressure gas, which may be ionized by applying a voltage across electrodes on opposite sides of the device. The electrodes are arranged vertically on one side, and horizontally on the other, so that the gas may be made to ionize, and thus glow, in a small, discrete point, by energizing one vertical and one horizontal electrode. By exploiting the difference between the sustaining and extinguishing voltages of the gas, the panel can provide its own memory, and thus maintain an image with neither computer intervention nor external memory hardware. This does not provide for gray scale, although it has been shown that several stable ionized states of the gas may be used to provide different intensities, but this is still experimental.[B1] The plasma panel display suffers from the same data manipulation problems as all the non-vector displays, namely, that a large amount of information must be passed to the display in order to move some part of the image. This can become a serious problem in real time situations.

Plasma panel displays have just recently become commercially available for use with small computers, although their resolution (about sixty points per inch) and size (about nine inches square) somewhat limits their usefulness.[R7, R9]

1.1.2.2. Light Emitting Diode Matrices

These displays are constructed from subarrays of integrated light emitting diodes. Each individual diode is addressable in a manner similar to that of a word in a plane of random access memory: by selecting the row and column containing the desired diode. Light emitting diodes respond to changes in current very rapidly (within a few nanoseconds), and so smearing is not a problem. Since they have no inherent memory the display must be continually refreshed by the computer or external circuitry. Even so, this task is less difficult than with cathode ray tube systems, because light emitting diodes are low voltage devices and thus do not require high voltage drive circuitry.

Currently the cost of this type of display is very high, even though its resolution is poor (at best about fifty points per inch). This situation will undoubtedly improve as integrated circuit technology continues to advance.

1.1.2.3. Liquid Crystal Displays

Liquid crystal displays are based upon the fact that certain chemicals change their light scattering properties in response to an electric field. A flat, rectangular cell with electrodes on the front and back (similar to the plasma panel) containing liquid crystal material can be made to selectively scatter or transmit incident light at each of the cells formed by the matrix of electrodes. Note that the display does not generate any light of its own; thus, constrast is improved, rather than degraded, by high ambient light conditions. By viewing scattered light, a conventional display is presented. By viewing transmitted light, the display becomes projecting, and thus may generate very large images.

Liquid crystal displays are capable of inherent memory, and share most of the data rate problems of point addressable displays. In addition, they are quite slow to respond to changes in the electric field, and thus smear can be a severe problem for rapidly moving images. Resolution is approaching one hundred points per inch, but cost reliability are still major problems. Further research should eliminate most of these difficulties.

1.1.3. Conclusions

Cathode ray tube displays are the best selection for a small scale graphics system for two reasons. First, their technology is well developed, allowing relatively low cost through mass production and proven driving circuitry design. Second, it can be used to directly display lines, as opposed to the quantization inherent in raster and matrix displays. This is a result of the fact the cathode ray tube is an analog device, and thus not limited to a fixed set of digital inputs.

The benefits of both advantages may be had by using an oscilloscope as the display device. Aside from its self-contained power supplies and deflection amplifier circuitry, the oscilloscope often has the additional feature of being readily available at a minicomputer installation, perhaps in connection with maintainance of the computer and its peripheral equipment.

Since the oscilloscope may be used in several ways to generate images, it must be decided which technique is appropriate for the desired graphics system. In the small scale system considered here, the images are to consist entirely of lines (or vectors), so that the image generation problem reduces to one of vector generation.

1.2. Vector Generation

Because many real time computer graphics applications require lines to be drawn as part of the completed image, it is worthwhile to have vector generation capability in the display hardware. Points can then be generated by treating them as zero length vectors, and thus a vector generator is sufficient to generate most desired images. The vector generator has no processing power, and simply draws a line segment between two specified endpoints. Also, the vector generator has no refresh memory, and thus the task of refreshing the display is left to the display controller.

In an oscilloscope based graphics system, vectors may be generated by either digital or analog circuitry. Analog circuitry has the advantage that it can be connected directly to the oscilloscope, whereas digital circuitry requires digital to analog converters in order to drive the oscilloscope. Digital techniques will be treated first, then analog techniques.

1.2.1. Digital Vector Generation Techniques

Digital techniques generate a vector as a sequence of points, and thus if the vectors comprising the image are to be drawn fast enough to avoid flicker, the hardware must generate the points very rapidly indeed. For example, in a system designed to display up to two hundred vectors with an average of two hundred points each, and refresh the display thirty time per second, points must be generated at a rate of more than 1,000,000 per second. While this is not excessive, it does require careful circuit design and a simple sequential algorithm in order to be practical.

Three digital techniques are treated here: binary rate multipliers, digital differential analyzers, and integer-scaled proportioning.

1.2.1.1. Binary Rate Multipliers[B1]

An n-bit binary rate multiplier accepts as input an n-bit number m and a sequence of clock pulses, and during its 2^n clock pulse period generates m more or less regularly spaced pulses. Thus, by using two binary rate multipliers with counters to accumulate their output pulses, any given vector can be approximated. The algorithm and circuitry involved for the binary rate multiplier is quite simple, and in fact high speed implementations are available as a single integrated circuit. Its major disadvantage is that the m output pulses are not sufficiently equally spaced in time to draw straight vectors, resulting in lines which vary in intensity and direction along their length. Note that since each circuit will go through one complete 2^n clock pulse cycle for each vector drawn, each vector requires 2^n clock pulses to be generated, regardless of its length and direction. While this tends to simplify the display controller, it lowers the number of vectors which may be displayed without flicker.

1.2.1.2. Digital Differential Analyzers[B1]

The digital differential analyzer type of vector generator seeks to determine the set of points out of which to form the desired line segment by approximating the solution to the differential equation which determines the line. This differential equation is

$$\frac{dY}{dX} = \frac{\Delta Y}{\Delta X}$$

where ΔX and ΔY are the X and Y coordinate differences between the two endpoints of the line. There are various forms of digital differential analyzer algorithms, all of which are incremental in nature. That is, the next step in the discrete solution is found by incrementing either the X or Y coordinate (or both) of the previous step. This means that a new point is generated for each clock pulse, and so the desired vector may be generated much more quickly than with the binary rate multiplier form of vector generator. However, all forms of the digital differential analyzer require division to determine one or more of the parameters involved in the algorithm. This division can be performed in the vector generator, but this adds considerably to the cost of the circuitry. On the other hand, if the division is performed by the computer, it is time consuming, and thus reduces the number of vectors which can be displayed during one refresh cycle. The division required by this type of vector generator is its main fault. In some algorithms, the parameters are set up so that the division operation becomes a matter of shifting, after forcing the divisor (an estimate of vector length) to be a power of two. While this avoids the division, it causes some points to be displayed more than once and displays some points which should not be displayed.

1.2.1.3. Integer-Scaled Proportioning

This method takes advantage of the fact that since the slope of any vector to be displayed is rational, the parameters of the vector can be scaled up by an integer (namely, the product of the X and Y differences of the vector endpoints) in order to make all of the parameters integral. Explicit multiplication is not necessary, and so this method does not succumb to arithmetic difficulties, as does the digital differential analyzer. Since this type of vector generator is not treated in (B1), it will be presented here in somewhat more detail than the other two types of digital vector generators.

This method is similar to the binary rate multiplier technique, in that it generates pulses which are fed to counters, and the counters keep track of the coordinates of the point which is to be displayed. The method is iterative, and either one or both of the counters is incremented during each iteration. A pair of registers keeps track of which of the counters is to be incremented. These registers maintain a scaled version of the progress made in drawing the line segment, with the scaling such that in order for the generated line segment to be as close as possible to the desired line segment, the two registers should be as close to being equal as possible. The desired scaling results if, during each iteration, the counter corresponding to the smaller of the two registers is incremented, and that register is increased by the component of the desired line segment along the other coordinate. For example, if the X register is smaller, the X counter is incremented, and the X register is increased by the difference of the Y coordinates of the endpoints of the desired line segments. If the two registers are equal, then both counters are incremented and both registers are increased.

The algorithm is defined in FØRTRAN below, where X and Y are the counters and XC and YC are the registers. Notice that the coordinate differences DX1 and DY1 are actually the differences plus one, and the registers are initially these differences rather than zero. This causes the algorithm to work for horizonatal and vertical line segments, without additional logic.

```
      SUBROUTINE DRAW(XA,YA,XB,YB)
      IMPLICIT INTEGER (A-Z)
      DX1 = XB-XA + 1
      DY1 = YB-YA + 1
      XC = DY1
      YC = DX1
      X = XA
      Y = YA
      CALL PLOT(X,Y)
    1 IF(X.EQ.XB .AND. Y.EQ.YB) RETURN
      IF(XC-YC)2,3,4
    2 X = X + 1
      XC = XC + DY1
      CALL PLOT(X,Y)
      GO TO 1
    3 X = X + 1
      Y = Y + 1
      XC = XC + DY1
      YC = YC + DX1
      CALL PLOT(X,Y)
      GO TO 1
    4 Y = Y + 1
      YC = YC + DX1
      CALL PLOT(X,Y)
      GO TO 1
      END
```

In the above subroutine, (XA,YA) and (XB,YB) are the endpoints of the vector to be drawn, with XB≥XA and YB≥YA, and PLOT intensifies the given point on the screen. By keeping the quantities X and Y in up/down counters, the hardware can easily handle those cases where XB<XA or YB<YA. Note that the algorithm involves no multiplications or divisions, and generates a new point during each iteration. It can be implemented inexpensively with integrated circuitry, and is quite fast. An additional feature of this particular method is that the lines which are generated are symmetric from endpoint to endpoint and balanced around the line being approximated. This means that, aside from

generating visually pleasing lines, no extra points will be intensified if a line is retraced opposite its original direction.

1.2.1.4. Conclusions

Any of the three types of digital vector generators presented here are suitable for a low cost graphics system. However, the integer-scaled proportioning technique yields the best lines, far better than those of the binary rate multiplier technique, and completely avoids the divisions of the digital differential analyzer methods. In addition, the proportioning technique can elegantly handle any combination of vector endpoints, whereas some of the digital differential analyzer methods must split the possibilities into several cases, which may complicate the controller. Thus, the integer-scaled proportioning technique is probable the best suited for a digital vector generator in a small scale graphics system.

1.2.2. Analog Vector Generation Techniques

In the past analog generators have been difficult to work with and expensive, since their proper and accurate operation depended upon careful adjustment of complex circuitry. More recently, improvements in integrated and hybrid circuit technology have considerably reduced both the complexity and sensitivity of analog circuitry, and so analog vector generators have become more practical. This is fortunate, because analog vector generators are inherently much faster than digital ones, and can require less control circuitry. And, of course, analog generators actually draw lines, as opposed to a series of points.

There are many forms of analog vector generations, most of which fall into three basic classes: exponential, interpolation, and integration.

1.2.2.1. Exponential Techniques[B1]

The exponential methods are based on the fact that the charging curves of resistor-capacitor (RC) circuits are all scale models of each other. Thus, if simultaneous (but possibly unequal in magnitude) voltage steps are input to identical RC circuits (initially in equilibrium), the output voltages will be such that a line segment will result, if they are plotted parametrically. If the input voltage function is

$$V_{in} = \begin{cases} V_a & \text{for } t < 0 \\ V_b & \text{for } t \geq 0 \end{cases}$$

then the output voltage function of an RC circuit with this input is

$$V_{out} = \begin{cases} V_a & \text{for } t<0 \\ V_a + (V_b - V_a)(1 - \exp(-t/RC)) & \text{for } t \geq 0 \end{cases}.$$

Figure 1 shows this relationship.

This generator suffers from two basic problems: the generated line segment does not attain its endpoint, and the intensity of each vector varies along its length. As can be seen from the output voltage function above, the "final" voltage value V_b is never achieved, only approached as t approaches infinity. Thus, a vector will be 99 percent complete when t is 4.61RC, but will still never attain its endpoint. This problem can be somewhat alleviated by reducing the resistances as each vector nears its endpoint, and so increasing the rate at which the endpoint is approached. This does not result in attainment of the endpoint in a finite time unless the resistance values are decreased to zero, but if the values become reasonably small, the endpoint is attained, practically speaking, within an acceptably short period of time. Correcting the varying intensity requires changing the beam current to compensate for the changing beam velocity across the screen. Since the function relating beam current and screen intensity is highly nonlinear, adequate compensation is difficult to achieve. One advantage of the technique is its freedom

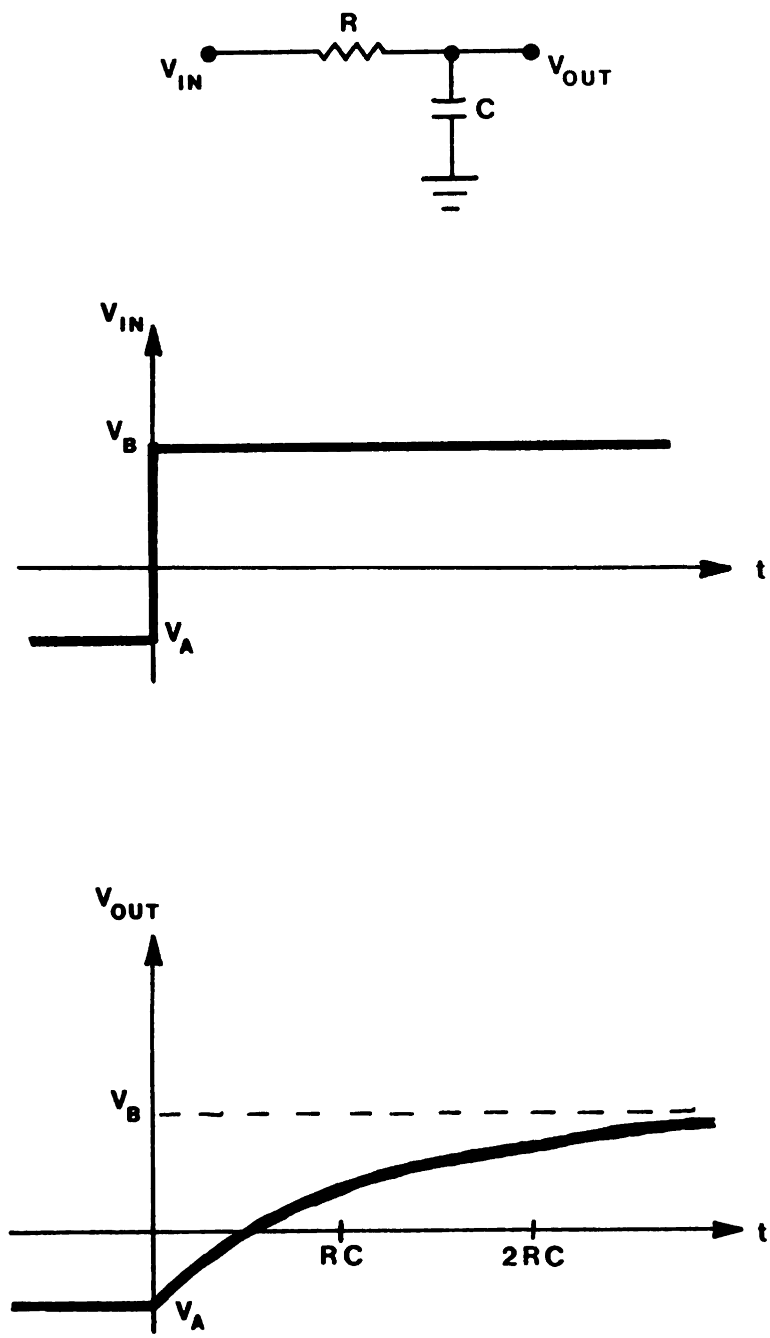


FIGURE 1. RC CIRCUIT AND WAVEFORMS

from drift; the relative error of the vector endpoints does not increase between the first and last vectors drawn during each frame.

1.2.2.2. Interpolation Techniques[B1]

Interpolation schemes form a line segment by taking a weighted average of the endpoints of the segment, as the weights vary linearly between zero and one:

$$V_X = X_a + W(X_b - X_a)$$
$$V_Y = Y_a + W(Y_b - Y_a) \quad .$$

The particular form of the equations above demonstrates why this is known as an interpolation technique. The basic problem with this type of technique is keeping the sum of the weights unity, or, in the equations above, accurately performing the multiplication. This difficulty may be circumvented by designing the vector generator around a modified multiplying digital to analog converter. However, the circuitry involved in this approach is somewhat more complicated and critical than that of ordinary digital to analog converters.

Advantages of this technique are lack of drift and uniform intensity along the length of each vector.

1.2.2.3. Integration Techniques[B1]

Integration techniques involve the use of operational amplifiers with capacitive feedback in order to generate ramps with slopes proportional to the input voltages. When the output voltages are plotted against each other on the display screen, a straight line results. The orientation and length of the line can be controlled by varying the integrator time constants, the input voltages, or the integration times.

An integrator with variable time constant requires an electronically variable resistor or capacitor, which are generally difficult to control with the necessary accuracy. Also, changing the time constants of the integrators changes the rate at which the beam is swept across the screen, and thus the intensity of the generated vectors will vary with their length and orientation.

All vectors may be made to be of equal intensity, without resorting to modulation of beam current, if the beam sweep speed is held constant. Integration time is then proportional to vector length, and the integrator input voltages are proportional to the components of the unit vector in the direction of the desired vector. Notice that this essentially requires the polar form of the vector, and that the integration time be variable, but highly accurate (on the order of one microsecond). While the required timing can be performed in the vector generator hardware without difficulty, the polar coordinate conversion is difficult and potentially time consuming whether done in the vector generator, display controller, or the computer itself.

Leaving the time constants and integration time unchanged and modifying only the input voltages is easily done, and has several advantages over these other techniques. The ideal integrator transfer function is

$$V_{out} = -1/(RC) \int V_{in}\, dt$$

and thus if the initial point of a vector is (X_a, Y_a) and the final point is (X_b, Y_b) then the integrator inputs are proportional to $X_b - X_a$ and $Y_b - Y_a$; further, the integrator outputs are initially X_a and Y_a, and change linearly to X_b and Y_b. This is illustrated in Figure 2. Since the differences $X_b - X_a$ and $Y_b - Y_a$ can be positive or negative, bipolar voltages must be applied to the integrator inputs. The integration time is constant, so each vector requires the same amount of time to be drawn, and if the beam current is constant, long vectors will be dimmer than short

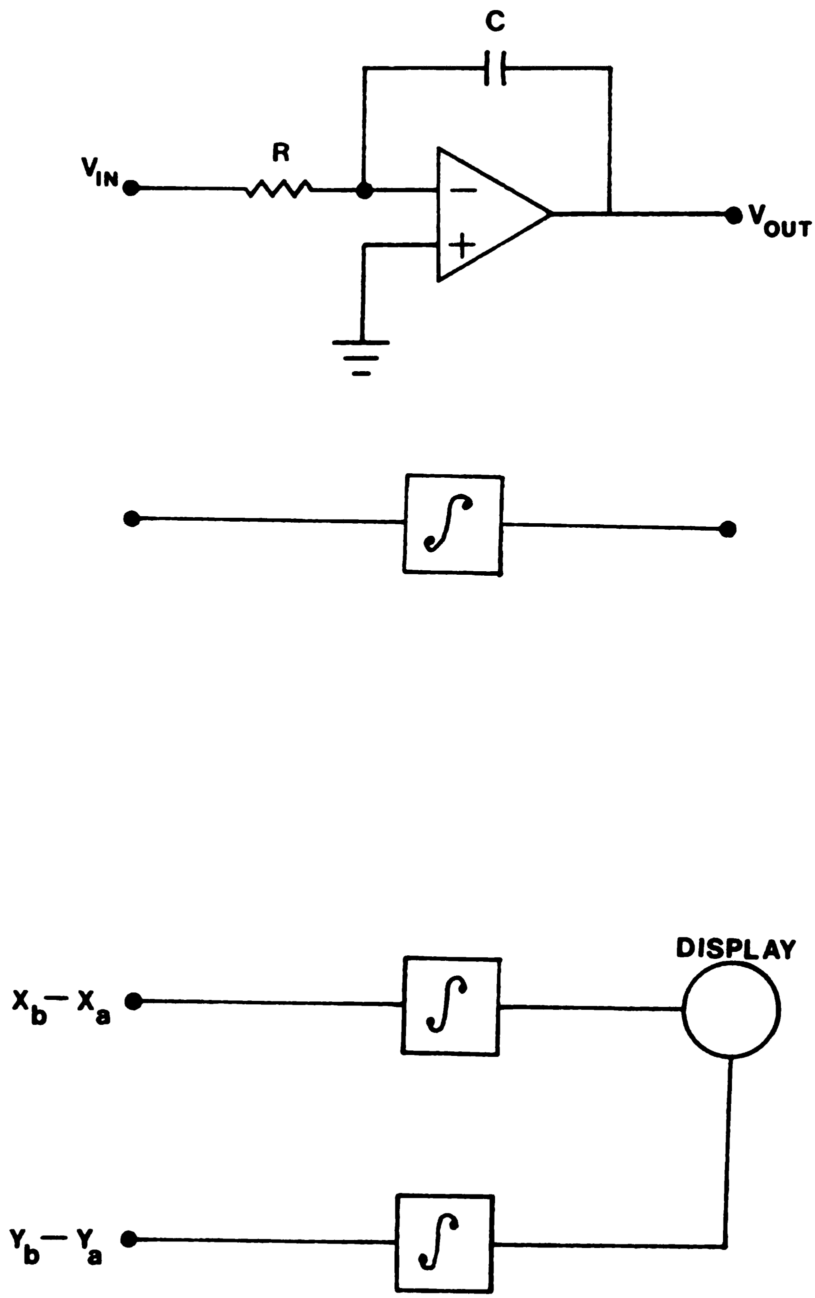


FIGURE 2. INTEGRATOR VECTOR GENERATOR

vectors. Even so, since each vector is traced at a constant speed, it will be of constant intensity along its length.

The main problem with the integrator type of analog vector generator is integrator inaccuracy. This stems mostly from feedback capacitor loading and leakage, and operational amplifier output offset. Capacitor loading is caused by the finite input impedance of the operational amplifier draining the charge of the capacitor, and capacitor leakage is due to the imperfect capacitor dielectric. Operational amplifier output offset is due to electrical imbalance within the amplifier, and causes the output to be at a nonzero voltage, even if the amplifier inputs are grounded. The effect of capacitor loading and leakage is to displace the vector endpoints and distort the vectors, so that they appear to bend, in severe cases. These effects tend to be about equally bad for each vector in a given image. However, output offset is constantly integrated by the integrators, and always has the same sign, and thus accumulates, making each successive vector endpoint more in error than the previous one. Feedback techniques have been developed which minimize these problems.[B1]

1.2.2.4. Conclusions

The integration type vector generator is the best suited analog technique for three reasons. First, unlike the exponential method, the vectors it generates have uniform intensity along their lengths. This is especially useful with three dimensional displays, since false depth cues are created by vectors which grow dim at one end. Second, both the generator and controller circuitry are simple, compared to that required for the interpolation techniques. And third, the inaccuracies of the integration method are readily avoided for the short periods of time involved in generating a small scale display.

1.2.3. Conclusions

Analog vector generator techniques, particularly the integration method mentioned above, can be satisfactorily implemented in a small scale graphics system at a lower cost than comparable digital generators. This is due to the comparative complexity of the controller required in the digital systems.

Also, a low cost vector generator will tend to have poor resolution, since lowering resolution is one way to lower cost. In an analog vector generator, this affects only the possible positions of the vector endpoints; the vectors themselves will still be line segments. In a digital vector generator, lower resolution implies coarser, less pleasing discrete approximations to the desired vectors.

1.3. Display Controller

It is the responsibility of the display controller to direct the display generation hardware, in order to provide the desired images. The controller is not always identifiable in a graphics system, since it may be distributed in various other parts of the system.

1.3.1 Display Controller Separate From Computer

The key feature of a display controller which is separate from the computer is that the display can be maintained on the screen without the intervention of the computer. If the controller is sufficiently powerful, a dynamic image, with rotating objects, for example, can be displayed without the attention of the computer. For a small scale system this is clearly impractical, and so the controller is limited to the ability to maintain only static displays.

If the controller is to maintain an image without computer intervention, then it must have some type of memory, either of its own or shared with the computer. If the controller has its own memory, then the computer must have some access to it, possibly via the controller itself, in order to modify the display. If the computer shares its own memory with the controller, then the computer can expeditiously change the "controller memory" in any desired manner.[B3, B4]

1.3.2. Computer As Display Controller (Software Display Controller)

If the computer is fast enough and its software efficiently coded, it is possible to have the computer directly control the display generation circuitry. This saves the display system the extra hardware and expense of a separate controller, and gives the system some extra versatility, in that the display controller is actually software, and can be recoded to serve specific needs. However, the computer must spend some of its time maintaining the display, even if it is not changing, and this essentially lowers the computer power available to the process (such as a simulation) which is being displayed by the system. As the complexity of the image displayed rises, a greater proportion of time is spent refreshing the display, until the display begins to flicker or there is no compute time left for the process being displayed.

1.3.3. Conclusions

A software display controller is preferable in a small scale system. Such a controller involves no extra hardware cost, and can be modified easily so that it performs efficiently with the type of problem being investigated with the system. Although this type of

to

sc

be

1

g

t

a

of controller can cause available processing power to decrease, this is not a serious problem in a small scale system, since overly complex images are not being attempted.

1.4. Conclusions

These considerations lead to the following graphics system: display controller incorporated into the computer, integrator type analog vector generator, and an oscilloscope as the display device. Since a suitable oscilloscope is generally associated with a small computer installation, and an integrator vector generator is fairly simple, such a system meets the low cost objective.

2. The Prototype System

The prototype system is a small scale real time computer graphics system, composed of a laboratory oscilloscope, an inexpensive analog vector generator, a software display controller resident in an IBM 1800 computer, and FORTRAN compatible user level routines for the IBM 1800. The first three of these components are discussed below, and the last is discussed in the third chapter.

2.1. Display Device

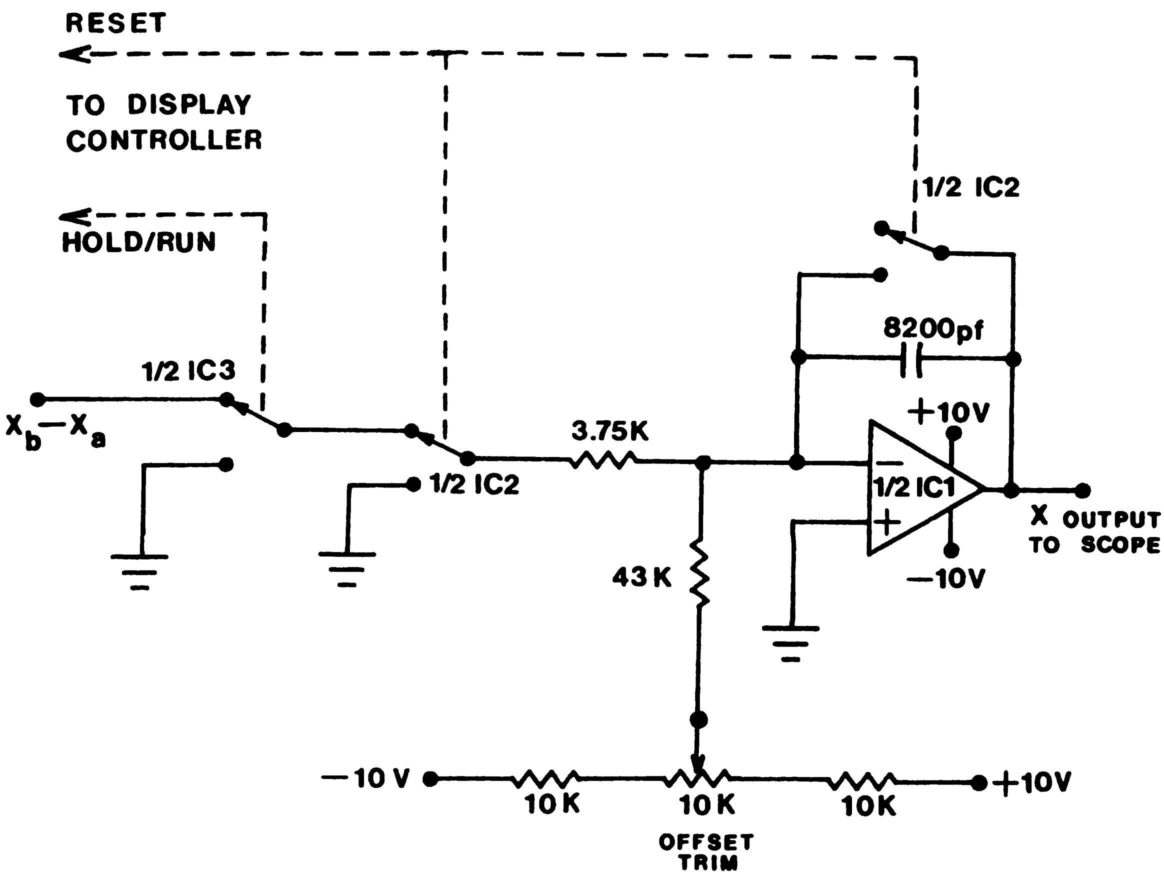
An oscilloscope was chosen as the display device because a small computer installation will often have one already available, thus avoiding the costs involved in purchasing some other type of display. Also, nearly any modern oscilloscope will suffice, since the requirements of the system are quite moderate. If the system is to display two hundred vectors in one sixtieth of a second (thus refreshing at a rate of thirty Hertz with 50 percent duty cycle), only 12,000 vectors per second are being displayed. An oscilloscope with a one megahertz bandwidth can easily reproduce the waveforms required for such a display.

The small size of the oscilloscope screen is potentially a disadvantage to a graphics system, since the details of a complex image may be obscured. However, this is unlikely to happen in a small scale graphics system, since the displays cannot become excessively complicated. Also, the beam can be deflected far off the visible part of the screen with no ill effect other than loss of time, which cannot be serious because of the image simplicity. This relieves the system of the burden of clipping the display, and leads to a substantial reduction in display computation time.

2.2 Vector Generator

The prototype system vector generator is of the analog integrator type. The problems inherent in this type of vector generator have been overcome to the point that performance is quite good, and yet cost is held to a minimum.

A circuit diagram of the vector generator is shown in Figure 3. Polystyrene dielectric capacitors and field-effect transistor input operational amplifiers are used to minimize capacitor leakage and loading. Output offset effects are neutralized during each refresh cycle by a trimming potentiometer connected to the integrator summing junctions, which is adjusted by hand while watching the display for best results. This simple adjustment need be performed only rarely.



IC1--Motorola MC458CP

IC2,3--National LH0014

X CIRCUIT (Y CIRCUIT IDENTICAL)

FIGURE 3. PROTOTYPE SYSTEM VECTOR GENERATOR

Output offset if prevented from accumulating by the solid state "RESET" switch, which resets the integrators to zero volts output between refresh cycles, under direction of the display controller. Since each refresh cycle starts from this reset state, the image is quite stable on the screen.

One of the most crucial parts of the vector generator design is the solid state switch on the input of each integrator. This is the "HOLD/RUN" switch, which grounds the integrator inputs between the generation of successive vectors. During this time, the digital to analog converters which provide the voltages for the integrator inputs are receiving new digital values from the display controller. The individual bit switches in the digital to analog converters do not all respond in the same length of time, so the analog outputs make several transitions, instead of a single one, as the new value is converted. This results in impulse-like voltage spikes ("glitches"), which when integrated displace the endpoints of consecutive vectors. This cannot be tolerated, so the display controller switches the integrator inputs from the converter outputs to ground, whenever a new value is being converted. While eliminating the glitch problem, this technique causes the endpoints of each vector to be somewhat brighter than the vector itself, since the beam dwells there before beginning the next vector. Generally this effect is not displeasing.

While on the subject of vector intensity, it should be noted that although vector intensity is inversely related to vector length, the oscilloscope screen phosphor quickly saturates on the moderately short vectors, thus preventing too wide a range of vector intensities.

2.3. Display Controller

The prototype system display controller is actually a refresh program in the IBM 1800 computer. The user level routines maintain a list of X and Y screen coordinate differences and beam controls, called the refresh buffer, and periodically this list is traced by the refresh program and transmitted to the vector generator. A refresh cycle is begun whenever a timer interrupts the computer and forces execution of the refresh program. As a refresh cycle begins, the program turns the "RESET" control line off, enabling the vector generator. It then begins sending pairs of X and Y screen coordinate differences to the digital to analog converters, while sending the beam control to the oscilloscope. While each pair of coordinates is being converted, the program sets the "HOLD/RUN" line to "HOLD", to avoid integrating converter glitches, and then to "RUN" in order to draw the vector. At the end of the refresh cycle, the beam control is turned off, and the "RESET" line is turned on. Then, the program sets the timer, and returns to the program which was interrupted by the start of the refresh cycle.

Thus the time between the end of one refresh cycle and the beginning of the next is constant. This results in a lower refresh rate for complicated displays, but prevents the system from stalling by spending all of its time on display refreshing. If the lower limit of refresh rate is R Hertz, and each vector requires V seconds to be displayed, then the constant time above is 1/(2R), and the number of vectors which can be displayed 1/(2RV), assuming the computer spends half its time refreshing the display, at most. For the prototype system, V is about 200 microseconds, and so about 167 vectors can be refreshed at fifteen Hertz, while 50 percent of the processing power of the computer is available for other computations. The IBM 1800 used in the prototype system is quite slow, with a four microsecond cycle time. Modern minicomputers are at least four times faster, and so a corresponding performance improvement should be noted if such a machine were used.

2.4. Summary

The combination of oscilloscope, integrator type analog vector generator, and software display controller provides for the display of 167 vectors fifteen times per second while consuming only half of the available processing time, even on a slow computer. This is accomplished with a total hardware cost of about thirty dollars, assuming the availability of an oscilloscope.

3. The Prototype Software System

A set of FORTRAN compatible subroutines acts as the interface between the prototype system and the user of the system. These routines are object oriented, that is, they are designed to facilitate the generation and manipulation of wire frame skeleton objects in 3-space. A set of objects is defined by the user as a collection of point sets, along with a set of vectors which interconnect these points. The software maintains a list of the points, and the object to which they belong, and a list of vectors which constitute a path for drawing the set of objects (note that some of the vectors are dark vectors, representing a move as opposed to a draw). It also maintains a table of object transformation parameters, specifying X, Y, and Z axis translations and rotations. Whenever the user calls for a display update, each point is transformed according to the parameters specified for the object to which the point belongs, and a new image is traced into the refresh buffer. This is diagramed in Figure 4.

The first section of this chapter is devoted to a description of the user level routines and their

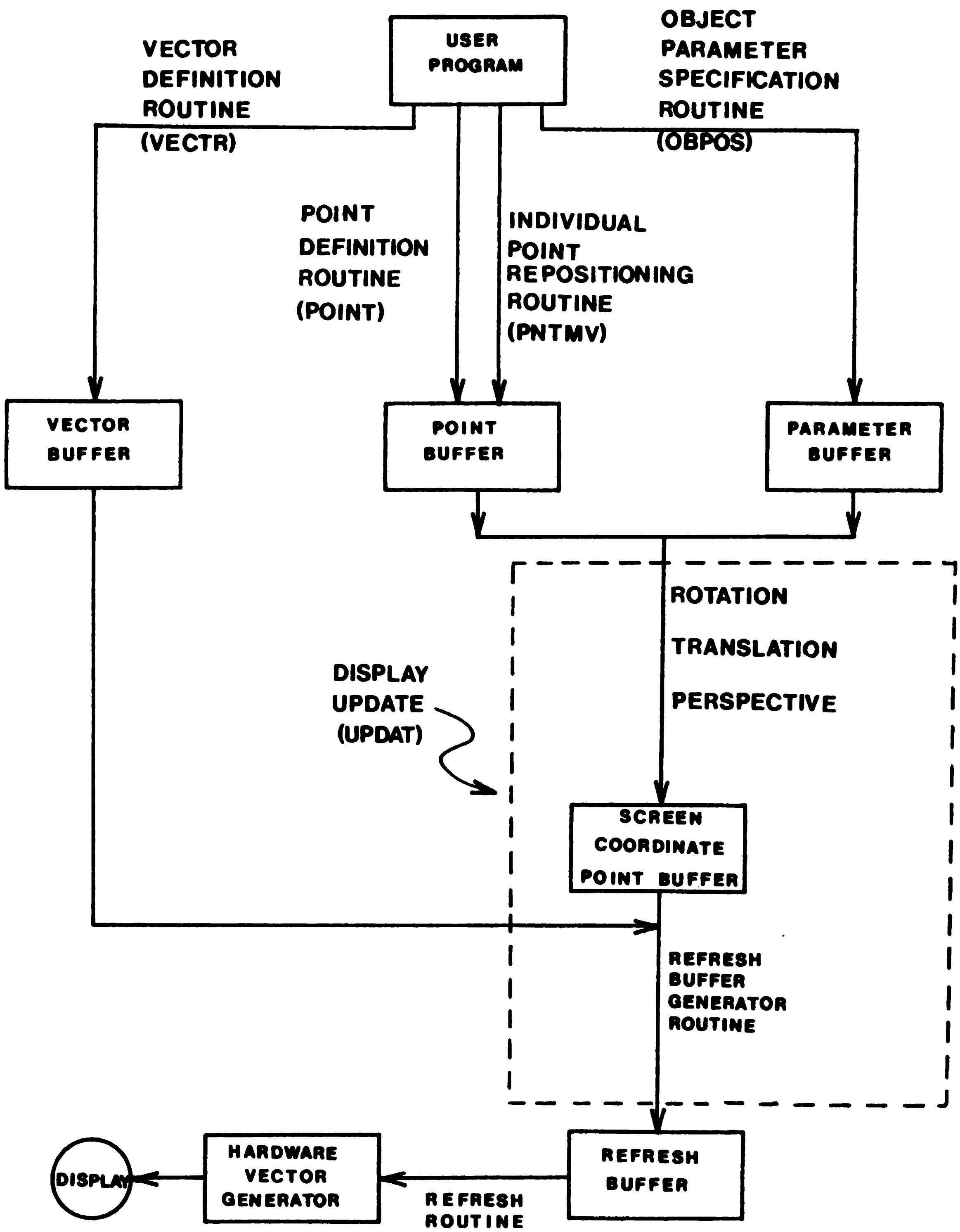


FIGURE 4. SOFTWARE OVERVIEW

use. The second section is a description of the operation of the transformation software.

3.1. User Level Routines

These routines are primarily designed to show how the basic transformation and display software may be interfaced to user-oriented software, and to show that the transformation software is capable of functioning in real time (as Csuri notes, "Too often systems which are the result of a research experiment in hardware or software design do not go beyond a beautiful demonstration of potentialities.")[B2]

The software is object oriented, and all computations are performed in the object coordinate system, down to the final transformation into screen coordinates. The software is in three basic categories: display control, object generation, and object manipulation.

3.1.1. Display Control

3.1.1.1. Activate Refresh--VGØN

The routine VGØN first checks an initialization flag, and initializes the display software system if it is not set. In the prototype system, the hardware does not have its own power supplies, and derives its power (+10, -10, and +5 volts) from analog outputs on the IBM 1800. So, as part of the initialization process, these analog output lines are set to the appropriate voltages. Also, the point and vector counts are set to zero. Finally, the initialization flag is set, to prevent subsequent call to VGØN from causing reinitialization, and the refresh programming is activated. This activation consists of replacing a timer interrupt vector with the address of the refresh program, on the prototype system. The old address is saved (see section 3.1.1.4).

The second phase of VGØN is always performed, regardless of the setting of the initialization flag. In this phase the timer is set for the time that is to elapse between the end of one refresh cycle and the start of the next, and started. VGØN then returns to the calling program, with the refresh programming maintaining the display.

3.1.1.2. Deactivate Refresh--VGØFF

This routine turns off the timer, preventing timer interrupts and thus disabling the refresh programming. VGØFF then returns to the calling program, with the display screen blank.

3.1.1.3. Erase Screen--ERASE

The ERASE routine is used to clear the screen, and should be called only when the refresh programming is active. ERASE first turns off the timer, disabling the refresh programming. Then, the refresh buffer is cleared, and the refresh vector count is set to zero. Finally, the timer is restarted, reactivating the refresh programming, with the display screen blank.

3.1.1.4. Reset System--VGEND

The VGEND routine restores the timer interrupt address (see section 3.1.1.1.) after disabling the timer, thus restoring the operating system of the computer to its original state, in preparation for terminating the run of the graphics program. VGEND should be called only after the graphics software has been initialized (by a call to VGØN), so that the location reserved for the saved interrupt address indeed contains that address. VGEND returns to the calling program, not to the operating system, and thus the system may be restarted with VGØN.

3.1.2. Object Generation

3.1.2.1. Point Generation--PØINT(IOBJ,IX,IY,IZ)

The PØINT routine increments the point count and stores the coordinates of the point specified by the integer triple (IX,IY,IZ). Points thus stored are used as vertices for wire frame representations of objects. An integer object number, IØBJ, is associated with the point, so that all points with the same object number can be transformed as a single set, or "object" (see section 3.1.3). The actual value of IØBJ is not important, as long as the same value is used for other points that are to be in the same object, and the value used does not exceed the maximum number of objects for which the system is configured. For example, the prototype system is configured for a maximum of ten objects, so the integers from one to ten, inclusive, may be used as object numbers. Also, the number of points which may be created is limited by the amount of storage which the graphics software has reserved for them. In the prototype system, up to two hundred points may be created. Although points are transformed in groups according to object number, vectors may span between any two points, and thus between points belonging to different objects (see section 3.1.2.2).

3.1.2.2. Vector Definition--VECTR(IPNT,IBEAM)

The routine VECTR creates a vector from the current point to the point with integer ordinal IPNT. That is, the endpoint of the vector to be created is the point which was created in the IPNT-th call to PØINT. The starting point for the vector is the endpoint of the previous vector, so that the starting point of the first vector is undefined. The integer argument IBEAM controls the intensification of the vector; if IBEAM is one, the vector is visible, and if IBEAM is two, the vector is invisible. Thus, the first call to VECTR should always have IBEAM equal to two, to ensure that the vector with the undefined initial point is not visible. Since a point ordinal is specified to determine the endpoint of each vector, a vector may span between any two points, even if the points are in different "objects" for the purposes of transformations. Only the points are subjected to transformation, and when the transformation parameters change so that the screen coordinates of a point change, all the vectors which have that point as an endpoint are altered so as to end at the new position of the point. Thus, the vectors act as "rubber bands" stretched between their endpoints, and automatically follow the motions of the endpoints, stretching and shrinking as the points move relative to one another. There is a maximum number of vectors which may be defined; the prototype system is configured for a maximum of two hundred vectors. In order to save time, no checking is performed against this limit, and the results are unpredictable if it is exceeded.

3.1.3. Object Manipulation

3.1.3.1. Object Positioning and Orientation-- ØBPØS(IØBJ,IXANG,IYANG,IZANG,IXMØV,IYMØV,IZMØV)

The ØBPØS routine allows the user to specify rotational and translational transformations to be applied to the set of points with object number IØBJ. First, the specified set of points is rotated IXANG degrees about X-axis. Then, the result is rotated IYANG degrees about the Y-axis, and then IZANG degrees about the Z-axis. The sense of rotation is right-handed for the X and Y-axes, and left-handed for the Z-axis, meaning that a positive angle specification will cause a rotation in the direction of the curl of the fingers of the appropriate hand, if the thumb is pointing in the positive direction of a given axis. The result of the rotation transformations is then translated IXMØV units in the direction of the positive X-axis (to the right on the screen), IYMØV units in the direction of the positive Y-axis (towards the top of the screen), and IZMØV units in the direction of the positive Z-axis ("into" the screen). The position and orientation of the point set on the screen does not change at the time of the call to ØBPØS. Rather, the parameters of the call are stored in the parameter buffer for that particular point set, and the new transformations are performed when the display buffer is updated by a call to the UPDAT routine (see section 3.1.3.3).

3.1.3.2. Individual Point Repositioning-- PNTMV(IØBJ,IPNT,IXMØV,IYMØV,IZMØV)

The PNTMV routine allows individual point positions to be changed, and thus change the "shape" of a displayed object, even if all of its vertices belong to the same point set. A particular point to be moved is referenced as the IPNT-th point of the IØBJ-th point set, or "object." The translation parameters IXMØV,IYMØV, and IZMØV are added to the originally defined coordinates of the specified point, and the sums replace the original coordinates. No change is made to the display until the UPDAT routine is called to recalculate the transformations (see section 3.1.3.3). The routine can be used to translate an entire point set (by using multiple calls), so that if a rotation and a compensating translation are specified for the translated point set, the effect is to offset the axis of rotation. This increases the versatility of the prototype rotation scheme (see section 3.2.1).

3.1.3.3. Display Buffer Update--UPDAT

The UPDAT routine performs the transformations specified by calls to ØBPØS on the points defined by calls to PØINT and modified by calls to PNTMV. It then creates a display buffer by tracing through the point-to-point vectors specified by calls to VECTR, using the screen coordinate representations of points resulting from the perspective projection. Then, while inhibiting display refresh, the new display buffer is copied over into the refresh buffer area, and the refresh vector count is set to the vector count of the new buffer. Thus, all transformations which are specified between calls up to UPDAT do not take effect until UPDAT is called. After the refresh buffer is updated, refresh is reactivated, and a refresh cycle is begun immediately, in order to minimize the time the display is blank after the buffer transfer is complete.

3.1.4. Example of Software Use

Figure 5 illustrates the use of the user level routines in a short FORTRAN program segment. First, the call to VGØN initializes the software and activates the display. Next, a small square base pyramid is defined by calls to PØINT and VECTR. The pyramid is as high as its base is wide. The first DØ loop repeatedly calls ØBPØS and UPDAT, moving the pyramid slowly away from the user while rotating it twice about its vertical axis. After this, VGØFF is called, blanking the display, and a square is defined which surrounds the pyramid, by calls to PØINT and VECTR. A call to ØBPØS specifies no translations or rotations for this new object, and when VGØN is called to reactivate the display, it shows the square in the plane of the screen, and the pyramid quite far away. Note that the plane of the screen is just the plane of perspective projection, and objects can pass through it with no effect, since the eye point is located along the negative Z-axis.

```
      CALL VGON
      CALL POINT(1,1000,0,1000)
      CALL POINT(1,1000,0,-1000)
      CALL POINT(1,-1000,0,-1000)
      CALL POINT(1,-1000,0,1000)
      CALL POINT(1,0,2000,0)
      CALL VECTR(1,2)
      CALL VECTR(2,1)
      CALL VECTR(3,1)
      CALL VECTR(4,1)
      CALL VECTR(1,1)
      CALL VECTR(5,1)
      CALL VECTR(2,1)
      CALL VECTR(3,2)
      CALL VECTR(5,1)
      CALL VECTR(4,1)
      DO 100 J=1,720
      CALL OBPOS(1,0,J,0,0,0,J*10)
      CALL UPDAT
  100 CONTINUE
      CALL VGOFF
      CALL POINT(2,3000,3000,0)
      CALL POINT(2,-3000,3000,0)
      CALL POINT(2,-3000,-3000,0)
      CALL POINT(2,3000,-3000,0)
      CALL VECTR(6,2)
      CALL VECTR(7,1)
      CALL VECTR(8,1)
      CALL VECTR(9,1)
      CALL VECTR(6,1)
      CALL OBPOS(2,0,0,0,0,0,0)
      CALL VGON
      DO 200 J=1,720
      CALL OBPOS(1,0,0,0,0,0,7200-J*10)
      CALL UPDAT
  200 CONTINUE
      DO 300 J=1,360
      CALL OBPOS(1,J,0,0,0,0,0)
      CALL OBPOS(2,J,0,0,0,0,0)
      CALL UPDAT
  300 CONTINUE
      CALL VGEND
```

FIGURE 5. EXAMPLE OF PROTOTYPE SYSTEM SOFTWARE USE

The second DØ loop serves only to move the pyramid back to the plane of the screen, while the square remains stationary. The third DØ loop rotates the square and pyramid in unison about the X-axis once. This causes a dramatic perspective effect as the apex of the pyramid and the top and bottom of the square alternately approach and recede. After this single rotation, the display system is reset with a call to VGEND in preparation for terminating execution of the program. If the program had a new set of images to display, ERASE could be called instead of VGEND to remove the previously defined objects from the graphics system.

3.2. Transformation Software

The transformation routines are a critical part of a real time graphics system which does not have transformation hardware. If the display is to be updated often enough to maintain the illusion of motion, the routines which perform translation, rotation, and perspective projection must be carefully coded (generally in assembly language) so as to be as fast as possible. As few minicomputers possess floating point arithmetic hardware, it is generally impossible to attain the necessary speed unless all the transformation calculations are performed using integer arithmetic. The integer multiply and divide instructions found on many minicomputers allow the transformations to be rapidly performed.

3.2.1. Rotation

Rotation requires the evaluation of sums of products, where the products are of object coordinates and the sine or cosine of a rotation angle. This involves two difficulties: the evaluation of trigonometric functions, and the handling of numbers less than unity in magnitude, by using integer arithmetic. In keeping with the integer orientation of the low level display programming, an angle of rotation is represented as an integer, indicating number of degrees of rotation. Thus, if the angle is treated modulo 360, it may be used as a subscript to find the required values in precalculated tables. It is common for minicomputer divide instructions to have provisions for obtaining the remainder upon division, so that the modulo operation may be performed very quickly. However, if the divide instruction lacks this feature, the remainder may be computed in a manner analogous to the FORTRAN expression

```
N - N / 360 * 360  .
```

Once the angle is reduced to between zero and 359 degrees by the modulo 360 operation, it may be used directly as a subscript to look up the corresponding sine and cosine values, if enough memory is available to contain the required tables. Each value occupies one word, and two tables are required, so a total of 720 words is necessary, if separate tables are used. However, because of the identity

$$\cos(\theta) = \sin(\theta + 90°)$$

the cosine table can start at the ninety-first entry in the sine table, so that the combined sine-cosine table occupies 450 words. Although this is still a good deal of memory to use for a table, it is not excessive, and yields a very fast technique for obtaining the values of sine and cosine. By exploiting the symmetries of these functions, further compression of the table can be accomplished, at the expense of more programming to reference the table and a longer access time. If sufficient memory is available to contain the larger table, then such compression efforts serve only to lower the performance of the system.

The significance of the one degree resolution of the trigonometric functions is two-fold. First, it allows the use of a very natural technique for specifying angles: an integer number of degrees. This is consistent with the idea of avoiding floating point arithmetic, and is convenient for the user of the system. Second, it provides a simple method for programming animation. For example, if an object is rotated through an angle which is increased by one degree per frame, say, then the object will appear to rotate continuously, if slowly, even though each change of angle would be barely perceptible, if viewed by itself. Thus, a number of rotation rates may be programmed by incrementing rotation angles by small integers. Because of the modulo 360 computation involved in the sine and cosine evaluations, angles in excess of three-hundred sixty degrees are permitted, and in fact angles as large as the maximum integer size imposed by the word length of the minicomputer can be used. In the prototype system, this limit is 32767, and thus an object will rotate more than ninety times before integer overflow occurs, causing a sudden jerk in the otherwise smooth rotation.

Now, the required calculations may be performed if the actual numbers stored in the trigonometric table are scaled to fit the available integer range of the minicomputer. Thus, with the sixteen bit orientation of the prototype system, the trigonometric table entries are scaled by a factor of 32767. A typical integer multiply instruction will form a thirty-two bit product from two sixteen bit numbers (assuming a sixteen bit word), so if a number is multiplied by a value from the trigonometric table, the most significant word of the product will be within one least significant bit of half of the desired product. This is true because the table is scaled by a factor of 32767, and the result of the multiplication is scaled by a factor of 1/65536, since only the most significant half of the product is used. Therefore, two of the aforementioned "rotation products" may be computed and added without fear of overflow, or they may be shifted one bit to the left before or after the addition, yielding the correct result, within the two least significant bits.

The actual transformation equations used to implement rotation depend upon the means chosen to represent object orientation. One commonly used technique specifies an axis of rotation (in terms of direction cosines) and an angle through which the rotational motion acts. This is somewhat inconvenient for the user, since direction cosines are not a natural way to specify a direction. Unfortunately, there seems to be no convenient way for the user to specify a given rotational motion. The prototype system attacks this problem by providing the user with three ordered rotation transformations: X-axis, Y-axis, and Z-axis rotations, where the user specifies the number of degrees of each rotation. The transformations are applied in the order shown, allowing the general orientation capability offered by the direction cosine techniques above, and also providing a natural way for the user to specify certain classes of orientations. This method has performed moderately well, in terms of ease of use. Also, it allows a good deal of computation optimization, thus helping to assure smooth animation (see section 3.2.4).

3.2.2. Translation

Translation can be implemented by simply adding the appropriate integer displacements to each coordinate of each point involved. If all three coordinates are treated identically, however, the resulting "coordinate space" will be cubical, due to integer range limitations, and thus has very limited depth and perspective cues. This can be remedied by treating the Z-coordinate separately (see section 3.2.6).

3.2.3. Perspective Projection

Generating a perspective image requires division, which is generally a time-consuming operation, and thus perspective projection is not provided in many inexpensive real time display systems. If the display screen is the plane Z = 0, the eye point (X_e,Y_e,Z_e) with $Z_e < 0$, and the point to be projected (X_e,Y_e,Z_e), the projected point in screen coordinates is

$$X_s = X_e + (X-X_e) * \frac{Z_e}{Z_e - Z}$$

$$Y_s = Y_e + (Y-Y_e) * \frac{Z_e}{Z_e - Z} .$$

With the exception of the division, the equations are trivial to implement using integer arithmetic. A typical integer division instruction acts on a thirty-two bit dividend and a sixteen bit divisor to produce a sixteen bit quotient and a sixteen bit remainder, if possible, and this instruction can be used to provide the division required by the perspective projection equations. If the numerator is placed in the thirty-two bit register and divided by the denominator, the required quotient is obtained, scaled by a power of two dependent upon where the numerator was positioned in the dividend register. This scaling can be used to increase the depth of field available from the system (see section 3.2.6).

3.2.4. Concatenation of Transformations

A significant increase in the speed of the transformations can be realized if they are applied simultaneously rather than one at a time. In a typical graphics system, the transformations are represented as matrices, and the transformations are applied by multiplying the vector representation of the points to be transformation matrices, so time may be saved by computing the product of all the transformation matrices beforehand, since matrix multiplication is associative. While the matrix implementation is straightforward and versatile, since transformations may be added or deleted at will, it is relatively slow, and thus is a luxury that cannot be afforded in a small real time system. Instead, concatenation of transformations is accomplished while the transformations themselves are being performed. Taking the prototype system as an example, the order of transformations is X-axis, Y-axis, and Z-axis rotations, X-axis, and Z-axis translations, and X-axis and Y-axis screen coordinate perspective projections. But instead of implementing them in this order strictly, the software overlaps the calculations as much as the available registers will allow. The results of each calculation are used immediately, if possible, to eliminate unnecessary load and store operations. This saves precious time during the operation of transforming objects from their original position and orientation down to screen coordinates, which can make the difference between a real time system and a jerky, eye-fatiguing graphics display. The integrated character of the transformation coding in the prototype system is evident from the program listing (see appendix).

3.2.5. Transformation Simplification

Further enhancement of transformation speed is possible if the transformation equations are reduced to their simplest practical state. In the perspective transformation equations, one addition and one subtraction can be eliminated from each equation if the eyepoint is constrained to lie on the line X = Y = 0. This limits the system to displays in which the projection plane and the observer are stationary, and the objects being displayed move relative to them. This is not a serious limitation, because the user of the system is always facing in the direction of the display, and thus should receive the impression that the objects which he is observing are in motion, rather than himself.

Three operations of addition or subtraction may be eliminated from each rotation equation, if all the axes of rotation pass through the origin:

$$X' = (X - X_o) \cdot \cos(\theta_z) - (Y - Y_o) \cdot \sin(\theta_z) + X_o$$

becomes

$$X' = X \cdot \cos(\theta_z) - Y \cdot \sin(\theta_z)$$

where (X_o, Y_o) is the point where the axis of rotation (here, parallel to the Z-axis) intersects the plane Z = 0. Since as many as six rotation equations must be implemented (see section 3.1.3.1), the elimination of these operations can save a good deal of time in processing the transformations. In the prototype system, this saves about 18 percent of the time which would otherwise be spent in performing rotation computations: 104 microseconds versus 128 microseconds per expression evaluation. The IBM 1800 has a relatively slow multiplication instruction, so that a machine with a faster multiplication instruction will realize a greater percentage gain in time by using the simplified equations (although the faster machine would probably not need the increase in speed).

3.2.6. Transform Limitations

Because sixteen bit integers are used throughout the transformation calculations (using the prototype system as an example), the coordinates used to define points (object vertices) must be sixteen bit integers. This means that the object space is a cube, centered on the origin (the center of the display screen), and approximately 65,000 units on a side. This is many times larger than the display screen (typically about 1,000 units in diameter), but lacks sufficient depth for convincing perspective effects. This is overcome by scaling during the concatenation process, so that the depth of field is increased by a factor of sixteen. During the computation of the denominator of the perspective factor, the Z-coordinate of the point being processed is shifted three bits right, instead of the normal one bit left, before the Z-coordinate translation is added. This amplifies the effect of the Z-axis translation by a factor of sixteen, without distorting the cubical object space. Thus, whenever a Z-axis translation is specified by the user, he must be sure to use one sixteenth of the actual translation desired.

4. Summary

The prototype system described in this paper is capable of displaying over 100 vectors, representing as many as ten objects, and perform all the described transformations fast enough to present a flicker-free display. Fewer objects increases its vector count for a flicker-free display, and more still may be displayed by tolerating some degree of flicker. Static displays may have up to 200 points before flicker becomes noticable.

The hardware cost of the system, exclusive of the minicomputer and the oscilloscope, is about thirty dollars. The software must of course be rewritten in assembler for each different minicomputer, but this is not difficult, and generally can be done by adapting the prototype system code, provided in the appendix.

The IBM 1800 used in the prototype system is quite slow, by modern standards, and so a new system, using a minicomputer, should yield markedly better performance.

Thus, the prototype system achieves its goal of being a small scale real time minicomputer graphics system.

APPENDIX

```
        ENT     UPDAT
        ENT     OBPOS
        ENT     POINT
        ENT     VECTR
        ENT     VGOFF
        ENT     VGON
        ENT     VGEND
        ENT     ERASE
        ENT     PNTMV
****************************************************
****************************************************
***                                              ***
***                                              ***
***     VECTOR GENERATOR SOFTWARE VERSION 2      ***
***                                              ***
***                                              ***
***     MASTER'S THESIS PROGRAM                  ***
***     CHRISTOPHER SCUSSEL                      ***
***     MICHIGAN STATE UNIVERSITY                ***
***     ENGINEERING COLLEGE COMPUTER OPERATIONS  ***
***                                              ***
***                                              ***
****************************************************
****************************************************
*
*
****************************************************
*                                                  *
*       SOFTWARE CONFIGURATION CONTROL             *
*                                                  *
****************************************************
*
MXPNT  EQU      200      SET 200 POINTS MAXIMUM
MXVEC  EQU      200      SET 200 VECTORS MAXIMUM
MXOBJ  EQU      10       SET 10 OBJECTS MAXIMUM
LOGC0  EQU      0        SET LOGIC 0 TO 0 VOLTS
LOGC1  EQU      10000    SET LOGIC 1 TO 3 VOLTS
*
*
```

```
****************************************************
*                                                  *
*      SOFTWARE PARAMETER STORAGE                  *
*                                                  *
****************************************************
*
=PNTS DC                    NUMBER OF POINTS
=VEC  DC                    NUMBER OF VECTORS
=VREF DC                    NUMBER OF VECTORS TO BE
*                           DISPLAYED BY REFRESH ROUTINE
LOGO  DC        LOGCO       ADDRESSED LOGIC 0
LOG1  DC        LOGC1       ADDRESSED LOGIC 1
*
****************************************************
*                                                  *
*      DISPLAY UPDATE ROUTINE                      *
*                                                  *
****************************************************
*
UPDAT DC                    ENTRY POINT
      LD    L  INTFG        FETCH INITIALIZATION FLAG
      BSC   I  UPDAT,+-        RETURN IF NOT INITIALIZED
      STX   L3 X3NIN        SAVE INDEX REGISTER 3
*
*     ROTATE, TRANSLATE, AND PROJECT POINTS
*     DOWN TO SCREEN COORDINATES
*
      LDX   L1 PTSTR        FETCH POINT STORAGE ADDRESS
      LDX   I3 =PNTS        FETCH NUMBER OF POINTS
MOVLP LD     1 0            FETCH PARAMETER STORAGE ADDRESS
      STO      TEMP1
      LDX   I2 TEMP1        PREPARE TO RETRIEVE PARAMETERS
      LD     1 2            GET OBJECT Y-COORD
      M      2 0            MULT BY SIN X
      STO      TEMP1        STORE INTERMEDIATE
      LD     1 3            GET OBJECT Z-COORD
      M      2 1            MULT BY COS X
      S        TEMP1        SUBTRACT INTERMEDIATE
      SLA      1            CORRECT FOR TRIG TABLE
      STO      TEMP3        STORE X-ROT Z-COORD
      LD     1 2            GET OBJECT Y-COORD
      M      2 1            MULT BY COS X
      STO      TEMP1        STORE INTERMEDIATE
      LD     1 3            GET OBJECT Z-COORD
      M      2 0            MULT BY SIN X
      A        TEMP1        ADD INTERMEDIATE
      SLA      1            CORRECT FOR TRIG TABLES
      STO      TEMP2        STORE X-ROT Y-COORD
      LD       TEMP3        GET X-ROT Z-COORD
      M      2 2            MULT BY SIN Y
```

```
      STO      TEMP1    STORE INTERMEDIATE
      LD     1 1        GET OBJECT X-COORD
      M      2 3        MULT BY COS Y
      S        TEMP1    SUBTRACT INTERMEDIATE
      SLA      1        CORRECT FOR TRIG TABLES
      STO      TEMP4    STORE XY-ROT X-COORD
      LD       TEMP3    GET X-ROT Z-COORD
      M      2 3        MULT BY COS Y
      STO      TEMP1    STORE INTERMEDIATE
      LD     1 1        GET OBJECT X-COORD
      M      2 2        MULT BY SIN Y
      A        TEMP1    ADD INTERMEDIATE
      SRT      3        CORRECT TRIG AND ADJUST PERSPECTIVE
      A      2 8        ADD Z-TRANSLATION
      A        EYEZ     COMPUTE DISTANCE FROM EYE
      STO      TEMP3    STORE PERSPECTIVE FACTOR
      LD       TEMP2    GET XY-ROT Y-COORD
      M      2 4        MULT BY SIN Z
      STO      TEMP1    STORE INTERMEDIATE
      LD       TEMP4    GET XY-ROT X-COORD
      M      2 5        MULT BY COS Z
      S        TEMP1    SUBTRACT INTERMEDIATE
      SLA      1        CORRECT FOR TRIG TABLE
      A      2 6        ADD X-TRANSLATION
      SRT      6        ADJUST PERSPECTIVE
      D        TEMP3    COMPUTE PERSPECTIVE
      STO    1 4        STORE X-SCREEN COORDINATE
      LD       TEMP2    GET XY-ROT Y-COORD
      M      2 5        MULT BY COS Z
      STO      TEMP1    STORE INTERMEDIATE
      LD       TEMP4    GET XY-ROT X-COORD
      M      2 4        MULT BY SIN Z
      A        TEMP1    ADD INTERMEDIATE
      SLA      1        CORRECT FOR TRIG TABLE
      A      2 7        ADD Y-TRANSLATION
      SRT      6        ADJUST PERSPECTIVE
      D        TEMP3    COMPUTE PERSPECTIVE
      STO    1 5        STORE Y-SCREEN COORDINATE
      MDX    1 6        ADDRESS OF NEXT PTSTR RECORD
      MDX    3 -1       DECREMENT COUNT
      MDX      MOVLP    DO NEXT POINT
      MDX      IMAGE    PREPARE TO TO UPDATE RFSTR
TEMP1 DC
TEMP2 DC
TEMP3 DC
TEMP4 DC
```

```
*
*       CONSTRUCT IMAGE OF REFRESH STORAGE AREA
*
IMAGE SRA      16       ZERO ACCUMULATOR
      STO      XPOS     INITIALIZE X-POSITION
      STO      YPOS     INITIALIZE Y-POSITION
      LDX   L1 VCSTR    PREPARE TO RETRIEVE VECTORS
      LDX   I2 =VEC     FETCH NUMBER OF VECTORS
      LDX   L3 PLTBF    FETCH PLOT BUFFER ADDRESS
DRWLP LD     1 0        FETCH ENDPOINT ORDINAL
      SLA      1        MULTIPLY BY 2
      A      1 0        FORM PRODUCT WITH 3
      SLA      1        FORM PRODUCT WITH 6
      A        PTADR    COMPUTE X ADDRESS
      A        ADDR4    COMPUTE X ADDRESS
      STO      XSTR+1   STORE ADDRESS
      A        ADDR1    COMPUTE Y ADDRESS
      STO      YSTR+1   STORE Y ADDRESS
XSTR  LD     L *-*      FETCH X-SCREEN COORDINATE
      S        XPOS     COMPUTE DIFFERENCE
      STO    3 0        STORE DIFFERENCE IN PLOT BUFFER
      A        XPOS     RESTORE X-SCREEN COORDINATE
      STO      XPOS     STORE CURRENT X-POSITION
YSTR  LD     L *-*      FETCH Y-SCREEN COORDINATE
      S        YPOS     COMPUTE DIFFERENCE
      STO    3 1        STORE DIFFERENCE IN PLOT BUFFER
      A        YPOS     RESTORE Y-SCREEN COORDINATE
      STO      YPOS     STORE CURRENT Y-POSITION
      LD     1 1        FETCH BEAM CONTROL
      STO    3 2        STORE IN PLOT BUFFER
      MDX    1 2        PREPARE TO FETCH NEXT VECTOR
      MDX    3 3        PREPARE TO STORE NEXT VECTOR
      MDX    2 -1       DECREMENT VECTOR COUNT
      MDX      DRWLP    LOOP BACK FOR NEXT VECTOR
```

```
*
*     TRANSFER PLOT BUFFER TO REFRESH STORAGE
*
      LDX   I1 =VEC     FETCH NUMBER OF VECTORS
      LDX   L2 PLTBF    FETCH PLOT BUFFER ADDRESS
      LDX   L3 RFSTR    FETCH REFRESH STORAGE ADDRESS
      XIO   L  TMOFF    CEASE REFRESHING
XFER  LD     2 0        DATA TRANSFER
      STO    3 0        DATA TRANSFER
      LD     2 1        DATA TRANSFER
      STO    3 1        DATA TRANSFER
      LD     2 2        DATA TRANSFER
      STO    3 2        DATA TRANSFER
      MDX    2 3        INCREMENT ADDRESS
      MDX    3 3        INCREMENT ADDRESS
      MDX    1 -1       DECREMENT VECTOR COUNTER
      MDX      XFER     GO TRANSFER NEXT VECTOR
      LD    L  =VEC     FETCH NUMBER OF VECTORS
      STO   L  =VREF    SET NUMBER OF REFRESH VECTORS
      XIO      LEVL1    TAKE AN IMMEDIATE REFRESH
*                       CYCLE AND RESUME REFRESHING
      LDX   I3 X3NIN    RESTORE INDEX REGISTER 3
      BSC   I  UPDAT    RETURN TO CALLING PROGRAM
*
*     CONSTANTS AND VARIABLES
EYEZ  DC       1024     EYE POINT Z-COORDINATE
XPOS  DC                X FOR GENERATOR CONVERSION
YPOS  DC                Y FOR GENERATOR CONVERSION
PTADR DC       PTSTR-6  ADDRESS FOR ORDINAL POINT RECALL
ADDR1 DC       1        1 FOR USE LOCALLY
ADDR4 DC       4        4 FOR USE LOCALLY
X3NIN DC                TEMPORARY STORAGE FOR XREG 3
*                       IN NONINTERRUPT ROUTINES
*
*     IOCC
*
      BSS   E  0        GET AN EVEN ADDRESS
LEVL1 DC       /4000    LEVEL 1 INTERRUPT BIT SET
      DC       /04A0    PROGRAMMED INTERRUPT IOCC
*
```

```
****************************************************
*                                                  *
*                      REFRESH ROUTINE             *
*                                                  *
****************************************************
RFRSH DC                 REFRESH ROUTINE ENTRY POINT
      STS       RSTS     SAVE STATUS
      STD       RSAVE    SAVE ACCUMULATOR
      STX     1 RSAVE+2  SAVE XREG 1
      STX     2 RSAVE+3  SAVE XREG 2
      XIO       TMOFF    PREVENT TIMER CYCLE STEALS
      LDX    I1 =VREF    GET NUMBER OF REFRESH VECTORS
      MDX     1 1        ADD EXTRA FOR LAST VECTOR
      LDX    L2 RFSTR    FETCH STARTING ADDRESS OF RFSTR
      XIO       RNON     TURN RUN LINE ON
OUTLP LD      2 0        FETCH X-SCREEN COORDINATE
      STO       XOUT     STORE IN ANALOG OUTPUT TABLE
      LD      2 1        FETCH Y-SCREEN COORDINATE
      STO       YOUT     STORE IN ANALOG OUTPUT TABLE
      LD      2 2        FETCH BEAM CONTROL
      STO       BMOUT    STORE IN ANALOG OUTPUT TABLE
      XIO       VCOUT    OUTPUT VECTOR TO GENERATOR
      MDX     2 3        ADVANCE TO NEXT VECTOR ADDRESS
      MDX     1 -1       DECREMENT VECTOR COUNT
      MDX       OUTLP    OUTPUT NEXT VECTOR
      XIO       STDBY    PLACE GENERATOR IN STANDBY MODE
      XIO       TMRST    RESET TIMER
      LD        TIME     FETCH RERESH CYCLE TIME
      STO    L  /0006    STORE AT TIMER LOCATION
      XIO       TMON     TURN TIMER ON
      LDD       RSAVE    RESTORE ACCUMULATOR
      LDX    I1 RSAVE+2  RESTORE XREG 1
      LDX    I2 RSAVE+3  RESTORE XREG 2
RSTS  LDS       0        RESTORE STATUS
      BOSC I    RFRSH    RETURN AND TURN OFF INTERRUPT
*
*
*     CONSTANTS AND VARIABLES
TIME  DC        -3       RERFESH CYCLE TIME
      BSS    E  0        GET EVEN ADDRESS
RSAVE BSS       4        STATUS SAVE STORAGE AREA
```

```
*
*
*       IOCC'S
TMOFF DC        /0000   TIMER STATE -- OFF
      DC        /0420   TIMER STATE CONTROL IOCC
TMON  DC        /2000   TIMER STATE -- C ON
      DC        /0420   TIMER STATE CONTROL IOCC
TMRST DC                DUMMY WORD FOR TIMER STATUS IOCC
      DC        /0721   TIMER STATUS SENSE IOCC
RNON  DC        LOG1    SEND OUT LOGIC 1 ON
      DC        /6105   LINE 5, THE RUN LINE
VCOUT DC        VOTAB   TABLE ADDRESS FOR
      DC        /6500   ANALOG OUTPUT INITIALIZE WRITE
STDBY DC        SBTAB   TABLE ADDRESS FOR
      DC        /6500   ANALOG OUTPUT INITIALIZE WRITE
*
*
*       ANALOG OUTPUT TABLES
*       ------ ------ ------
*       TABLE TO OUTPUT VECTOR TO GENERATOR
VOTAB DC        /400A   SINGLE SCAN WITH NO INTERRUPT
      DC        7       ATTACH INPUT SWITCH LINE AND
      DC        LOGC0   TURN IT OFF
      DC        2       ATTACH X-OUTPUT LINE AND
XOUT  DC                SEND X-VOLTAGE OUT
      DC        4       ATTACH Y-OUTPUT LINE AND
YOUT  DC                SEND Y-VOLTAGE OUT
      DC        3       ATTACH BEAM CONTROL LINE AND
BMOUT DC                SEND BEAM VOLTAGE OUT
      DC        7       ATTACH INPUT SWITCH LINE AND
      DC        LOGC1   TURN IT ON
*
*       TABLE FOR VECTOR GENERATOR STANDBY MODE
SBTAB DC        /400A   SINGLE SCAN WITH NO INTERRUPT
      DC        7       ATTACH INPUT SWITCH LINE AND
      DC        LOGC0   TURN IT OFF
      DC        3       ATTACH BEAM CONTROL LINE AND
      DC        32767   TURN IT OFF WITH 10 VOLTS
      DC        5       ATTACH RUN LINE AND
      DC        LOGC0   TURN IF OFF
      DC        2       ATTACH X-OUTPUT LINE AND
      DC        0       SET IT TO 0
      DC        4       ATTACH Y-OUTPUT LINE AND
      DC        0       SET IT TO 0
*
```

```
*****************************************************
*                                                   *
*     OBJECT PARAMETER ENTRY ROUTINE                *
*                                                   *
*****************************************************
*
OBPOS DC                   ENTRY POINT
      STX    L3 X3NIN      SAVE XREG 3
      LDX    I1 OBPOS      FETCH PARAMETER ADDRESS TABLE
      LD     I1 0          FETCH OBJECT ORDINAL
      SLA       3          MULTIPLY BY 8
      A      I1 0          FORM PRODUCT WITH 9
      A         PRMAD      ADD PARAMETER TABLE ADDRESS
      STO       TEMP5
      LDX    I2 TEMP5      TRANSFER TO XREG 2
      LD     I1 1          FETCH X-ANGLE
      BSI       ANGFX      NORMALIZE ANGLE INTO XREG 3
      LD     L3 SIN        FETCH SINE OF ANGLE
      STO     2 0          STORE IN PARAMETER TABLE
      LD     L3 COS        FETCH COSINE OF ANGLE
      STO     2 1          STORE IN PARAMETER TABLE
      LD     I1 2          FETCH Y-ANGLE
      BSI       ANGFX      NORMALIZE ANGLE INTO XREG 3
      LD     L3 SIN        FETCH SINE OF ANGLE
      STO     2 2          STORE IN PARAMETER TABLE
      LD     L3 COS        FETCH COSINE OF ANGLE
      STO     2 3          STORE IN PARAMETER TABLE
      LD     I1 3          FETCH Z-ANGLE
      BSI       ANGFX      NORMALIZE ANGLE INTO XREG 3
      LD     L3 SIN        FETCH SINE OF ANGLE
      STO     2 4          STORE IN PARAMETER TABLE
      LD     L3 COS        FETCH COSINE OF ANGLE
      STO     2 5          STORE IN PARAMETER TABLE
      LD     I1 4          FETCH X-TRANSLATION
      STO     2 6          STORE IN PARAMETER TABLE
      LD     I1 5          FETCH Y-TRANSLATION
      STO     2 7          STORE IN PARAMETER TABLE
      LD     I1 6          FETCH Z-TRANSLATION
      STO     2 8          STORE IN PARAMETER TABLE
      LDX    I3 X3NIN      RESTORE XREG 3
      BSC    L1 7          RETURN TO CALLING PROGRAM
```

```
*     ANGLE NORMALIZATION ROUTINE
ANGFX DC                ENTRY POINT
      SRT       16      PREPARE FOR DIVIDE
      D         '360'   DIVIDE BY 360
      SLT       16      GET REMAINDER
      BSC       +Z      RESULT READY IF + OR 0
      A         '360'   FORCE REMAINDER TO BE NONNEGATIVE
      STO       TEMP5   PREPARE TO LOAD INDEX REGISTER
      LDX   I3  TEMP5   PUT RESULT IN XREG 3
      BSC   I   ANGFX   RETURN TO CALLING PROGRAM
*
*
*     CONSTANTS AND VARIABLES
PRMAD DC        OBPRM-9 ADDRESS OF OBJECT PARAMETER TABLE
TEMP5 DC                LOCAL TEMPORARY
'360' DC        360     USED IN ANGLE MANIPULATIONS
*
*
*****************************************************
*                                                   *
*     POINT ENTRY ROUTINE                           *
*                                                   *
*****************************************************
*
POINT DC                ENTRY POINT FOR POINT ENTRY
      LD    L   =PNTS   FETCH NUMBER OF POINTS
      A         ONE1    ADD 1
      STO   L   =PNTS   STORE NEW NUMBER OF POINTS
      SLA       1       MULTIPLY BY 2
      A     L   =PNTS   FORM PRODUCT WITH 3
      SLA       1       FORM PRODUCT WITH 6
      A     L   PTADR   ADD POINT STORAGE ADDRESS
      STO       TEMP6   PREPARE TO LOAD XREG 2
      LDX   I2  TEMP6   FETCH POINT ADDRESS
      LDX   I1  POINT FETCH PARAMETER TABLE ADDRESS
      LD    I1  0       FETCH OBJECT ORDINAL
      SLA       3       MULTIPLY BY 8
      A     I1  0       FORM PRODUCT WITH 9
      A     L   PRMAD   ADD PARAMETER TABLE ADDRESS
      STO    2  0       STORE AT POINT STORAGE ADDRESS
      LD    I1  1       FETCH X-COORDINATE
      STO    2  1       STORE AT POINT STORAGE ADDRESS
      LD    I1  2       FETCH Y-COORDINATE
      STO    2  2       STORE AT POINT STORAGE ADDRESS
      LD    I1  3       FETCH Z-COORDINATE
      STO    2  3       STORE AT POINT STORAGE ADDRESS
      BSC   L1  4       RETURN TO CALLING PROGRAM
```

```
*****************************************************
*                                                   *
*     VECTOR ENTRY ROUTINE                          *
*                                                   *
*****************************************************
*
VECTR DC                   ENTRY POINT FOR VECTOR ENTRY
      LD    L   =VEC       FETCH NUMBER OF VECTORS
      A         ONE1       ADD 1
      STO   L   =VEC       STORE NEW NUMBER OF VECTORS
      SLA       1          MULTIPLY BY 2
      A         VCADR      ADD VECTOR STORAGE ADDRESS
      STO       TEMP6      PREPARE TO LOAD XREG 2
      LDX   I2  TEMP6      FETCH VECTOR STORAGE ADDRESS
      LDX   I1  VECTR      FETCH PARAMETER TABLE ADDRESS
      LD    I1  0          FETCH ORDINAL POINT REFERENCE
      STO    2  0          STORE AT VECTOR STORAGE ADDRESS
      LD    I1  1          FETCH WRITE CONTROL
      S         ONE1       SUBTRACT 1
      BSC       Z          SKIP IF WRITE CONTROL WAS 1
      LD        BMOFF      FETCH BEAM OFF VALUE
      STO    2  1          STORE BEAM CONTROL IN VECTOR ENTRY
      BSC   L1  2          RETURN TO CALL ING PROGRAM
*
*     CONSTANTS AND VARIABLES
TEMP6 DC                   LOCAL TEMPORARY
ONE1  DC        1          1 FOR USE LOCALLY
VCADR DC        VCSTR-2    VECTOR STORAGE ADDRESS
BMOFF DC        32767      BEAM OFF CONTROL VALUE
```

```
*****************************************************
*                                                   *
*     POINT MOVE ROUTINE                            *
*                                                   *
*****************************************************
*
PNTMV DC                 ENTRY POINT FOR POINT MOVE
      STX   L3 X3NIN     SAVE XREG 3
      LDX   I1 PNTMV     FETCH PARAMETER TABLE ADDRESS
      LD    I1 0         FETCH OBJECT ORDINAL
      SLA      3         MULTIPLY BY 8
      A     I1 0         FORM PRODUCT WITH 9
      A        PRMAD     ADD PARAMETER TABLE ADDRESS
      STO      OBCOD     STORE ADDRESS FOR COMPARISON
      LD    I1 1         FETCH NUMBER OF CHANGE POINT
      STO      PTNUM     STORE AS COUNTER
      LDX   L2 PTSTR     FETCH POINT STORAGE ADDRESS
      LDX   I3 =PNTS     FETCH TOTAL NUMBER OF POINTS
PSRCH LD     2 0         FETCH PARAMETER TABLE ADDRESS
      CMP      OBCOD     COMPARE WITH NEEDED ADDRESS
      MDX      PSRC1     PREPARE FOR NEXT POINT
      MDX      PSRC1     PREPARE FOR NEXT POINT
      MDX   L  PTNUM,-1 CORRECT OBJECT,DECREMENT COUNT
      MDX      PSRC1     PREPARE FOR NEXT POINT
      LD    I1 2         FETCH X-DISPLACEMENT
      A      2 1         ADD TO CURRENT X-COORDINATE
      STO    2 1         STORE NEW X-COORDINATE
      LD    I1 3         FETCH Y-DISPLACEMENT
      A      2 2         ADD CURRENT Y-COORDINATE
      STO    2 2         STORE NEW Y-COORDINATE
      LD    I1 4         FETCH Z-DISPLACEMENT
      A      2 3         ADD Z-COORDINATE
      STO    2 3         STORE NEW Z-COORDINATE
      LDX   I3 X3NIN     RESTORE XREG 3
      BSC   L1 5         RETURN TO CALLING PROGRAM
PSRC1 MDX    2 6         ADVANCE ADDRESS TO NEXT POINT
      MDX    3 -1        DECREMENT POINT COUNT
      MDX      PSRCH     SEARCH NEXT POINT
      BSC   L1 5         RETURN TO CALLING PROGRAM,
*                        SPECIFIED POINT NOT FOUND,
*                        NO ACTION TAKEN.
*
*     CONSTANTS AND VARIABLES
OBCOD DC                 OBJECT PARAMETER TABLE ADDRESS
*                        USED TO FIND POINTS IN THE
*                        DESIRED OBJECT
PTNUM DC                 COUNTS NUMBER OF POINTS TO GO
*                        IN DESIRED OBJECT BEFORE THE
*                        DESIRED POINT IS FOUND.
```

```
*****************************************************
*                                                   *
*     HARDWARE INITIALIZATION AND DISPLAY CONTROL   *
*                                                   *
*****************************************************
*
*     DISPLAY ACTIVATION
*
VGON  DC                ENTRY POINT FOR DISPLAY ACTIVATION
      LD       INTFG    FETCH INITIALIZATION FLAG
      BSI  L   INIT,+-  INITIALIZE IF FIRST CALL
      LD   L   TIME     FETCH INTERVAL FOR TIMER
      STO  L   /0006    SET TIMER FOR REFRESH INTERVAL
      XIO  L   TMON     TURN TIMER ON
      BSC  I   VGON     RETURN TO CALLING PROGRAM
*
*     DISPLAY DEACTIVATION
*
VGOFF DC                ENTRY POINT FOR DEACTIVATION
      XIO  L   TMOFF    TURN TIMER OFF
      BSC  I   VGOFF    RETURN TO CALLING PROGRAM
*
*     DISPLAY ERASE
*
ERASE DC                ENTRY POINT FOR DISPLAY ERASE
      BSI      VGOFF    DEACTIVATE DISPLAY
      LD       INITO    FETCH 0
      STO  L   =PNTS    SET NUMBER OF POINTS TO ZERO
      STO  L   =VEC     SET NUMBER OF VECTORS TO ZERO
      STO  L   =VREF    SET NUMBER OF REFRESH
*                       VECTORS TO ZERO
      BSI      RFCLR    CLEAR REFRESH AREA
      BSI      VGON     REACTIVATE DISPLAY
      BSC  I   ERASE    RETURN TO CALLING PROGRAM
*
*     END-OF-SYSTEM-USAGE ROUTINE
*
VGEND DC                ENTRY POINT
      LD       INTFG    FETCH INITIALIZATION FLAG
      BSC  I   VGEND,+-    RETURN IF NOT INITIALIZED
      BSI      VGOFF    DEACTIVATE DISPLAY
      LD       INITO    FETCH ZERO AND
      STO      INTFG    TURN INITIALIZATION FLAG OFF
      LD       LV1AD    FETCH STANDARD INTERRUPT ADDRESS
      STO  L   /000C    REPLACE IN INTERRUPT LOCATION
      BSC  I   VGEND    RETURN TO CALLING PROGRAM
```

```
*      INITIALIZATION
*
INIT   DC                   THIS ROUTINE SETS INTERRUPT
*                           ADDRESSES AND SYSTEM PARA-
*                           METERS.  IT IS CALLED WHEN
*                           NECESSARY BY VGON.
*
       XIO       POWER      ACTIVATE HARDWARE
       XIO       BTRNS      TRANSFER BUFFERS FOR POWER SUPPLY
       STS   L   /0006,/40 CORE UNPROTECT TIMER
       STS   L   /000C,/40 CORE UNPROTECT LEVEL 1 ADRS
       LD    L   /000C      FETCH STANDARD INTERRUPT ADDRESS
       STO       LV1AD      SAVE INTERRUPT ADDRESS
       LDX   L1  RFRSH      FETCH REFRESH ROUTINE ADDRESS
       STX   L1  /000C      STORE AT INTERRUPT LOCATION
       LD        INIT0      FETCH 0
       STO   L   =PNTS      INITIALIZE NUMBER OF POINTS
       STO   L   =VEC       INITIALIZE NUMBER OF VECTORS
       STO   L   =VREF      INITIALIZE NUMBER OF VECTORS
*                           FOR REFRESH ROUTINE.
       BSI       RFCLR      CLEAR REFRESH AREA
       LD        INIT1      FETCH 1
       STO       INTFG      PREVENT SUBSEQUENT ENTRY
       BSC   I   INIT       RETURN TO CALLING PROGRAM
*
*      REFRESH AREA CLEARING ROUTINE
*
RFCLR  DC                   THIS ROUTINE SETS THE ENTIRE
*                           REFRESH AREA TO (0,0,BMOFF).
*
       LDX   L1  3*MXVEC      FETCH LENGTH OF REFRESH AREA
ELOOP  LD        INIT0      FETCH ZERO
       STO   L1  RFSTR-3      CLEAR FIRST WORD OF RECORD
       STO   L1  RFSTR-2      CLEAR SECOND WORD OF RECORD
       LD    L   BMOFF      FETCH BEAM OFF VALUE
       STO   L1  RFSTR-1      SET BEAM CONTROL TO OFF
       MDX    1  -3         SET UP FOR NEXT RECORD
       MDX       ELOOP      JUMP BACK TO DO NEXT RECORD
       BSC   I   RFCLR      RETURN TO CALLING PROGRAM
*
*      CONSTANTS AND VARIABLES
*
INTFG  DC        0          INITIALIZATION INDICATOR
INIT0  DC        0          0 FOR USE LOCALLY
INIT1  DC        1          1 FOR USE LOCALLY
LV1AD  DC                   SAVES STANDARD INTERRUPT ADDRESS
```

```
*
*      IOCC'S
*
       BSS   E  0       GET AN EVEN ADDRESS
POWER  DC       PWRUP   TABLE ADDRESS FOR POWER UP
       DC       /6500   ANALOG OUTPUT INITIALIZE WRITE
BTRNS  DC               DUMMY WORD FOR BUFFER TRANSFER
       DC       /6440   BUFFER TRANSFER IOCC
*
*      ANALOG OUTPUT TABLES
*      ------ ------ ------
*
PWRUP  DC       /4010   SINGLE SCAN WITH NO INTERRUPT
       DC       0       ATTACH POS AMP SUPPLY AND
       DC       32767   SET IT TO 10 VOLTS
       DC       1       ATTACH NEG AMP SUPPLY AND
       DC       -32768  SET IT TO -10 VOLTS
       DC       2       ATTACH X-OUTPUT LINE AND
       DC       0       SET IT TO 0 VOLTS
       DC       3       ATTACH BEAM CONTROL LINE AND
       DC       32767   SET IT TO 10 VOLTS
       DC       4       ATTACH Y-OUTPUT LINE AND
       DC       0       SET IT TO 0 VOLTS
       DC       5       ATTACH RUN CONTROL LINE AND
       DC       LOGCO   TURN IT OFF
       DC       6       ATTACH LOGIC SUPPLY LINE AND
       DC       16384   SET IT TO 5 VOLTS
       DC       7       ATTACH INPUT SWITCH LINE AND
       DC       LOGCO   TURN IT OFF
```

```
****************************************************
*                                                  *
*     ARRAY STORAGE USED IN DISPLAY ROUTINES       *
*                                                  *
****************************************************
*
*
*
PTSTR BSS     6*MXPNT POINT STORAGE
*
*     PTSTR STORAGE FORMAT
*     ----- ------- ------
*
* SIX-WORD RECORDS, ORGANIZED AS FOLLOWS
* 0 -- ADDRESS OF ASSOCIATED PARAMETER BUFFER
* 1 -- X-OBJECT COORDINATE
* 2 -- Y-OBJECT COORDINATE
* 3 -- Z-OBJECT COORDINATE
* 4 -- X-SCREEN COORDINATE
* 5 -- Y-SCREEN COORDINATE
*
*
*
VCSTR BSS     2*MXVEC TRAVERSAL PATH STORAGE
*
*     VCSTR STORAGE FORMAT
*     ----- ------- ------
*
* TWO-WORD RECORDS, ORGANIZED AS FOLLOWS
* 0 -- ORDINAL POINT NUMBER
* 1 -- BEAM CONTROL
*
*
*
PLTBF BSS     3*MXVEC REFRESH DATA BUFFER
*                     PLTBF IS AN EXACT IMAGE OF THE
*                     REFRESH BUFFER, SO THAT THE
*                     REFRESH BUFFER MAY BE UPDATED
*                     AT MAXIMUM SPEED
*
*     PLTBF STORAGE FORMAT
*     ----- ------- ------
*
* THREE-WORD RECORDS, ORGANIZED AS FOLLOWS
* 0 -- X-DIFFERENCE
* 1 -- Y-DIFFERENCE
* 2 -- BEAM CONTROL
```

```
RFSTR BSS     3*MXVEC REFRESH INFORMATION STORAGE
*
*      RFSTR STORAGE FORMAT
*      ----- ------- ------
*
* THREE-WORD RECORDS, ORGANIZED AS FOLLOWS
* 0 -- X-DIFFERENCE
* 1 -- Y-DIFFERENCE
* 2 -- BEAM CONTROL
*
*
*
OBPRM BSS     9*MXOBJ OBJECT PARAMETER STORAGE
*
*      OBPRM STORAGE FORMAT
*      ----- ------- ------
*
* NINE-WORD RECORDS, ORGANIZED AS FOLLOWS
* 0 -- SINE OF X-ANGLE
* 1 -- COSINE OF X-ANGLE
* 2 -- SINE OF Y-ANGLE
* 3 -- COSINE OF Y-ANGLE
* 4 -- SINE OF Z-ANGLE
* 5 -- COSINE OF Z-ANGLE
* 6 -- X-TRANSLATION
* 7 -- Y-TRANSLATION
* 8 -- Z-TRANSLATION
```

```
*       TRIGONOMETRIC TABLE
*       ------------- -----
*       THE FOLLOWING TABLE IS 5/4 PERIODS OF THE
*       SIN FUNCTION, ROUNDED TO 15 BITS AFTER
*       MULTIPLICATION BY 32767.  THE TABLE SERVES AS
*       BOTH A SINE AND COSINE TABLE, IN INCREMENTS
*       OF ONE DEGREE.  FOR BREVITY, THE TABLE IS IN
*       CHARACTER CODE.  IT OCCUPIES 450 WORDS.
*
*                   ................................
SIN     EBC         .             /   0     (     7.
        EBC         .     (     G   3              .
        EBC         .     M      . ( + +    9   .  .
        EBC         .  * )   -            R  , (   .
        EBC         . R 3           K            = .
        EBC         .= ' ' ' ' = =(=0   .     K    .
COS     EBC         .       K     .  =0=(= ' ' ' ' .
        EBC         .= =              K           3.
        EBC         . R  ( ,    R          -   ) * .
        EBC         .     .   9  + + ( .        M  .
        EBC         .                 3   G     (  .
        EBC         .   7     (     0   /          .
        EBC         .   D  9(7 4Q2 0     ZFX V T   .
        EBC         .      Q O M K O    OIYGUEUCZA2.
        EBC         .0            D 8         $    .
        EBC         . =         3     G     T      .
        EBC         .       4 5                   6.
        EBC         .         $ 3     5   =        .
        EBC         .           =   5     3 $      .
        EBC         .   6                   5 4    .
        EBC         .         T     G     3        .
        EBC         . =     $         8 D          .
        EBC         .0 A2CZEUGUIY 0  O K M O Q     .
        EBC         .    T V X ZF    O 2 4Q7 9(   D.
        EBC         .             /   0     (     7.
        EBC         .     (     G   3              .
        EBC         .     M      . ( + +    9   .  .
        EBC         .  * )   -            R  , (   .
        EBC         . R 3           K            = .
        EBC         .= ' ' ' ' = =(=0   .     K    .
*                   ................................
*
*
*
        END
```

LISTS OF REFERENCES

BIBLIOGRAPHY

B1. Newman, William M., and Sproull, Robert F. Principles of Interactive Computer Graphics. McGraw-Hill, 1973.

B2. Csuri, Charles, 'Computer Animation', Proceedings of the Second Annual Conference On Computer Graphics and Interactive Techniques, Computer Graphics, SIGGRAPH-ACM, Spring 1975.

B3. Noll, A.M. "Scanned-Display Computer Graphics, Communications of the ACM, March 1971.

B4. Ball, N. A., Foster, H. Q., Long, W. H., Sutherland, I. E., and Wigington, R. L., "A Shared Memory Computer Display System." IEEE Transactions on Electronic Computers, October 1966.

LIST OF REFERENCES

The following companies market the hardware described in the test.

R1. Intermedia Systems, Cupertino, California, 1975

R2. Lexidata Corporation, Lexington, Massachusetts, 1975

R3. Data Translation Inc., 1975

R4. Megatek, San Diego, California, 1975

R5. Adage, Inc., Boston, Massachusetts, 1969

R6. Vector General, Inc., Canoga Park, California, 1972

R7. Applications Group, Inc., Maumee, Ohio, 1975

R8. Evans and Sutherland Computer Corporation, Salt Lake City, Utah, 1974

R9. Owens-Illinois, Inc., Electro/Optical Display Business Operations, Toledo, Ohio, 1975