The adage graphics terminal

by THOMAS G. HAGAN, RICHARD J. NIXON
and LUIS J. SCHAEFER

Adage, Inc.
Boston, Massachusetts

INTRODUCTION

Interactive computer graphics applications require that the system present visual displays to the user for his interpretation and that these displays change in response to actions taken by the user as he goes about solving the problem for which he is using the system. Effectiveness of a graphics system is very dependent upon the complexity of the displays which it is able to present and the speed with which it can produce changes in the displayed image. A CRT displaying dynamic images of sufficient complexity for meaningful visual assimilation imposes a data transfer and arithmetic burden very much greater than that imposed by such conventional output media as typewriters, line printers and point plotters.

Terminals intended to provide interactive graphics capability have, therefore, quickly evolved to the point where they typically include a small computer, as shown in Figure 1, for purposes of refreshing the CRT and for some local management of the image manipulation workload imposed by the terminal.[1] For images composed of more than a few dozen vectors, however, the coordinate transformation task necessary for a truly dynamic display is too much for the limited arithmetic capability of a small digital computer. Thus, lacking special coordinate transformation capability, such systems are restricted to handling only the quasi-static display of reasonably complex images—those comprising 1,000 or more vectors. It has been suggested that the rotation and scaling tasks be thrown back upon the central computer to which the graphics terminal is connected.[2] Generalized scaling and six-degree-of-freedom geometrical coordinate transformations require about 25 arithmetic operations per vector. If the display is to be truly dynamic, these operations must be performed once per frame for each vector. At 40 frames per second, such operations upon an image composed of 1,000 vectors constitute a computational

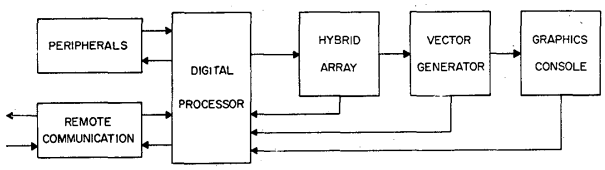

FIGURE 1—Overall organization of a graphics terminal incorporating a local digital processor

load of one million arithmetic operations per second, most of them multiplications. Central computers do exist which are capable of such performance, interleaved with the necessary high-data-rate output communication to a display device, but operational costs are prohibitive for routine computer graphics applications, and therefore this method of achieving a dynamic output display is suitable only for feasibility studies and programming research. In practice, users of systems which depend upon the central computer for coordinate transformation, or upon the digital arithmetic capabilities of a small local computer, are restricted to working in applications areas for which quasi-static displays are satisfactory, and the powerful pattern recognition abilities of the eye and brain which depend upon *motion* of a perceived image can be exercised only for very simple images composed of a few dozen vectors at most.

Organization of the AGT

An alternative approach was taken in the design of the AGT. Coordinate transformation hardware is included in the graphics terminal itself so that the transformations necessary for dynamic displays can be included among the image manipulations accomplished locally, with minimum recourse to the central computer. As a result, images composed of as many as 5,000 line segments can be displayed dynamically with arbitrary

changes in scale factor, position and rotation between successive frames.

Overall system organization conforms to the block diagram of Figure 1. The digital processor is a 30-bit word length, 2-microsecond cycle time machine with core memory sizes ranging from 4k to 32k words. Digital peripherals include card reader, line printer, high and low speed magnetic tape units, and a disk subsystem with storage capacity ranging from 3.1 million characters (1 disk drive unit attached) to 12.5 million characters (4 disk drive units attached).

Remote communications interfaces are available for telephone line communication to a central computer at speeds ranging from 1,200 baud (150 ASCII characters per second) to 40.8 Kilobaud (5.1K ASCII characters per second). Operation with the lower speed communications to the central computer is feasible because of the comparatively high degree of autonomy of the terminal.

The hybrid array shown as part of the system in Figure 1 is used for scaling and coordinate transformation of values as they are passed from core memory in the digital processor to the vector generator. The vector generator output drives horizontal, vertical and intensity inputs of the CRT contained in the graphics console. The console also contains function switches, light pen, control dials, data tablet and joystick, whose outputs are monitored by the digital processor. A photograph of a complete AGT is shown in Figure 2.

Line drawing CRT display

Most graphics terminals which incorporate a local computer use a pair of DAC's between the digital processor and the vector generator, as shown in Figure 3. Digital values extracted sequently from a display list contained in memory in the digital processor are fed through the DAC's to present analog x′ and y′ values to an analog vector generator. The vector generator, in turn, develops striaght-line segments on the face of the CRT.

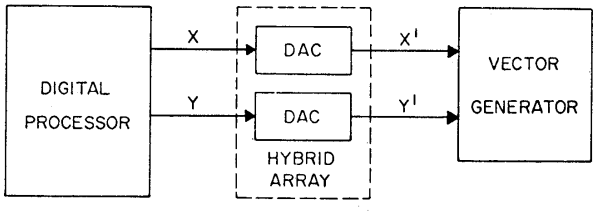

FIGURE 3—A graphics terminal in which coordinate values are passed from the digital processor to the vector generator through a pair of DAC's

Systems of this type develop pictures on the CRT by drawing a series of connected vectors, in contrast with systems which modulate intensity during a TV-like raster scan. Each vector in the series drawn on the CRT can be specified as either visible or invisible. A visible vector corresponds to a "draw" operation; an invisible vector to a "move" operation. Use of such a scheme for generating a figure such as the block letter "A" is shown in Figure 4. In this case, a total of 13 vectors is used, 11 visible "draw" vectors and 2 invisible "move vectors.

Image manipulation

Graphics terminals using the end-to-end vector drawing technique for generating pictures have usually required that the complete picture displayed on the CRT

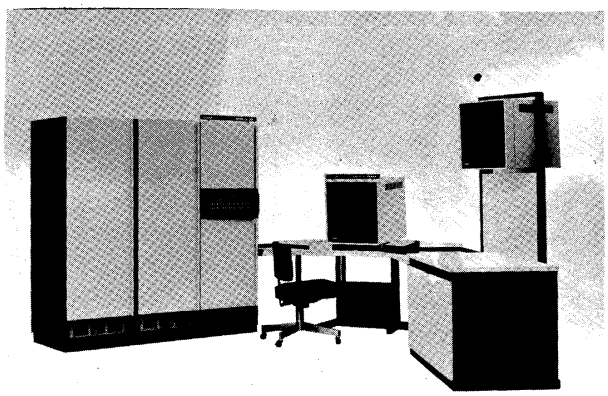

FIGURE 2—The Adage Graphics Terminal, with auxiliary high-mount viewing scope

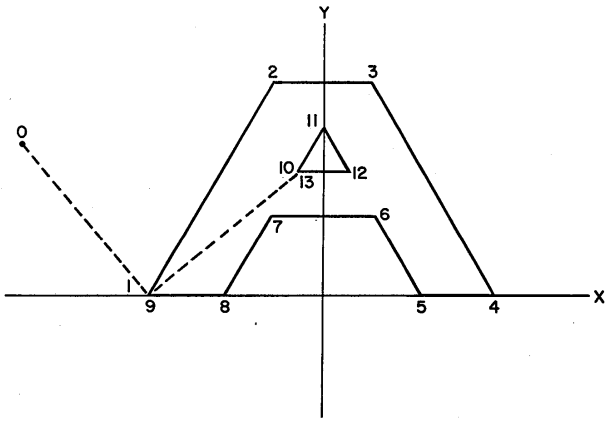

FIGURE 4—Creating a picture (here the block letter "A") with a series of connected visible and invisible vectors

be represented by a display list in memory specifying a coordinate pair for each and every vector end point contained in the picture. Any change in the picture is accomplished by computing new values for coordinates stored in the display list. Manipulation of images in such systems, therefore, consists of performing arithmetic procedures to achieve desired results. For example, to permit translation of an image element in x and y, it is necessary to implement a procedure whereby each coordinate describing the image element has added to it values that represent the increments of motion desired in x and y.

Scaling of an image element, i.e., changing its size, can be accomplished by implementing a procedure whereby two multiplications are accomplished for each coordinate describing the image element.

Structured images

A simple image is one whose description consists of a string of values specifying the end points of vectors or straight lines composing the image. A structured image is built up of sub-images by performing operations upon one or more simple images. The operations can consist of translation and scaling transformations of the type we have already described. A complete description of a complex image, therefore, includes one or more coordinate lists and also the specification of one or more transformation operations.

The picture shown in Figure 5 can be composed in several different ways. If composed as a simple, unstructured image, 25 coordinate pairs would have to be specified. Twenty of these would be necessary for the 20 "draw" vectors, and five would be necessary for the invisible "move" operations necessary to get from square to square, including an inital move to get positioned to begin the first square. A total of fifty values—25 for x and 25 for y—stored in memory would therefore be necessary to describe this image.

On the other hand, if scaling and translation operations are available, the image can be built up of sub-images, using a smaller number of values to describe the same resultant image. For example, it is possible to compose the image by storing five coordinate pairs for one square and then specifying five different scale factors and five x-y positions in order to compose the total image comprising five squares. If described this way, a total of 25 values is sufficient to describe the image: 10 x-y values for a square, 5 scale factor values, and 10 x-y position values to specify the five positions at which the squares are to be located. With no scale factor operator available, 30 values could suffice to define the image: five coordinate pairs would specify the large square, five coordinate pairs would specify the small square, and then five pairs would specify the positions for each of the five squares. This is a total of 15 coordinate pairs, or 30 values. The point of all this is that the availability of translation and scaling operations allows a complex image to be described with fewer words of memory than are necessary to describe a totally unstructured image.

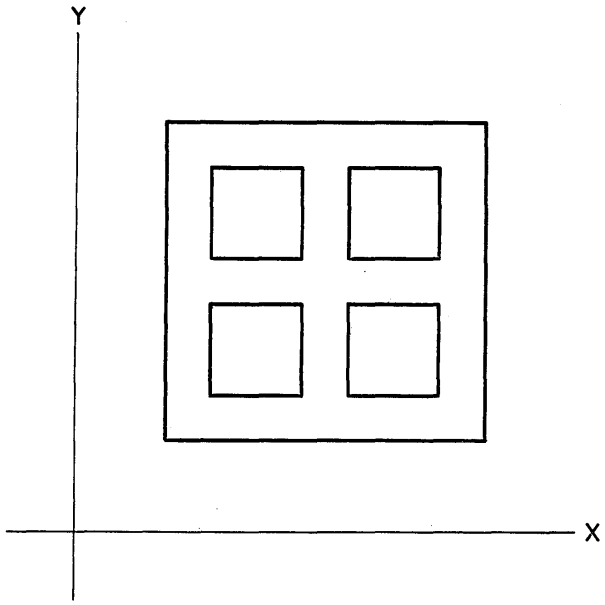

FIGURE 5—A picture resulting from an image which can be defined in terms of five squares

Even more important is the ease of introducing changes in a structured image. A change need only be made once in a sub-image to have it properly reflected in all repeated instances of the sub-image. In the present case, by changing the square to a hexagon. for example.

The translation and scaling operations necessary to permit the kind of structured image composition described above can be accomplished by implementing the following equations:

$$x' = x_d + SC \cdot x \quad (1)$$

$$y' = y_d + SC \cdot y \quad (2)$$

Hardware for image manipulation

The hybrid array shown in Figure 6 is incorporated in the AGT10 for implementing translation and scaling. It accepts digital inputs as coefficients for specifying translation and scale parameters and then, subsequently, it accepts successive x and y input values to produce appropriately translated and scaled x′ and y′ output values. The key to performing the necessary

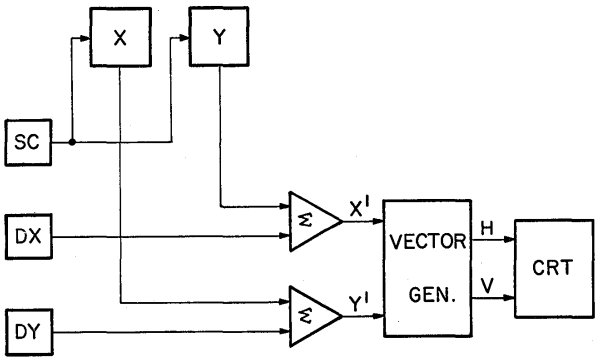

FIGURE 6—The hybrid array included in the AGT10 for scaling and positioning

multiplications is the multiplying DAC element included in the array. The operation of multiplying DAC's for image manipulation purposes has been described in a previous paper.[3]

A limited degree of structured image manipulation capability has been provided in other display systems by driving the x and y output DAC's from accumulator registers. This permits operation in an incremental vector mode, where each line segment in the string is specified in terms of incremental displacement in x and y from the end of the preceding vector. Display lists organized as strings of incremental coordinate pairs can be treated as sub-images and can be located at arbitrary positions on the screen by first loading the x and y accumulator registers with initial values corresponding to the desired displacements in x and y. Such systems are, therefore, capable of positioning sub-images whose display lists are organized in terms of incremental coordinates, but they cannot subject sub-images to operations such as scaling or rotation, and they cannot treat as a sub-image any display list composed of absolute coordinates. In the AGT, use of hybrid coordinate transformation hardware described above makes it possible to eliminate these restrictions for translation and scaling in the AGT10, intended primarily for 2-D work, and also for rotation in the AGT30 and AGT50, which are capable of handling 3-D images.

Rotation of 3-D images

Figure 7 illustrates the projection onto a two-dimensional plane of a three-dimensional figure, in this case a cube, whose axes are rotated with respect to the axis system of the viewing plane. These equations describe rotation of a three-dimensional object:

$$x' = R_{11}x + R_{12}y + R_{13}z \quad (3)$$

$$y' = R_{21}x + R_{22}y + R_{23}z \quad (4)$$

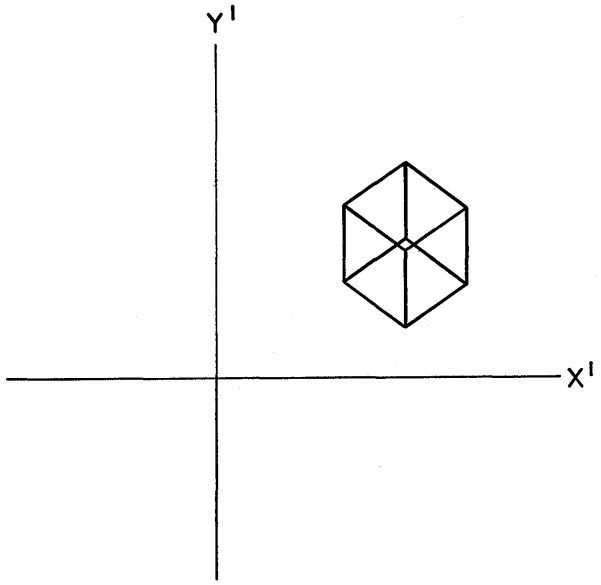

FIGURE 7—An orthogonal projection of a cube onto a two-dimensional viewing plane

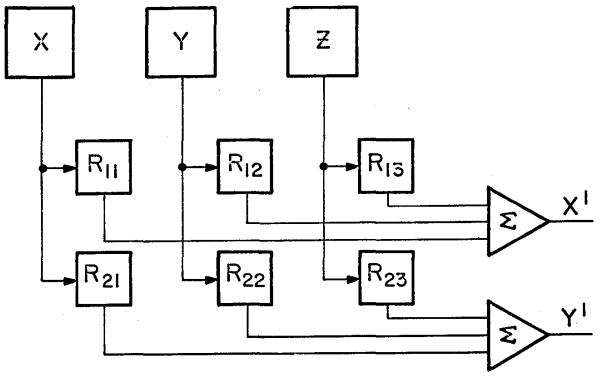

FIGURE 8—A hybrid array for rotation

A hybrid operator array appropriate for implementing these equations is shown in Figure 8. x, y and z are input coordinates describing the objects to be rotated. R11 through R23 are fixed coefficients specifying the rotation to which the input coordinates are to be subjected. x' and y' are the outputs representing the projection onto the x'y' plane of the xyz coordinates.

The hybrid array actually included in the AGT30 implements the following equation set:

$$x' = PS[x_d + SC(R_{11}x + R_{12}y + R_{13}z)] \quad (5)$$

$$y' = PS[y_d + SC(R_{21}x + R_{22}y + R_{23}z)] \quad (6)$$

$$z' = PS[y_d + SC(R_{31}x + R_{32}y + R_{33}z)] \quad (7)$$

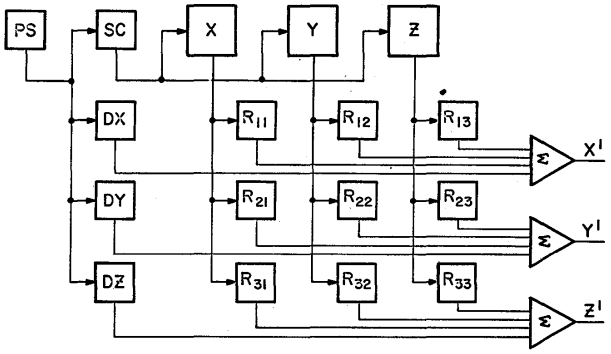

FIGURE 9—The hybrid array incorporated in the AGT30 and AGT50 for generalized scaling, translation and rotation

The block diagram of Figure 9 shows the organization of elements used to implement these equations. They are capable of providing nine multiplications for rotation, seven for scaling, and twelve summation operations, all within less than four microseconds. This is equivalent to sixteen multiples plus twelve adds in less than four microseconds, making it possible to display over 5,000 vectors subjected to arbitrary scaling and coordinate transformation at flicker-free rates.

Figure 10 shows some alphanumeric text which was generated programmatically (without the use of a character generator). Scaling and translation operations are performed by the array to place characters of appropriate size at appropriate locations on the screen.

The hybrid array is normally used to provide orthogonal rotational transformations; however, it can also be used to perform various skewed transformations. The alphanumeric text shown in Figure 11 is derived from the same programmatic font as that of Figure 10, but in this case it has been passed through the array while the array contained the set of coefficients that implemented a skewed transformation to produce the italicized, sloping text.

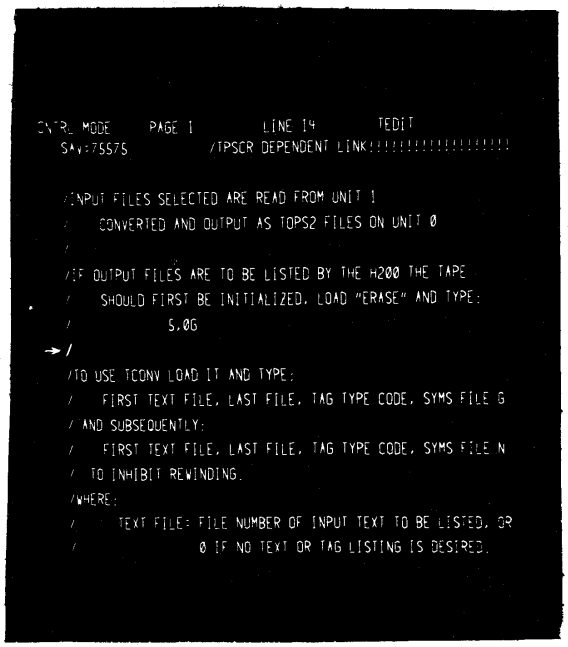

FIGURE 10—Programmatically generated text

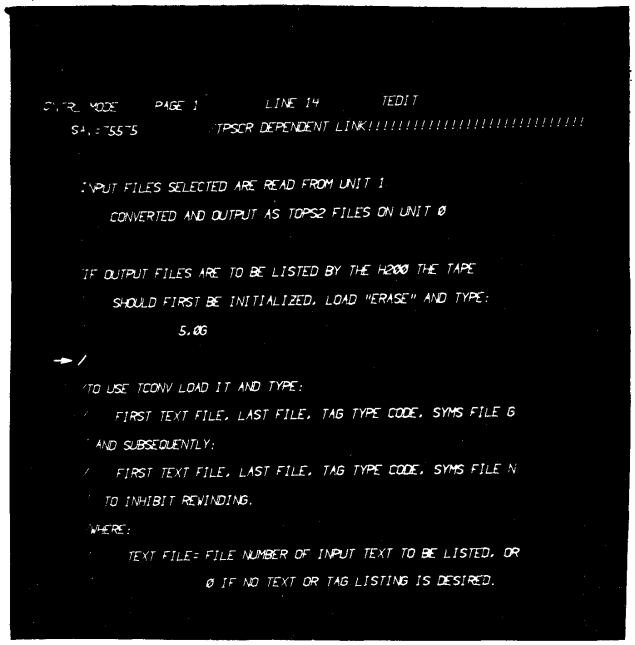

FIGURE 11—Programmatically generated text subjected to a skewed transformation

Figure 12 shows 1,000 vectors of approximately ran-

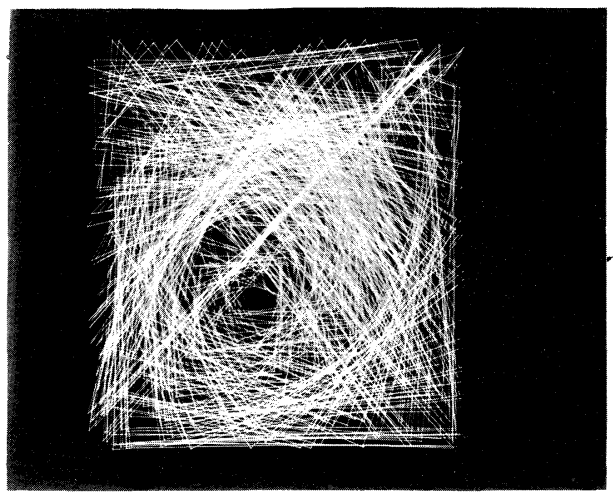

FIGURE 12—One thousand approximately random straight lines

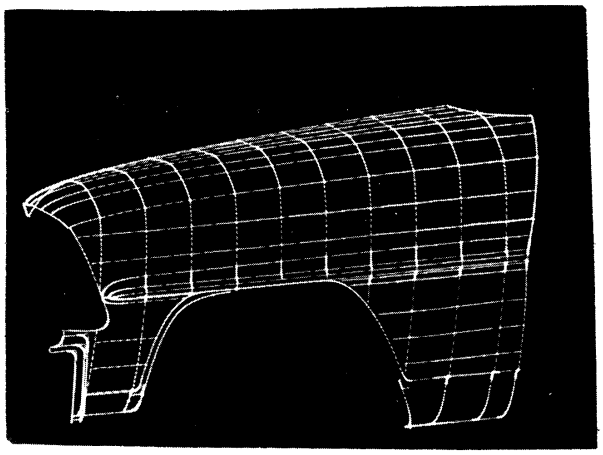

FIGURE 13—A CRT picture derived from an image described by approximately 2,500 short line segments

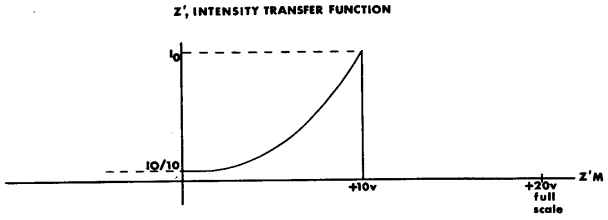

FIGURE 14—Beam intensity versus image depth for variable intensity depth cueing

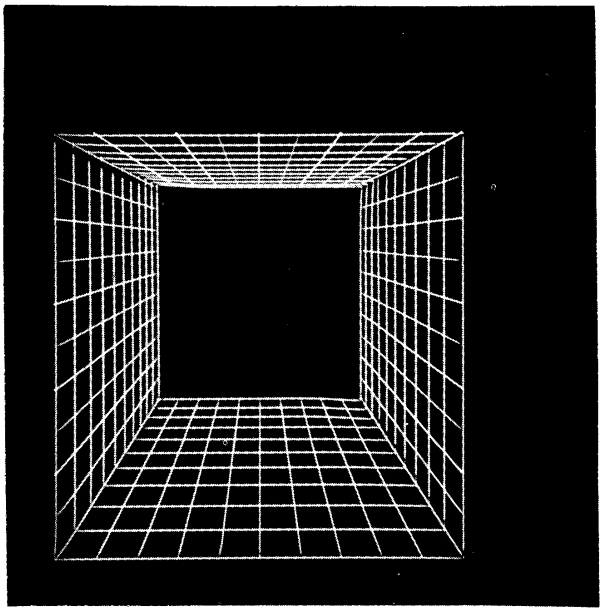

FIGURE 15—A picture without depth cueing

dom length and position. The system is capable of displaying pictures of this complexity with completely arbitrary scaling, translation and rotation.

The auto body part illustrated in Figure 13 consists of approximately 2,500 short vectors. This is about 50 percent of the limit of complexity that can be handled flicker-free by the system for images that are comprised of vectors of less than one-half inch in length.

Depth cues

The dynamic cues provided by rotating three dimensional objects and viewing their projection onto a two-dimensional plane do not completely resolve the question of what the shape looks like. Front-to-rear ambiguity occurs, as in the cube of Figure 7.

Variable intensity depth cueing has been incorporated in the AGT30 and AGT50 to help indicate picture depth by having the intensity of a line fall off as its z′ coordinate goes into the viewing screen. This is a rather natural cue as it is a common experience that objects get dimmer and less clear and textures get finer as they get farther away; depth cueing is the application of this experience to a much smaller scale. The transfer function from z′m to the intensity signal to the scope is shown in Figure 14. (Assume for now that operation takes place to the left of the +10v cutoff.) I falls exponentially as z′m decreases and is down 10:1 at z′m = 0. The exponential falloff has been found to lead to equal perceived brightness ratios for equal distance ratios. This agrees with the well-known results that the eye's response to brightness stimuli is logarithmic over a wide range.

Some illustrations of the results obtained on a CRT are shown in Figures 15 and 16. When one stares at the cube of Figure 7, it appears to flip from one orientation to another because the third dimension cue is missing. Without depth cueing, the object in Figure 15 appears to many people to be the top view of a pyramid. With depth cueing, as in Figure 16, the picture is much more clearly the perspective view of a rectangular "tunnel." Similar visual tricks occur in viewing orthogonal projections of rotating objects, and depth cueing is again found to reduce or eliminate the ambiguities which occur.

It has been suggested that equally important as a visual cue is the fact that the dimming of the line is accompanied by a narrowing of the line as well, thus giving a sort of linear perspective cue. While this perspective may provide some information about depth, the brightness depth cueing has been found to be valid when the observer is too far from the screen to distinguish differences in line width.

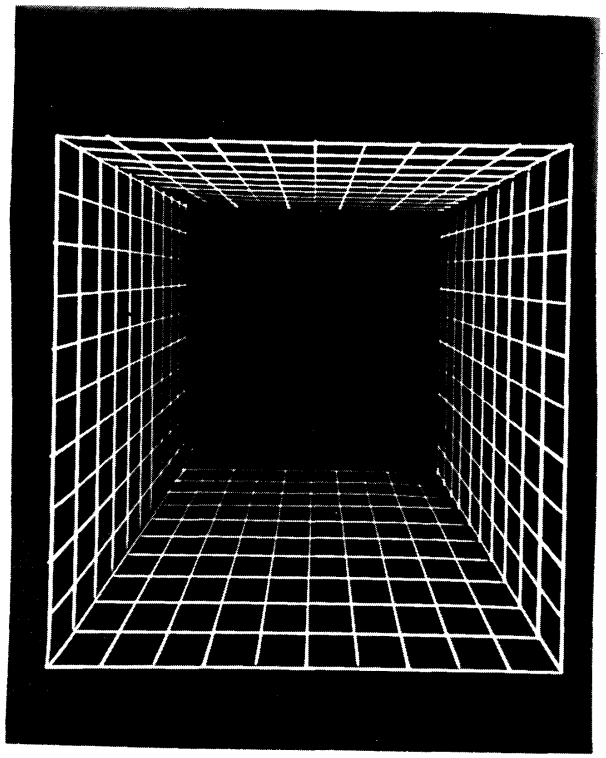

FIGURE 16—The picture of Figure 15, but with variable intensity depth cueing

Intensity windowing

The sharp cutoff at z′m = + 10v in Figure 14 leads to another useful display phenomenon which we call z-windowing. An image can be moved back and forth across the z_0 boundary. Those portions of lines whose z-values are greater than z_0 are blanked while those portions behind z_0 are displayed. Thus one can examine thin sections of an image. This has been found useful as an alternative to the removal of hidden lines as a means for "de-cluttering" images composed of many line segments.

The ability to look at sections gives some interesting flexibility in viewing complicated objects. The z_0 cutoff plane can easily be moved by a manual, interactive device such as a joystick. Moving through an object section by section can give rapid insight into the shape and nature of the object. Sectioning accompanied by rotation of the object also gives some interesting and useful effects; it enhances depth perception by cutting the front portion as one quickly learns to associate the cut plane with the front part of the object.

The transformation from z′ to z′m allows further freedom in expanding or shrinking the picture to match not only the z_0 cutoff but the depth-cue slope as well. Thus the picture can be easily moved in front or in back of z_0 without changing the data base; and the degree of depth-cue falloff can be varied at will from no perceptible change to extreme variations.

Removal of image portions in the "foreground" can also be used, with and without perspective transformations, to give the illusion of moving into the screen, i.e., objects move "behind" the observer and out of sight.

3-D windowing operator

A three dimensional windowing subsystem is available for the AGT in which upper and lower bounds can be placed (in digital registers) on x, y, and z. The vector generator then blanks whenever the beam goes beyond one of the bounds, and it also tells the program which bound was exceeded. This device finds use in a number of applications including uncluttering pictures, testing the dimensions and intersections of solids, and splitting the CRT screen up into rectangles allocated to different pictures, which can then move beyond the "edge" without encroaching upon its neighbor's display space.

Software environment

A standard software package called AMOS has been developed to support the AGT. It includes a resident monitor available in two versions, their use depending on where the program library is stored. One version is suitable for use with magnetic tape storage, the other for use with disk. The monitor provides for on-line user control of all hardware and software. Control is exercised by control statements in which program entries and names of variables use the same symbols as those in the programmer's source language. By typing the appropriate control statement, any subroutine in the program library can be loaded, linked and executed.

After initiating operation of a program, continued operator intervention is possible by use of a foreground/background mode, in which the monitor operates in the foreground on an interrupt basis while the user program operates in the background. The user can interact with the operating program in on-line fashion, interrogating memory contents and inspecting and changing parameter values. The user has access to all program entries and to all external variables and references by means of their symbolic names, as defined in the original source language programs.

A macro-assembler and a Fortran compiler permit preparation of source language programs either in symbolic machine language or ASA Basic Fortran. The macro-assembler can accept changes or additions to its own structure, so that the user who wishes to can alter it, departing from straight symbolic machine language

to devise his own problem-oriented language for particular graphics applications. Source language programs can be prepared for entry into the program library using an on-line text editor with which the user can scan selected files, display them on the CRT, and make additions, changes or deletions.

Graphics operators

Design of each of the software items mentioned briefly above is affected only slightly by the nature of the image manipulation hardware built into the AGT. A set of graphics software operators has been developed, however, whose design is much more intimately interwoven with the image manipulation capabilities of the system hardware. These graphics operators include a *display operator*, *save/retrieve* operators, a *build* operator, and a *freeze* operator. Some words about each of these is in order.

The display operator

The display operator is used to interpret data structures in core memory which represent simple or structured two- or three-dimensional images, and to display the resultant two-dimensional picture on the CRT. The data structure representing an image is *not* routinely translated to an unstructured display list for refreshing the CRT. Instead, it is left resident in core, and the display operator uses the coordinate transformation hardware to process it afresh with each successive frame to present the appropriate series of analog output values to the vector generator driving the CRT.

Each image is represented by sequences of variable length machine-word-lists called items. These may be logically linked, referenced, or looped into structures permitting hierarchical levels of processing with parametric and conditional control capabilities. The images are re-entrant and nestable for up to 16 levels of recursion.

Each item in an image describing data structure contains a command followed by an argument list. Each command has a field specifying the type of arguments contained in the argument list, and also an operation field which specifies the nature of the operation to be performed. Operations available for image description commands are of the following four types:

Element generation
Transform specification
View definition
Control operations

An Element Generating Operation specifies a particular kind of visual element in an image definition, such as a straight 3-D line or a string of packed 2-D strokes, or a string of packed characters. The associated argument list references any parameters needed for generating the elements.

Transform Specifying Operations provide for scaling, translation and rotation operations which are applied to subsequent items of the image definition. Associated arguments are used to specify values of scale factor, displacement, and angular rotation. In general, Transform Specifying Operations affect all subsequent items in the image, and their effects are cumulative. There are some Transform Specifying Operations, however, whose function is to clear the effect of all previously established transforms.

View Defining Operations use their associated arguments to specify boundaries in the x-y plane and in depth outside of which the picture is blanked. This permits viewing only an arbitrarily specified portion of the image with provision for the detection of images that extend beyond the specified bounds.

Control Operations available for inclusion in image-defining data structures permit operations such as the following:

Loop—Repetition of image items
Sub-image—Reference to another image as a single item in the present definition (the image equivalent of subroutine jumps).
End-image—Returns from current definition
Save—Saves current transform and repeat count.
Restore—Restores last-saved transform and repeat count.
Jump—Branching in an image.
Execute—Perform a computer subroutine (may be a necessary part of describing image portions which depend on values or states of real time inputs).
Jump/Execute Conditional—Image branching or program execution conditional on pen, widow, parameter values, etc.

Most image commands require values for parameters which specify their action—number of degrees of rotation, scale factor values, etc. The referencing information necessary to access these values is contained in the arguments associated with each image item. Values can be accessed via several addressing modes, including immediate, direct, multi-level indirect and structured (where the address is an index to be added to the following address reference).

The build operator

Whereas the display operator outputs data to the CRT from image descriptions in core, it is the function of the build operator to facilitate the on-line creation and manipulation of such image descriptions. Its per-

mits composition of appropriately structured image descriptions from sub-images previously stored in an image library, or in response to user-generated inputs from light pen, joystick or data tablet. Changes in the image produced by the build operator are immediately (by the next frame) reflected in the CRT output display by the display operator, providing thereby a high degree of responsiveness to user inputs.

Images are built in a parametric form, where the user is free to set his own parameters. For example, in constructing a spoked wheel from straight line segments, the coordinate values of each line segment may be of no interest. Useful parameters might be wheel size and axle position. The wheel radius and center would be defined as formal parameters. The formal parameters used by the image being built may be set to a value or set to track a value, such as the setting of a variable control dial or the joystick. Thus, the wheel might be positioned with the joystick and its radius set with a variable control dial.

For the most part, the operator uses the build operator by sitting at the console and using the light pen and function switches. A menu of build operations is always displayed on the CRT. Three separate menus are used with the build operator: a menu of control operators, a menu of the current list of formal parameters, and a menu which is a portion of the current external symbol table, including all display image items.

Save/retrieve operators

These are standard programs used for filing and retrieving image structures using local mass storage.

The freeze operator

The structure of an image created by means of the build operator can easily ve varied, and such an image will generally be economical for core memory, but it will usually take more time for processing by the display operator than the equivalent simple, unstructured image. The freeze operator permits the user, once he no longer requires the ease of varying a structured image, to transform it into a simple, unstructured image, perhaps for inclusion as a sub-image in a subsequent image. By saving a copy of the structured image prior to freezing, he can, of course, keep open the option to retrace his steps and retrieve the structured version of the image for further manipulation.

Interrupt hierarchy

Much of the operation of the AGT occurs in response to interrupts. The hierarchy of interrupt priorities is shown in Figure 17. The Display Clock, priority level 8, generates interrupts which are used to establish a frame rate of 60, 40, 30, 24 or 20 frames per second, as called for by the Display Operator.

The Vector Generator, priority level 11, generates interrupts in some of its operating modes, as a means of informing the digital processor that it is ready to accept the next vector.

CHANNEL	DEVICE
0	ARITHMETIC OVERFLOW
1	MAGNETIC TAPE
2	ALPHANUMERIC KEYBOARDS (UP TO 4)
3	MANUAL INTERRUPT FROM CONTROL PANELS
4	TELETYPE
5	COMMUNICATIONS DEVICES
6	DISK
7	CARD READER
8	DISPLAY CLOCK
9	WINDOWING OPERATOR
10	LIGHT PENS (UP TO 4)
11	VECTOR GENERATOR
12	CHARACTER GENERATOR
13	SPARE
14	PROGRAMMATICALLY INITIATED INTERRUPT
15 - 24	SPARE (LINE PRINTER, EVENT COUNTERS, ETC.)

FIGURE 17—Priority interrupt channel assignments

REFERENCES

1 M H LEWIN
An introduction to computer graphic terminals
Proceedings of the IEEE 1967 Vol 55 No 9 pp 1544–1552

2 J H MYER I E SUTHERLAND
On the design of display processors
Communications of the ACM June 1968 Vol II No 6 pp 410–414

3 T G HAGAN R TREIBER
Hybrid analog/digital techniques for signal processing applications
AFIPS Conference Proceedings 1966 Vol 28 p 379–388